RTCU IDE Users Manual

Version 9.98

logic IO
The M2M Enabler

* * * * THIS PAGE IS INTENTIONALLY LEFT BLANK * * *

Table of Contents

The RTCU Platform

1 The RTCU Platform

1.1 The RTCU Platform

The RTCU platform consists of a powerful programming environment and a broad range of RTCU devices, which offers the possibility of both digital- and analog I/O combined with wireless technologies such as Cellular, Bluetooth, WiFi and GPS. It is through these combinations that it becomes possible to solve many different tracking-, control-, regulation- and surveillance-applications.

With the RTCU platform, all necessary functions are combined in only one product, which is very easy to program in the text-based programming language "VPL". This combination opens the door to a wide array of efficient custom-tailored solutions today, and yet the same product is also able to solve tomorrow's challenges just through simple modifications to the program.

It is not necessary, however, to buy expensive customised adjustments of the product, because it is very easy to learn how to write a program that solves a specific problem on a standard PC by using the supplied development environment called RTCU IDE.

One of the major applications of the RTCU platform is remote surveillance thanks to its built-in cellular module. This module makes it possible to construct low-priced autonomous systems, which will be able to control and/or supervise different processes on distant localities without access to the wire-based telephone lines (PSTN). The system can be implemented in such a way that during normal conditions, it is able to manage control and surveillance all by itself. If an alarm should occur, the system is able to deliver an SMS-message, send a notification over a TCP/IP network, send a voice-message or send an e-mail to a predefined receiver.

The RTCU platform supports many IP based protocols, such as: The RTCU Communication Hub, sockets, MQTT, SMTP and FTP. TLS is fully supported.

A good place to start is:

- RTCU IDE Integrated Development Environment
- Quick start guide
- Examples
- Tutorial

For further and more detailed information please refer to the RTCU Execution Architecture.

1.2 RTCU Execution Architecture

1.2.1 RTCU Execution Architecture

*The **RTCU Execution Architecture** defines the functionality, capacity and performance available to the application.*

Starting in **1999** the **first generation** Execution Architecture were introduced initially without a formal name, but later known simply as the **'SMALL'** architecture. The hardware supported was with a modest 16 bit processor with only 10 KB of RAM.

In **2002** the **second generation** Execution Architecture was introduced at the same time as a full TCP/IP stack allowing GPRS support in one of the first devices in the world - the RTCU M10. This generation was named the **'LARGE'** architecture supporting up to 512 KB RAM - a significant boost over the 1st generation architecture memory constraints. Besides the support for larger programs and the over the air upgrade capability the **'LARGE'** and **'SMALL'** architecture shared the same virtual-machine architecture and did not offer sophisticated features such as multi-threading etc.

In **2006** the **third generation** Execution Architecture was offered named the **X32 architecture**. With the introduction of the **X32 architecture** the RTCU platform was ported to a powerful 32-bit ARM7 architecture and offered sophisticated multi-threading and the support for file-systems - including SD-CARD. Also introduced were advanced compiler and execution optimization technologies further boosting the power and performance of the RTCU devices.

The **fourth generation** Execution Architecture were introduced in **2014** named the **NX32 architecture**.
Besides offering much larger programs exploiting more memory, more threads, more strings, larger channels, etc the **NX32 architecture** is basically a **paradigm change** in the way a project is handled by the RTCU device.
With the **NX32 architecture** the concept of an Intellisync Project Drive is introduced, meaning that all elements of a project is represented by files in a **Intellisync** based file-system.

Finally, in **2017** the **NX32L Architecture** was introduced representing a massive leap in bringing the most sophisticated and open platform for tomorrows demanding IIoT/M2M applications. The **NX32L Architecture** rests on a powerful new hardware and software platform embracing a broad range of crucial technologies in the field of security, wireless communication standard and expandability. The **NX32L Architecture** is operating under a full and highly optimized Linux implementation.

Every new **RTCU Execution Architecture** introduced adhere to the RTCU Compatibility Manifesto.

1.2.2 The RTCU Compatibility Manifesto

To ensure device and application longevity the following compatibility manifesto has been formulated more than a **two decades** ago:

"A given Execution Architecture generation must be binary compatible for a minimum one generation backwards.
Additionally full source code compatibility for minimum two generations backwards must be ensured"

This means that for the **NX32** Execution Architecture it is possible to execute binary **X32** programs and with only re-compilation and no source code changes to execute **LARGE** architecture applications.

The RTCU Compatibility Manifesto has turned out to be a crucial element in the success of the the RTCU concept for more than 20 years - as well as for the future of the concept now transitioning into the exciting era of the **Internet of Things**.

1.2.3 Intellisync Project Drive

Introduced in the NX32 Execution Architecture a new concept for storing and maintaining all elements of a project such as application and voice data - the Intellisync Project Drive - is introduced.

The Project Drive has support for storing not just the application and voice files, but also has support for additional read-only files, e.g. for use as configuration files. All of these files are stored on a single drive, allowing for the most flexible memory usage, where e.g. a small application can be combined with a large amount of voice data, or a very large application could use most of the space for itself.

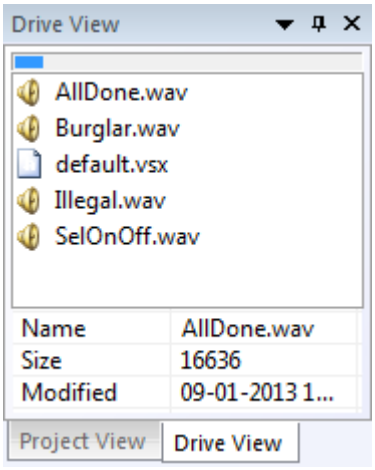
When accessed from application or the file-system window , it appears as a normal, write protected drive, using the drive letter P:, where all the ordinary file system functions work. The Project Drive does not support read out of VSX files.

The Project Drive offers a read only file system including application and voice data.

Limitations and capabilities:
NX32 Execution Architecture :
Support for a maximum of 127 files using 8.3 file names and no folders.

NX32L Execution Architecture :
Files using 8.3 file names and folders are supported with drag and drop capabilities. Supports a maximum of 1023 files and folder names are limited to 8 characters of length.

Maintaining the Project Drive is performed using the RTCU IDE or the RTCU Deployment Server utilizing a sophisticated synchronization method.
When the application is built from the RTCU IDE, it creates an .RPC file - a so called RTCU Project Container. This file contains all of the files in the project drive, including the application and the voice files. The .RPC file format is based on standard formats, allowing for files to be changed, added and removed after it has been built, e.g. by using RPCTOOL.EXE.



Intellisync

Maintaining the Project Drive is performed using a state-of-the-art **Intellisync** compaction and differencing algorithm that ensures that Project Drive synchronization is performed with minimum overhead and bandwidth usage.

In the **Intellisync** of a project with an RTCU device, the individual files are compared dynamically, and only the difference is transferred.

This means that if a single configuration file is updated on a Project Drive with many large files, only the changed file is transferred and updated, reducing the time and data needed to update it. When the size of a file increases, or a new file is added, there is a risk that it is necessary to move some of the other files to make room for it. This will be more likely to happen on a nearly full Project Drive.

1.2.4 NX32 Execution Architecture

Besides the Intellisync Project Drive the **NX32 Execution Architecture** offers considerable more resources available to the application.

Available application RAM is doubled from **192 KB to 386 KB** and at the same time application size is increased to **1 MB**.
Size of local variables in a FUNCTION is doubled from **256 to 512 bytes,** and there is considerable larger STRING resources available.

The NX32 run-time offers a boost in the multi-threading capabilities with an increase on the number of threads from 14 to 20.
A number of resources, such as the maximum number of threads, MUTEX, SEMAPHORE and CHANNEL are doubled in size/capacity.
Please refer to the RTCU multithreading data sheet for detailed information.

1.2.5 NX32L Execution Architecture

The NX32L Execution Architecture operates under Linux and is available on the latest generation of Linux based RTCU devices such as the RTCU NX and RTCU LX series.
NX32L inherits all the features and is fully backward compatible with the NX32 Execution Architecture, This means that any binary executable will run without any changes.

Using the NX32L compilation mode a long list of enhancements will be available:

- Support for Instrumented Execution (RTCU IEX).
- VPL application resource enhancements:
  - Image size increased from 1 Mbyte (NX32) to 4 Mbyte.
  - RAM size increased from 390 Kbyte (NX32) to 4 Mbyte.
  - Code size increased from 1 Mbyte (NX32) to 2 Mbyte.
- Project drive enhancements:
  - Multi-level sub-directory support with up to 1023 files.
  - The capacity is increased from 1 Mbyte to 2 Mbyte.
- Large String Support (LSS) is implicit and can not be disabled..
- Non-initialized application variables will not take up space in the execution image. This enables large data-structures without an increase in application size.
- Large array support with an unlimited number of elements and an unlimited size of each element. Only limitation is the RAM size.

The following run-time resource enhancements are also available:

- Total number of dynamic strings increased from 600 to 4096.
- Total maximum size of strings increased from 128 Kbyte to 512 Kbyte.
- Number of threads increased from 20 to 64.
- Number of Semaphores increased from 64 to 256.
- Number of Mutexes increased from 64 to 256.
- Number of Channels increased from 32 to 128.
- Total channel memory size increased from 39 Kbyte to 256 Kbyte.
- VPL thread stack size increased from 1 Kbyte to 4 Kbyte.
- Function local variables size increased from 0.5 Kbyte to 1 Kbyte.
- Prepared for VPL application dynamic memory management.

The RTCU NX32L execution architecture was introduced in **Runtime-firmware V1.40.00**.

1.2.6 EIS3

EIS3 (Enhanced Instruction Set V3) was introduced in **X32/NX32 firmware V5.00** and NX32L run-time-firmware **V1.30.00** and adds the following functionality:

- Unsigned integers with the following new datatypes: BYTE, USINT and UINT.
- Double floating-point precision support with the new DOUBLE datatype and a dual precision Math library.
- Large String Support (LSS) with up to 16384 characters dynamic STRING length and up-to 1024 characters static strings.

When any of the above features are used in a project it will automatically require EIS3. This information is available in the Project Information dialog.

1.3 RTCU Security features

1.3.1 RTCU Security Features

The security features are a collection of features in the RTCU devices that can be used to prevent unauthorized access to the device as well as protecting the data and communication against eavesdropping and tampering.

Network security with TLS
For NX32L devices, it is possible to enable network security through the use of the Transport Layer Security (TLS) protocol and X509 certificates.
The TLS protocol enables Client-Server applications to use encryption when communicating across a network.
TLS v1.2, v1.1 and 1.0 are supported

The SMTP, MQTT and raw TCP/IP sockets can be configured to use TLS.

Encrypted RTCU Communication Hub communication
TLS is fully supported using the RTCU Communication Hub and is the preferred approach for safe communication.
For non NX32L devices the communication can be encrypted by configuring an encryption key with the RTCU Communication Hub Settings dialog or sockSetGWParm.

Data encryption
Data encryption is achieved with the Advanced Encryption Standard (AES).
The AES is a specification for the encryption of electronic data established by the U.S. government.
A key length of 128 bit is supported. See rdEncrypt128 and rdDecrypt128 for more information.

Encrypted include files
To make it possible to share a parts of a project without providing access to the actual code, encrypted include files can be used.

Access restriction
It is possible to use passwords to restrict the access to the RTCU device.
Without the correct password, only type of device, the firmware version, serial number and debug messages can be read from the device.
For more information about passwords, see boardSetPassword and boardSetPasswordAlt.

1.3.2 Certificates

1.3.2.1 Certificates

A certificate is an electronic document, often in PEM format, which is used to prove the identity of a public key.

The certificate includes information about the key, the identity of the certificate holder (called the subject), and the digital signature of an entity that has verified the contents of the certificate (called the issuer).
In Transport Layer Security (TLS) the subject of a certificate is typically a computer or similar device.

The certificates are identified by the name that is provided when they are imported through the certificate dialog.
This name must be used when referring to a specific server or client certificate (e.g. when calling soConfigTLS).

Certificates which have an encryption key embedded (TLS server and TLS client certificates) may require a password when used.

The following three types of certificates are supported:

Root certificate
A root certificate is a self-signed certificate that identifies a root certificate authority (CA).
Root certificates are used to sign intermediate certificates, which in turn are used to sign TLS server certificates and TLS client certificates.
To verify that a TLS server or client certificate is valid, it is necessary to have the matching root certificate installed.

When validating external client/server certificates, all the installed root certificates are checked, regardless of their names.

TLS server certificate
In TLS, a server is required to present a certificate as part of the initial connection setup.
A client connecting to that server will verify at least two things:

1. The subject of the certificate matches the host-name to which the client is trying to connect.
2. The certificate is signed by a trusted CA.

Host names are listed in the Subject Alternative Name Field of the certificate.

If the server certificate is not signed directly by a root certificate, then it must include all the intermediate certificates that was used to sign it as a certificate chain.

TLS client certificate
Client certificates are less common than server certificates, and are used to authenticate the client connecting to a TLS server, for instance to provide access control.

If the client certificate is not signed directly by a root certificate, then it must include all the intermediate certificates that was used to sign it as a certificate chain.

1.3.2.2 PEM file format

The file format used for sending and storing certificates is the de facto standard PEM format. This format is designed to be safe for inclusion in both ASCII and rich-text documents, such as emails. This means that it is possible to copy and paste the content of a PEM file to another document and back.

The PEM format is the standard format for OpenSSL and many other SSL tools.
Some sources refer to the format as Base64 encoded X.509.

The PEM format defines 3 elements:
1. A one-line header, consisting of "-----BEGIN", a label, and "-----".
2. Base64 encoded binary data.
3. A one-line footer, consisting of "-----END", a label, and "-----"

The label determine the type of message encoded. Common labels include "CERTIFICATE" and "PRIVATE KEY".

For example, a certificate would be stored as:

```
-----BEGIN CERTIFICATE-----
MIICLDCCAdKgAwIBAgIBADAKBggqhkjOPQQDAjB9MQswCQYDVQQGEwJCRTEPMA0G
A1UEChMGR251VExTMSUwIwYDVQQLExxHbnVUTFMgY2VydGlmaWNhdGUgYXV0aG9y
aXR5MQ8wDQYDVQQIEwZMZXV2ZW4xJTAjBgNVBAMTHEdudVRMUyBjZXJ0aWZpY2F0
ZSBhdXRob3JpdHkwHhcNMTEwNTIzMjAzODIxWhcNMTIxMjIyMDc0MTUxWjB9MQsw
CQYDVQQGEwJCRTEPMA0GA1UEChMGR251VExTMSUwIwYDVQQLExxHbnVUTFMgY2Vy
dGlmaWNhdGUgYXV0aG9yaXR5MQ8wDQYDVQQIEwZMZXV2ZW4xJTAjBgNVBAMTHEdu
dVRMUyBjZXJ0aWZpY2F0ZSBhdXRob3JpdHkwWTATBgcqhkjOPQIBBggqhkjOPQMB
BwNCAARS2I0jiuNn14Y2sSALCX3IybqiIJUvxUpj+oNfzngvj/Niyv2394BWnW4X
uQ4RTEiywK87WRcWMGgJB5kX/t2no0MwQTAPBgNVHRMBAf8EBTADAQH/MA8GA1Ud
DwEB/wQFAwMHBgAwHQYDVR0OBBYEFPC0gf6YEr+1KLlkQAPLzB9mTigDMAoGCCqG
SM49BAMCA0gAMEUCIDGuwD1KPyG+hRf88MeyMQcqOFZD0TbVleF+UsAGQ4enAiEA
l4wOuDwKQa+upc8GftXE2C//4mKANBC6It01gUaTIpo=
-----END CERTIFICATE-----
```

Multiple messages

A PEM file may contain multiple messages, which is used among other things to provide a certificate chain or to combine the certificate and the private key in a single file.
To make a PEM file with a valid certificate chain, each certificate in the file must be followed by a certificate that certifies it until the root certificate is reached (see the certificate_list in RFC 4346, section 7.4.2).

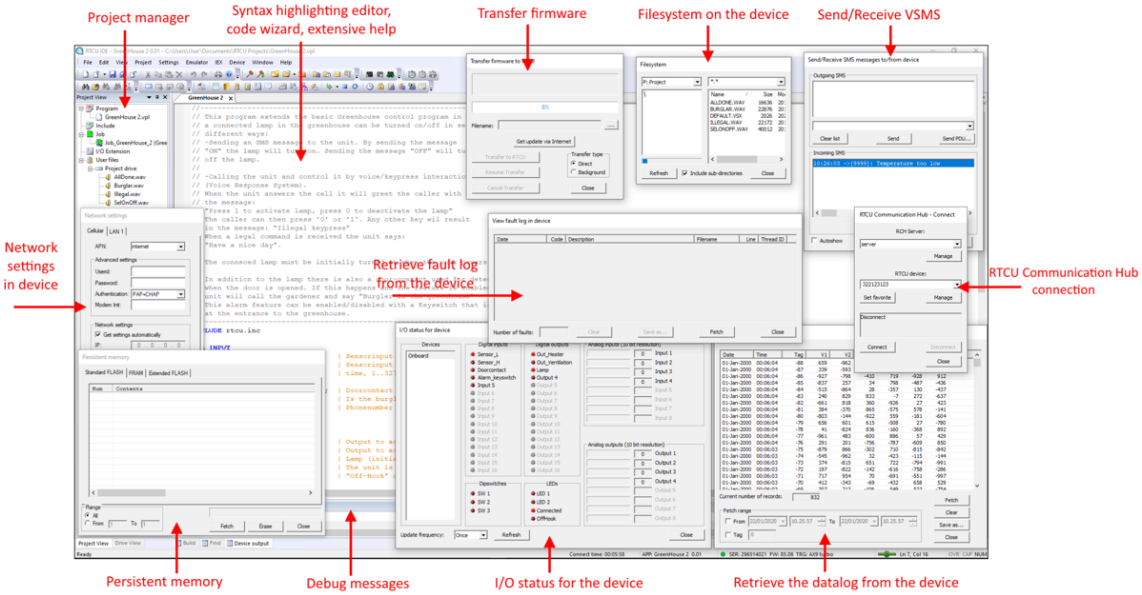
RTCU Integrated Development Environment

2 RTCU Integrated Development Environment

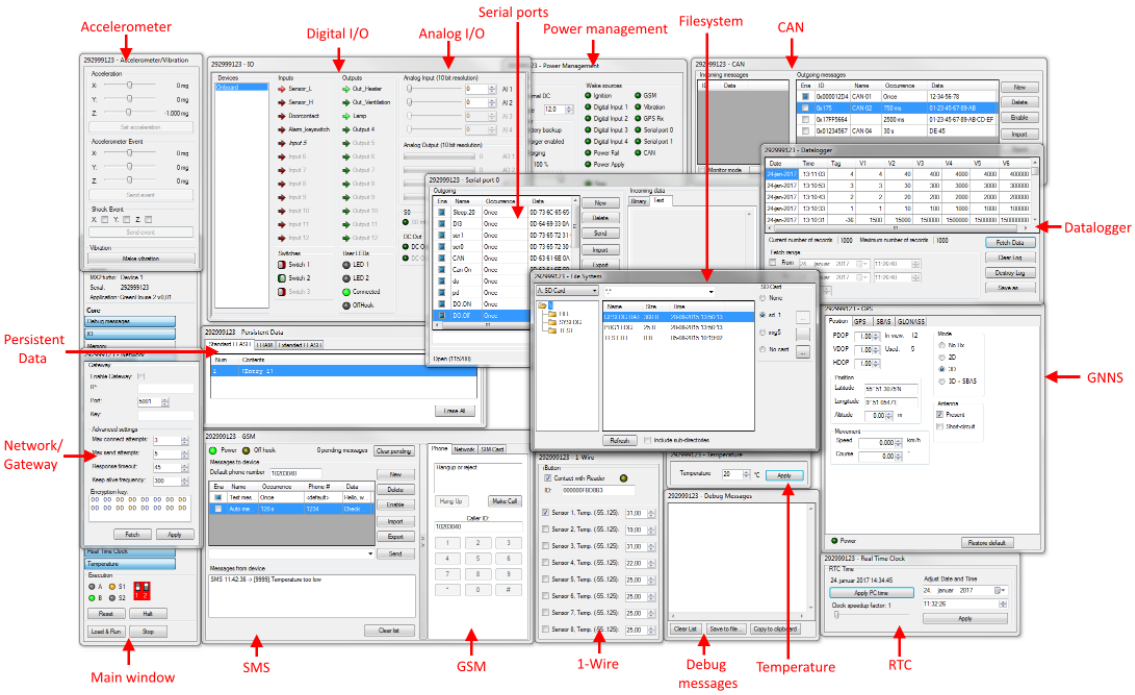
2.1 RTCU Integrated Development Environment

The RTCU IDE (Integrated Development Environment) is a very comprehensive development environment with all the features necessary to develop both simple and very advanced M2M/IoT applications.
With the tightly coupled RTCU Emulator it is possible to complete the entire development cycle from the same powerful environment before final deployment.
The two screen shots below shows a subset of the huge range of windows and features available in the RTCU IDE and RTCU Emulator.

The first screen shot shows the RTCU IDE with many windows open. You will be able to see some of the many features available:



The second screen shot shows some of the windows of the advanced RTCU Emulator, available as a separate download:



Some of the features of the RTCU IDE environment are listed below:

- **Complete Development Environment**
  There is no need for other programs, seeing as all tasks are performed from within the RTCU IDE environment. The built-in Project Control handles all aspects of a project - source code, jobs, voice messages, and the like.
- **Integration of the RTCU Emulator**
  In- and outputs, phone, SMS, network, real time clock etc. can all be emulated. Inputs can be controlled, the statuses of all outputs can be seen, full interaction with the phone is available, as are emulated sending and receiving of SMS messages, network communication, and so on. This enables you to write and develop an application and, on top of that, emulate the complete application before it is transferred to an RTCU device. The transfer of a project to an RTCU device can be done either by using a programming cable, by way of the RTCU Communication Hub, or by using a modem (CSD) connection.
- **Syntax highlighting editor**
  All keywords, comments, variables etc, are automatically shown in different colors. This greatly enhances the readability of the program.
- **Built-in help**
  Help on all built-in functions, syntax of the programming language etc. is available online from within the RTCU IDE.
- **Code generator for the VPL language and other built-in functions**
  This helps the developer remember infrequently used functions.
- **Built-in uploading function**
  With the click of a mouse button, all components of an application, like Jobs, text- and voice messages, can be transferred to an RTCU device.
- **Built-in upgrade function for Firmware**
  The firmware of an RTCU device can be upgraded from within the RTCU IDE when new versions become available - directly fetched from the internet.

A good place to start is:

- The RTCU Platform
- Anatomy of a VPL program
- The VPL Programming Language
- Menu items in the RTCU IDE Program
- Windows
- RTCU Emulator
- Reserved words
- Quick start guide
- Examples
- Tutorial

2.2 Accelerator keys

Accelerator keys for Help:

Key	Function
F1	Open Help for Dialog or Menu.
<SHIFT>+F1	Open Help for selected.
<CTRL>+F1	Open Help.
<ALT>+F1	Open Help for word under cursor.

Accelerator keys for editor:

Key	Function
<CTRL>+<KEYPAD +>	Magnify text size.
<CTRL>+<KEYPAD ->	Reduce text size.
<CTRL>+<KEYPAD />	Restore text size to normal.
<SHIFT>+<DEL>, <CTRL>+X	Cut selected text.
<CTRL>+<INS>, <CTRL>+C	Copy selected text.
<SHIFT>+<INS>, <CTRL>+V	Insert text.
<CTRL>+A	Select all.
<CTRL>+Z	Undo change.
<CTRL>+Y	Redo change.
<CTRL>+F	Find.
<SHIFT>+<CTRL>+F	Find in Project.
F3	Find next.
<SHIFT>+F3	Find previous.
<CTRL>+F3	Mark text and find next.
<SHIFT>+<CTRL>+F3	Mark text and find previous.
<SHIFT>+<CTRL>+R	Find file.
<CTRL>+H	Replace.
<CTRL>+L	Go to line.
<CTRL>+<SHIFT>+T	Copy line.
<CTRL>+D	Duplicate selected.
<CTRL>+T	Transpose line with previous.
<CTRL>+<UP>	Scroll up.
<CTRL>+<DOWN>	Scroll down
<CTRL>+<LEFT>	Previous word.
<CTRL>+<SHIFT>+<LEFT>	Extend selection to previous word.
<CTRL>+<RIGHT>	Next word.
<CTRL>+<SHIFT>+<RIGHT>	Extend selection to next word.
<SHIFT>+<END>	Extend selection to end of line.
<SHIFT>+<HOME>	Extend selection to start of line.
<CTRL>+<SHIFT>+<END>	Extend selection to end of file.
<CTRL>+<SHIFT>+<HOME>	Extend selection to start of file.
<CTRL>+<END>	Go to end of file.
<CTRL>+<HOME>	Go to start of file.
<CTRL>+<BACK>	Delete to start of word.
<CTRL>+<SHIFT>+<BACK>	Delete to start of line.
<CTRL>+<DEL>	Delete to end of word.
<CTRL>+<SHIFT>+<DEL>	Delete to end of line.
<CTRL>+N	New file (not included in project).
<CTRL>+O	Open file.
<CTRL>+P	Print.
<CTRL>+S	Save file.
F5	Open the RTCU Emulator
F7	Build project.
<ALT>+F7	Re-build project.
F8	Transfer application to RTCU.
<SHIFT>+F8	Quick transfer application to RTCU.
F11	Next window.

F12	Previous window.

Accelerator keys for bookmarks:

Key	Function
F2	Go to next bookmark.
<SHIFT>+F2	Go to previous bookmark.
<CTRL>+F2	Toggle bookmark.
<CTRL>+<SHIFT>+F2	Clear all bookmarks.
<CTRL>+0	Go to bookmark 0.
<CTRL>+1	Go to bookmark 1.
<CTRL>+2	Go to bookmark 2.
<CTRL>+3	Go to bookmark 3.
<CTRL>+4	Go to bookmark 4.
<CTRL>+5	Go to bookmark 5.
<CTRL>+6	Go to bookmark 6.
<CTRL>+7	Go to bookmark 7.
<CTRL>+8	Go to bookmark 8.
<CTRL>+9	Go to bookmark 9.
<CTRL>+<SHIFT>+0	Toggle bookmark 0.
<CTRL>+<SHIFT>+1	Toggle bookmark 1.
<CTRL>+<SHIFT>+2	Toggle bookmark 2.
<CTRL>+<SHIFT>+3	Toggle bookmark 3.
<CTRL>+<SHIFT>+4	Toggle bookmark 4.
<CTRL>+<SHIFT>+5	Toggle bookmark 5.
<CTRL>+<SHIFT>+6	Toggle bookmark 6.
<CTRL>+<SHIFT>+7	Toggle bookmark 7.
<CTRL>+<SHIFT>+8	Toggle bookmark 8.
<CTRL>+<SHIFT>+9	Toggle bookmark 9.

Accelerator keys for folding:

Key	Function
<ALT>+C	Fold the current.
<ALT>+0	Fold all levels.
<ALT>+1	Fold level 1.
<ALT>+2	Fold level 2.
<ALT>+3	Fold level 3.
<ALT>+4	Fold level 4.
<ALT>+5	Fold level 5.
<ALT>+6	Fold level 6.
<ALT>+7	Fold level 7.
<ALT>+8	Fold level 8.
<ALT>+<SHIFT>+C	Unfold the current.
<ALT>+<SHIFT>+0	Unfold all levels.
<ALT>+<SHIFT>+1	Unfold level 1.
<ALT>+<SHIFT>+2	Unfold level 2.
<ALT>+<SHIFT>+3	Unfold level 3.
<ALT>+<SHIFT>+4	Unfold level 4.
<ALT>+<SHIFT>+5	Unfold level 5.
<ALT>+<SHIFT>+6	Unfold level 6.
<ALT>+<SHIFT>+7	Unfold level 7.
<ALT>+<SHIFT>+8	Unfold level 8.

2.3 Menu items

2.3.1 Menu items

The following pages describe all the different menu items available in the RTCU IDE. Some of the menu items also have a shortcut key, which is shown under each of the menu items. A number of the menu items also have a corresponding button on one of the toolbars. Please see below.

- File
- Edit
- View
- Project
- Settings
- Emulator
- IEX
- Device
- Window
- Help

This is the Standard toolbar. It has buttons for the following commands:

- File: New file
- File: Open file
- File: Save
- File: Save all
- File: Close

- Edit: Cut
- Edit: Copy
- Edit: Paste
- Edit: Delete

- Edit: Undo
- Edit: Redo

- File: Print
- Help: About

This is the Find toolbar. It has buttons for the following commands:

- Find
- Find in project
- Find: Next
- Find: Previous
- Find: Replace

This is the Bookmarks toolbar. It has buttons for the following commands:

- Bookmark: Toggle
- Bookmark: Next
- Bookmark: Previous
- Bookmark: Clear All

This is the Project toolbar. It has buttons for the following commands:

- Project: Build
- Project: Build All
-
- Project: New
- Project: Open
- Project: Close

- Project: Settings
- Project: Information
- Project: Send via Email
- Project: Compress

This is the Device toolbar. It has buttons for the following commands:

- Device: RTCU Communication Hub - Connect

- Device: Persistent data
- Device: Datalogger
- Device: File system
- Device: Fault log
- Device: IO Monitor
- Device: SMS messages

- Device: Statistics
- Device: System Information
- Device: Network Information
- Device: Power Information

- Device: Synchronize
- Device: Halt
- Device: Reset

- Device: Adjust Clock
- Device: Set Password
- Device: Log / Debug
- Device: Power / Battery
- Device: GSM Options
- Device: Certificates

This is the Emulator toolbar. It has buttons for the following commands:

- Emulator: Launch
- Emulator: Close
- Emulator: Connect

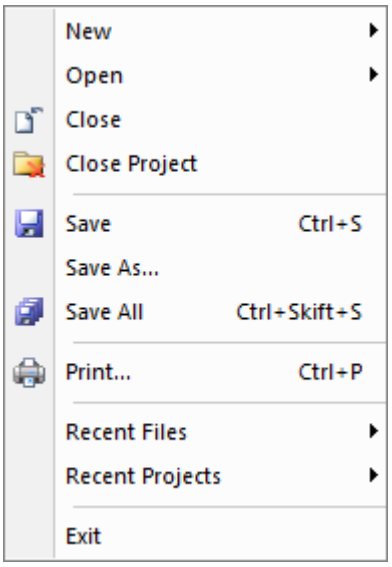
This is the IEX toolbar. It has buttons for the following commands:

- IEX: Launch
- IEX: Close
- IEX: Connect

2.3.2 Menu item: File

2.3.2.1 Menu item: File

The "File" menu item is meant for handling files. From this menu it is possible to create, open, close and save files and projects, as well as print files.



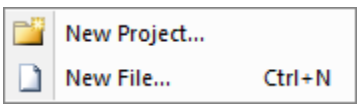
The individual items:

- New
- Open
- Close
- Close Project
- Save
- Save As...
- Save All
- Print...
- Recent Files
- Recent Projects
- Exit

2.3.2.2 File: New

2.3.2.2.1 File: New

By using the "File - New" menu, it is possible to create new files and projects.



The individual items:

- New Project
- New File

2.3.2.2.2 New: New Project

"New Project" will close the current project (if one is open). It will then present you with a file dialog, where you can type the name of a new project. When you press the "Save" button in the dialog, an empty project file is created.

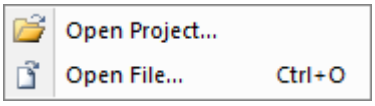
2.3.2.2.3 New: New File

This command creates a new empty file. When creating VPL files, it is generally better to use the New Program command in the project tree.

2.3.2.3 File: Open

2.3.2.3.1 File: Open

By using the "File - Open" menu, it is possible to open existing files and projects.



The individual items:
- Open Project
- Open File

2.3.2.3.2 Open: Open File

This command opens a file. You are presented with a file selection dialog that lets you select the file you want to open.

2.3.2.3.3 Open: Open Project

The "Open Project" command will close the current project (if one is open) and show you a dialog, where you can select a project file. Project files all have the ".PRJ" as their file extension.

2.3.2.4 File: Close

"Close" closes the active file.

2.3.2.5 File: Close Project

"Close Project" closes the active project.

2.3.2.6 File: Save

The "Save" command saves the current file.

2.3.2.7 File: Save as

The "Save As" command saves the current file. But unlike the Save command, you will always be presented with a dialog, from which you can name the file.

2.3.2.8 File: Save All

The "Save All" command saves all modified files.

2.3.2.9 File: Print

By using the "Print" command, you are able to print the current file. You will be presented with a normal print dialog, where you can adjust the specific features of your printer. The printout of files will be in full color mode (provided your printer is capable of this). Your file will be syntax highlighted on the printout in the exact same colors as used in the editor.

2.3.2.10 File: Recent files and projects

These menus show a list of the most recently opened files and projects in the RTCU IDE program. Selecting and clicking one of these files or projects will open it.

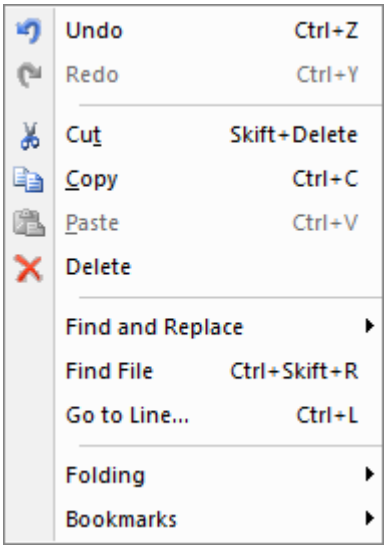
2.3.2.11 File: Exit

The "Exit" command closes the RTCU IDE program. If there are any open files that have not been saved yet, you will be asked if you want to save these before the program closes.

2.3.3 Menu item: Edit

2.3.3.1 Menu item: Edit

The "Edit" menu items are commands used to manipulate text in the current file. From this menu, you can "Cut" and "Copy", "Find" text, "Replace" text, and place bookmarks in the file (to ease navigation in large files).



The individual items:

- Undo
- Redo

- Cut
- Copy
- Paste
- Delete

- Find and Replace
- Find file
- Go to Line

- Folding
- Bookmarks

2.3.3.2 Edit: Undo

This command reverses the last change you made to the file

2.3.3.3 Edit: Redo

This command will redo the last performed action in the text.

2.3.3.4 Edit: Cut

This command will remove the selected text and put the selected text into the clipboard, where it will replace any text.

2.3.3.5 Edit: Copy

This command will copy the selected text into the clipboard, where it will replace any text.
If no text is selected in the current file, the "Copy" command will simply copy the contents of the line where the cursor is positioned and place the text into the clipboard.

2.3.3.6 Edit: Paste

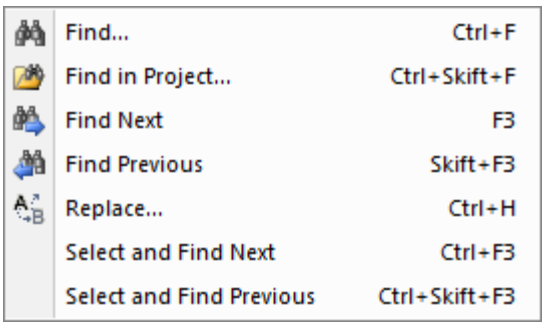
If there is any text in the clipboard, this command will copy this text and place it at the current cursor position.

2.3.3.7 Edit: Delete

This command will delete the selected text. It will not be copied into the clipboard. Use the Edit - Cut command instead if you want it to be.

2.3.3.8 Edit: Find and Replace

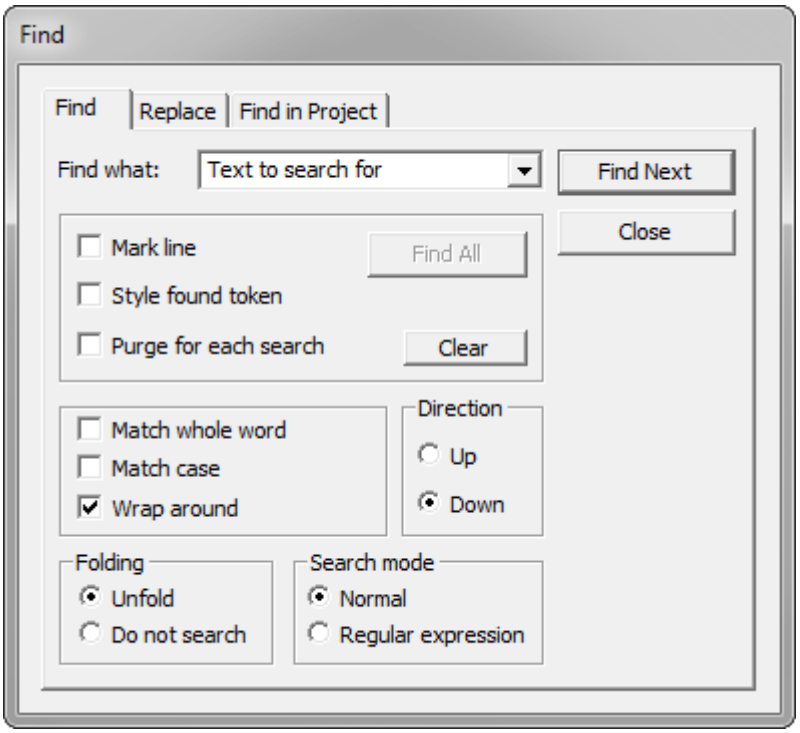
2.3.3.8.1 Edit: Find and Replace

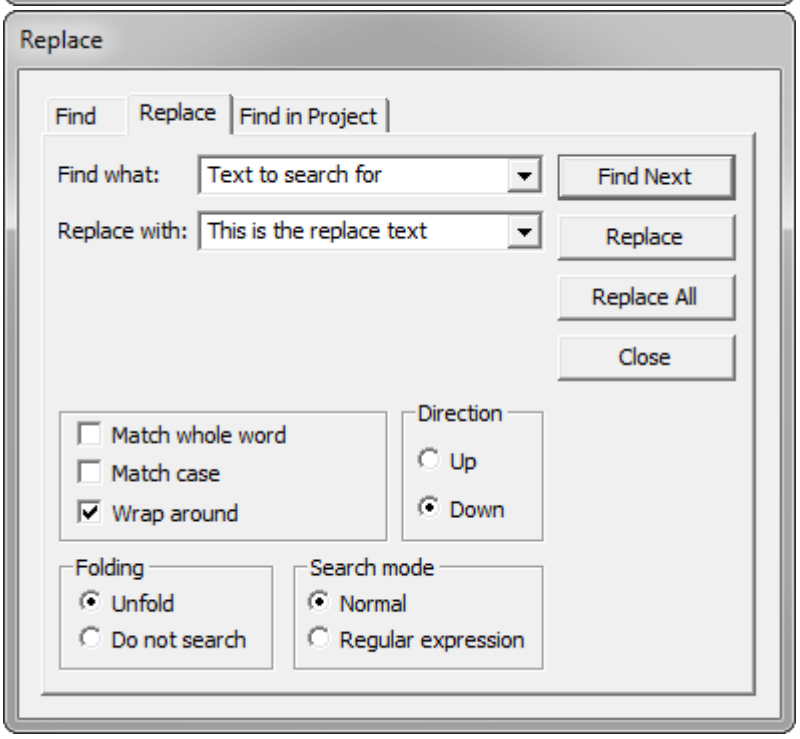


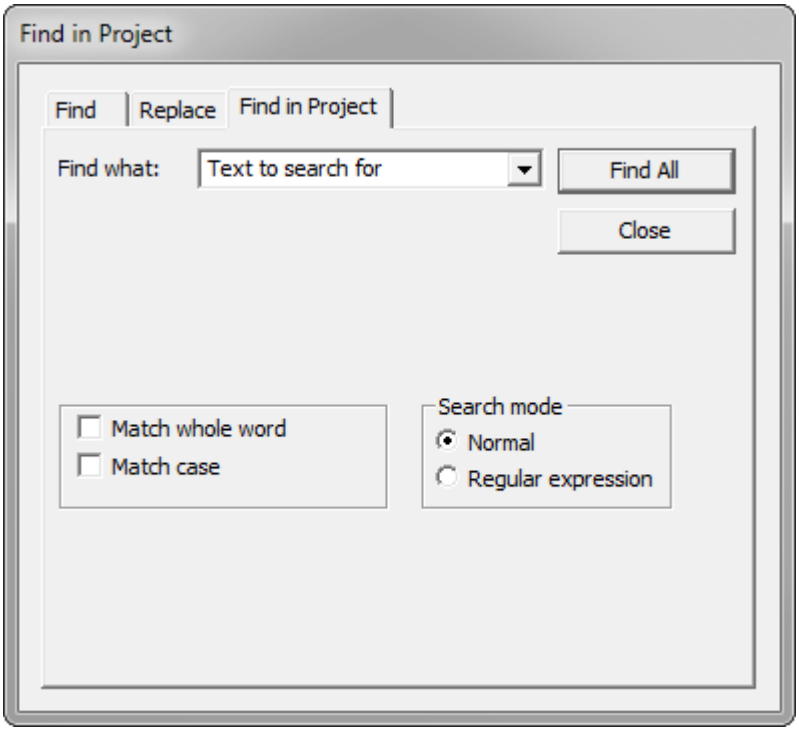
The individual items:

- Find
- Find in project
- Find Next
- Find Previous
- Replace
- Select and Find Next
- Select and Find Previous

2.3.3.8.2 Edit: Find / Replace / Find in Project







Using this command, you are able to search for text in the current file, as well as in all the files in the project.

The parameters are:

Find what
The text to search for.
The drop down list contains the text from the last 10 searches.

Replace with
The text that replaces the search text.
The drop down list contains the text from the last 10 replaces.

Find next
Search the file for the next instance of the search text.

Replace
Replaces the selected text and searches for the next instance in the file.

Replace All
Replace all instances of the search text in the file.

Mark line
Select this option if the search should mark the line of all instances in the file.
The search-mark is located in the "Bookmark" column of the file.
This option is only used with the "Find All" button.

Style found token
Select this option if the search should highlight all instances in the file.
The found tokens are highlighted with a light-red background.
This option is only used with the "Find All" button.

Purge for each search
Select this option if the Search mark and Token highlight should be removed when a new search is started.

This option is only used with the "Find All" button.

Find All
Find and mark all instances of the search text in the file.
The "Mark line" and "Style found token" options determine how instances are marked.

Clear
Clear all Search marks and Token highlighting from the file.

Match whole word
Select this option if the search should look for whole words only in the text.

Match case
Select this option if the search should look for matching case only.

Wrap around
Select this option if the search should continue around the end of the file (either top or bottom).

Direction
The direction of the search. "Up" towards the top of the file or "Down" towards the bottom.

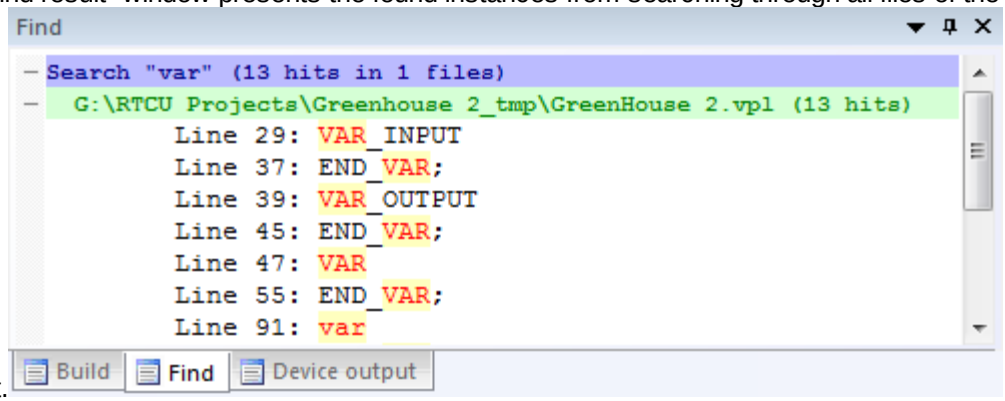
Folding
The policy of searching in folded blocks.
Unfold will expand the folded block if an instance is found within.
"Do not search" will ignore the folded block.

Search mode
The search mode determines what the search text contains - Normal text or Regular expression.
It is not possible to search back in the file when using regular expression.
In a regular expression, special characters interpreted are:

.	Matches any character.
(	This marks the start of a region for tagging a match.
)	This marks the end of a tagged region.
\n	Where n is 1 through 9 refers to the first through ninth tagged region when replacing. For example, if the search string is Fred\([1-9]\)XXX and the replace string is Sam\1YYY, when applied to Fred2XXX, it would generate Sam2YYY.
\<	This matches the start of a word using Scintilla's definitions of words.
\>	This matches the end of a word using Scintilla's definition of words.
\x	This allows you to use a character x that would otherwise have a special meaning. For example, \[would be interpreted as [and not as the start of a character set.
[...]	This indicates a set of characters. For example, [abc] means any of the characters a, b or c. You can also use ranges, for example [a-z] for any lower case character.
[^...]	The complement of the characters in the set. For example, [^A-Za-z] means any character except an alphabetic character.
^	This matches the start of a line (unless used inside a set, see above).
$	This matches the end of a line.
*	This matches 0 or more times. For example, Sa*m matches Sm, Sam, Saam, Saaam and so on.
+	This matches 1 or more times. For example, Sa+m matches Sam, Saam, Saaam and so on.

Find result dialog

The "Find result" window presents the found instances from searching through all files of the project.



Double-click on a line to open the file and select the found instance.

2.3.3.8.3 Edit: Find Next

This command will continue the search defined in Find.

2.3.3.8.4 Edit: Find Previous

This command will continue the search defined in Find, but it will also find the previous occurrence of the search text.

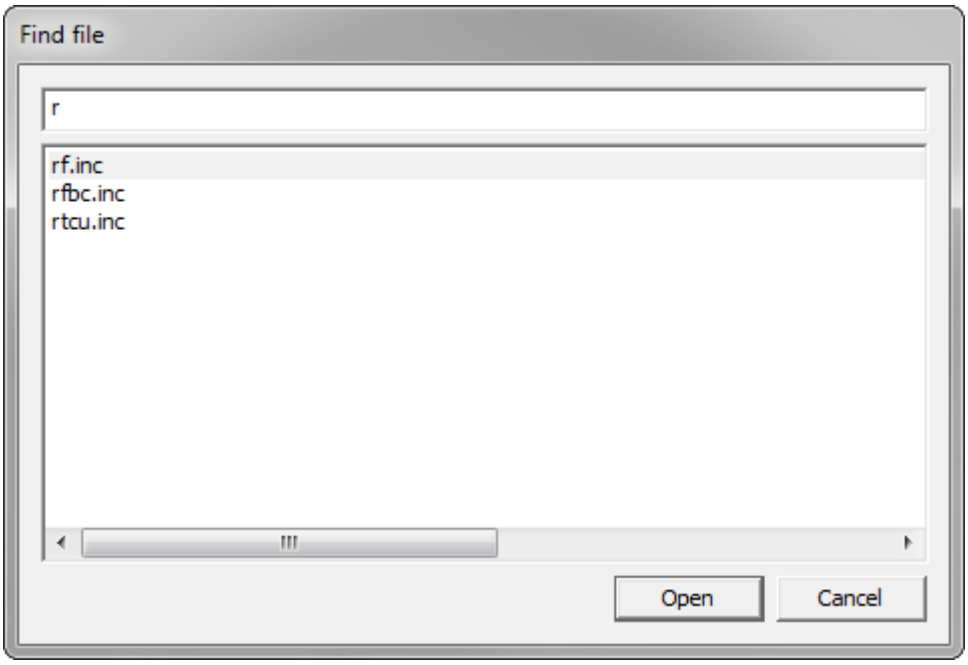
2.3.3.8.5 Edit: Select and Find Next

This command will find the next occurrence of the selected text.

2.3.3.8.6 Edit: Select and Find Previous

This command will find the previous occurrence of the selected text.

2.3.3.9 Edit: Find File

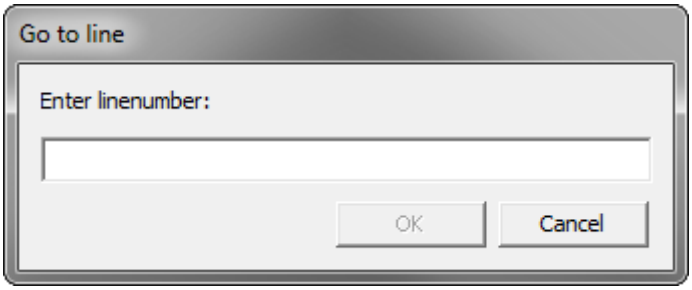


Using this command, you are able to search for source files in the project.
The list shows the files matching the filter entered in the top.
If no wildcard is used, all files starting with the filter are listed.

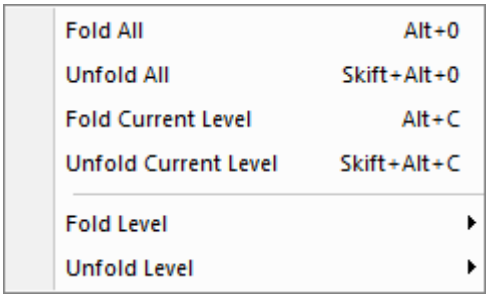
Pressing enter or clicking Open will open the selected file in the editor.

2.3.3.10 Edit: Go to line

By using the "Go to line" command, it is possible to jump to any line in the current file.



2.3.3.11 Edit: Folding



Fold all
This will fold every folding point in the file.

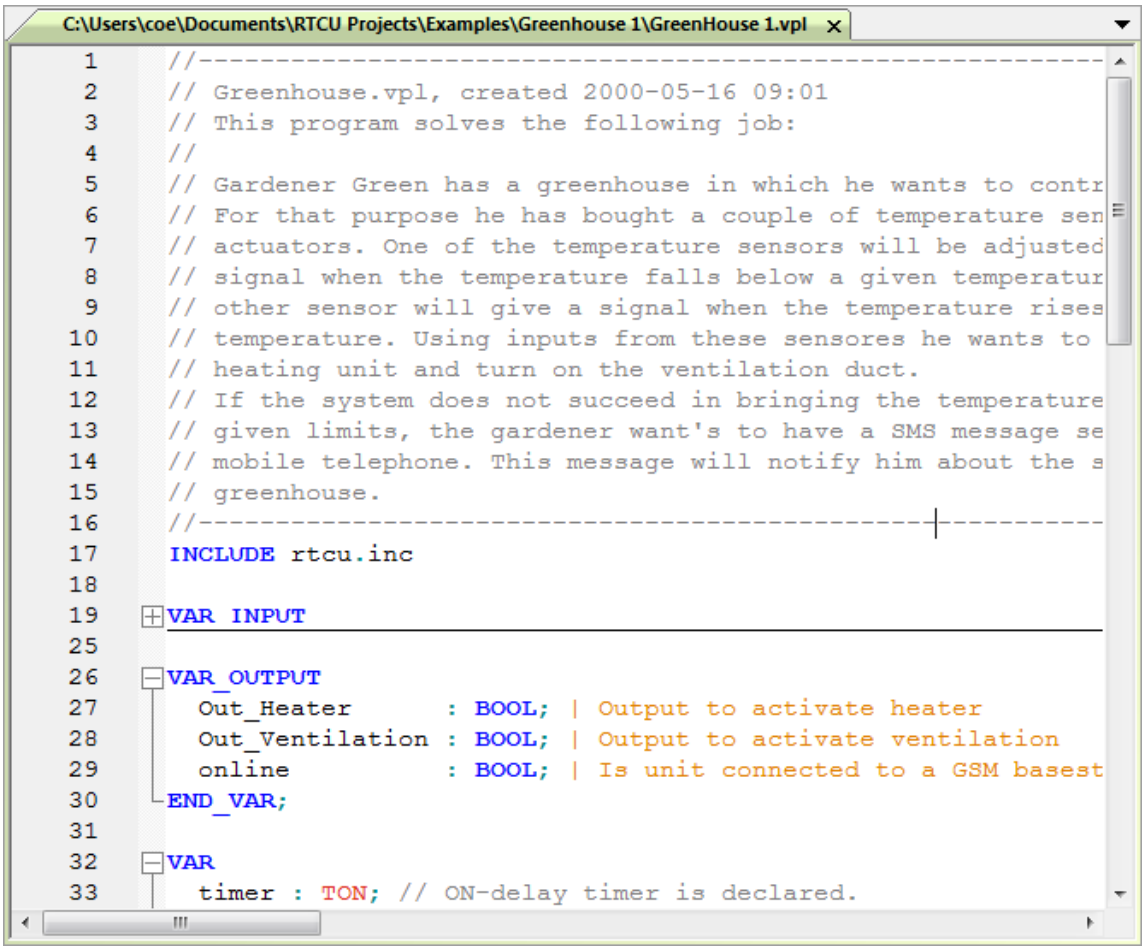
Fold the current level
This will fold the folding point of the code block that the current line is part of.

Unfold the current level
This will unfold the folding point of the code block that the current line is part of.

Fold the level
This will fold all folding points with the level that is selected from the sub menu.

Unfold the level
This will unfold all folding points with the level that is selected from the sub-menu.

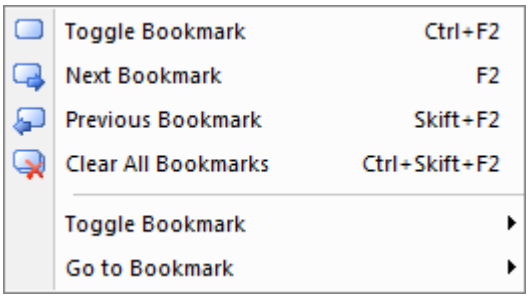
Unfold all
This will unfold every folding point in the file.



What is folding?
Folding is a feature that allows the user to selectively hide and display sections of the file.
This allows the user to manage large regions of potentially complicated text within one window, while still viewing only those subsections of the text that are specifically relevant during a particular editing session.
Folding points are the places that mark the beginning of a code block that can be folded.
There can be folding points inside the code block of other folding points. This is where the folding level comes in.
A folding level of one indicates that the folding point is not part of another code block, a folding level of two indicates that the folding point is part of only one code block, etc.

2.3.3.12 Edit: Bookmarks

2.3.3.12.1 Edit: Bookmarks



Bookmarks can be used to ease the navigation of large files.

The individual items:

- Toggle Bookmark
- Next Bookmark
- Previous Bookmark
- Clear All Bookmarks

- Toggle Bookmark
- Go to Bookmark

2.3.3.12.2 Edit: Toggle Bookmark

This command will place or remove a bookmark on the current line in the current file.

2.3.3.12.3 Edit: Next/Previous Bookmark

These commands will jump to the next or previous bookmark.

2.3.3.12.4 Edit: Clear all bookmarks

This will clear all the defined bookmarks in the current file.

2.3.3.12.5 Edit: Toggle Bookmark

Using the "Toggle Bookmark" sub menu, you can manipulate specific bookmarks in the file.

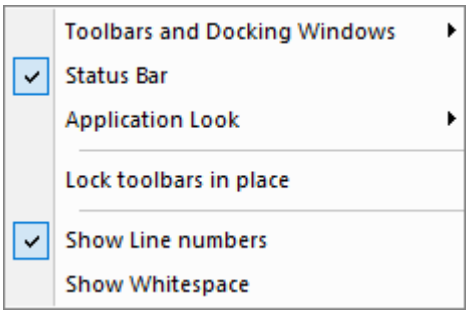
2.3.3.12.6 Edit: Go to Bookmark

Using the "Go to Bookmark" command, it is possible to jump to defined bookmarks (if any defined) in the current file.

2.3.4 Menu item: View

2.3.4.1 Menu item: View

The "View" menu is used for showing/hiding the various toolbars, project control and status bars of the RTCU IDE program as well as for adjusting what to show in the editor window.



The individual items:

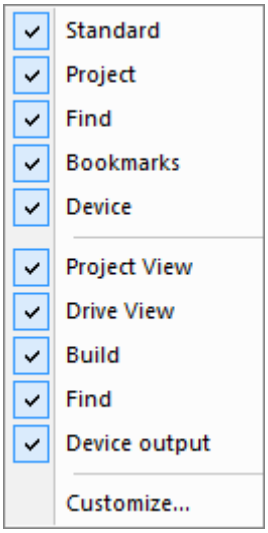
- Toolbars and Docking Windows
- Status Bar
- Application Look
- Lock toolbar in place

- Show Line numbers
- Show Whitespace

2.3.4.2 View: Toolbars and Docking Windows

2.3.4.2.1 View: Toolbars and Docking Windows

The "Toolbars and Docking Windows" menu is used to determine which toolbars and which windows to show.
Only the checked items are visible.



Toolbars:

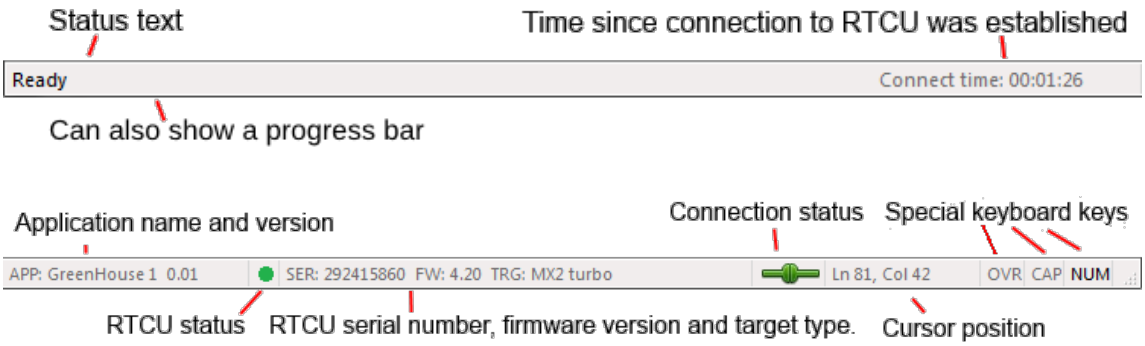
- Standard
- Project
- Find
- Bookmarks
- Device

Windows:
- Project View
- Drive View
- Build
- Find
- Device output

Customize:
Shows a dialog that makes it possible to customize the toolbars and menus.

2.3.4.3 View: Status Bar

The Status Bar is a small window at the bottom of the RTCU IDE program that shows information about a connected RTCU as well as information about the editor, including the status of the insert, caps-lock and num-lock keys.



The RTCU status uses the following icons to show the state of the connected RTCU:
- Running
- Halted
- Faulted
- Recovery mode

The connection status uses the following icons to show the status of the connection to the RTCU:
- Not connected
- Connected via cable
- High speed cable connection (NX32L devices only).
- Connecting to remote RTCU
- Connected via RTCU Communication Hub
- Connected via modem

2.3.4.4 View: Application Look

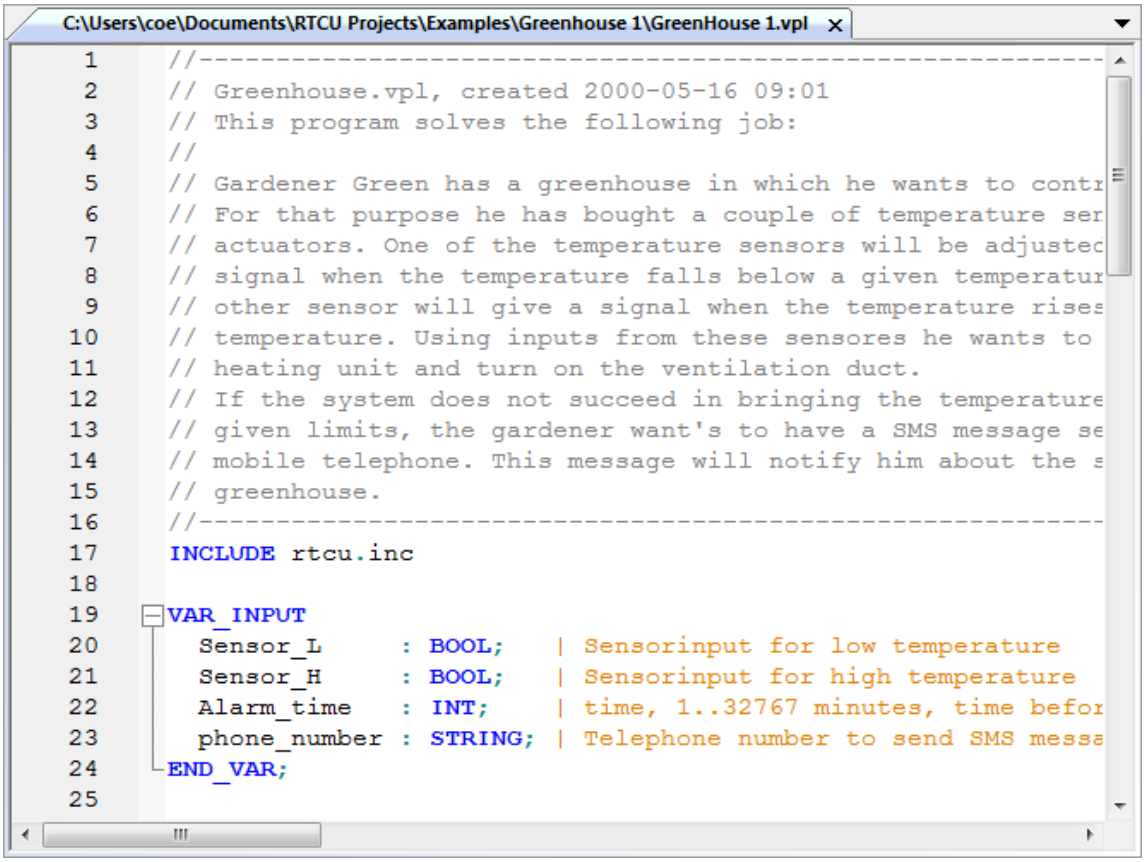
This menu determines the color scheme of the application.

2.3.4.5 View: Lock toolbars in place

Enabling the "Lock toolbars in place" will lock the toolbars at their current position and prevent them from being repositioned.

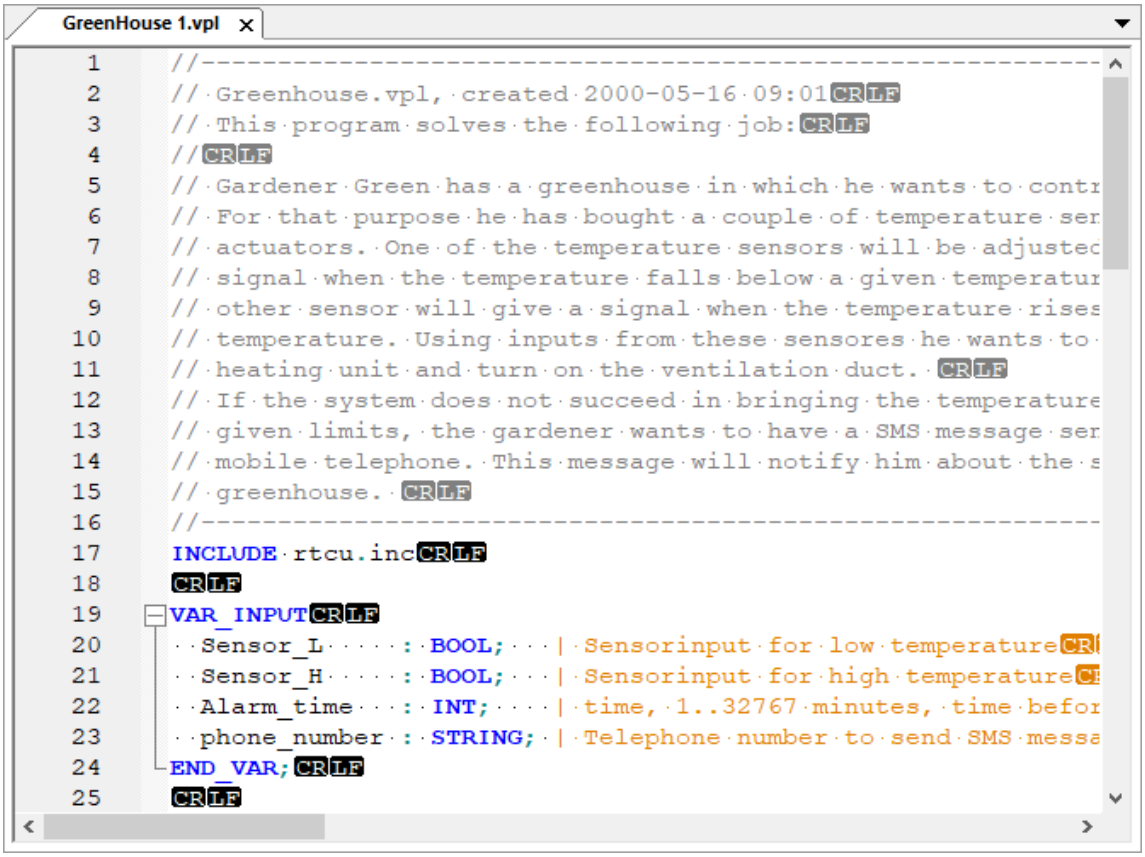
2.3.4.6 View: Show Line numbers

This command will show/hide line numbers in the margin of the VPL code-editor window.



2.3.4.7 View: Show Whitespace

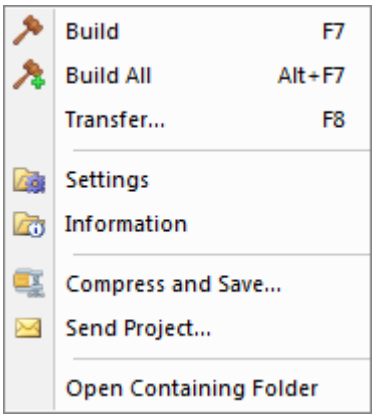
This command will show/hide whitespace in the VPL code-editor window.



2.3.5 Menu item: Project

2.3.5.1 Menu item: Project

The "Project" menu items are used to manipulate projects. From this menu, you are able to upload projects to an RTCU device, build, and send a complete project using email.



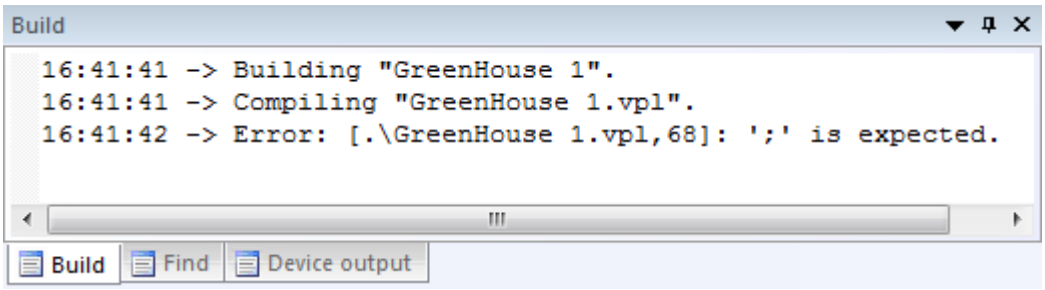
The individual items:

- Build
- Build All
- Transfer...
- Settings
- Information...
- Compress and save...
- Send project
- Open containing folder

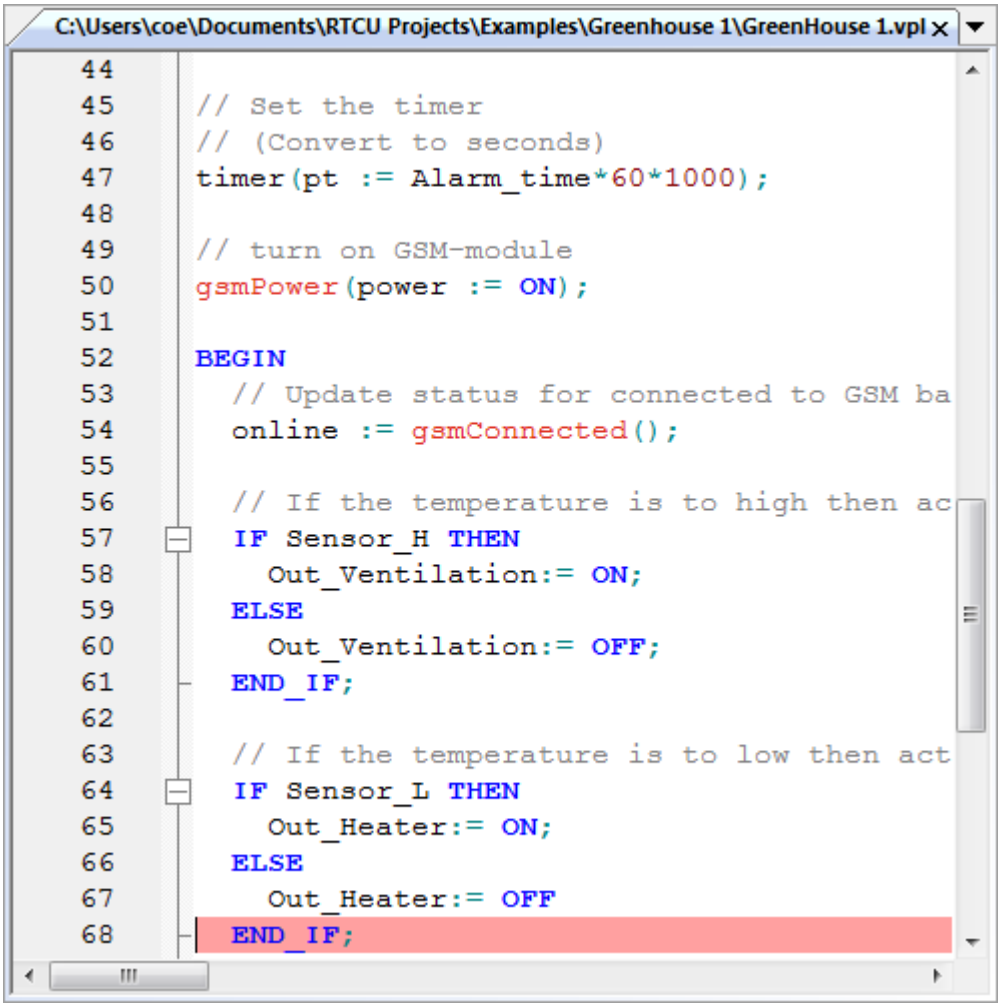
2.3.5.2 Project: Build

The "Project Build" command builds a project to make it ready for uploading to an RTCU device or executed in the RTCU Emulator. If any errors are encountered during the build process (mainly syntax errors in the various VPL files), an error message is shown in the build output window. The build process is then terminated and can be started again when the error has been fixed.

An example of an error could be:



And the current file will look like this :



It tells you that a ";" is missing in line 68, when actually it is the previous line that has not been terminated correctly with a ";".

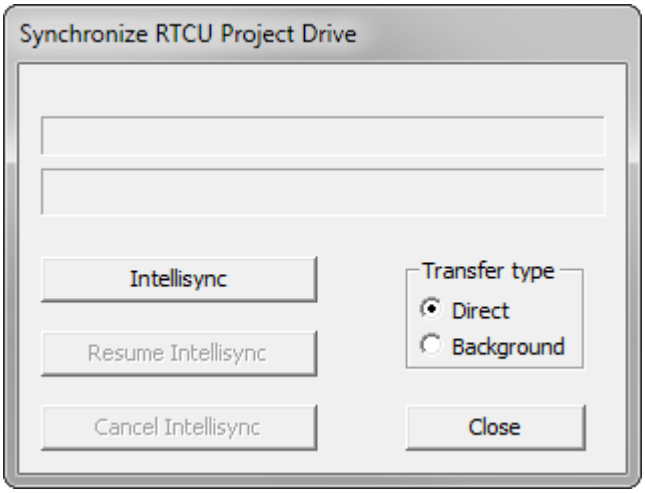
2.3.5.3 Project: Build All

The "Build All" command rebuilds the entire project.

2.3.5.4 Project: Transfer

This menu shows one of two different dialogs, depending on the type of project:

Syncronize RTCU Project Drive

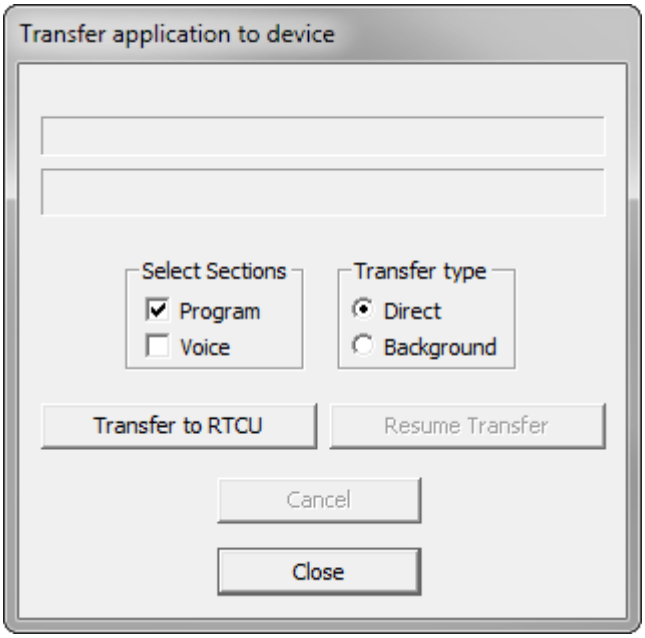


The "Syncronize RTCU Project Drive" dialog is used for synchronizing a project built for the NX32/NX32L execution architectures with the Intellisync Project Drive of a connected RTCU device.

The Intellisync only transfers the data that has changed compared to the image in the device, reducing the amount of data that is transferred and the amount of time it takes.
Depending on the RTCU type and/or connection type, you can choose to upload the project either as a "Direct" Intellisync or as a "Background" Intellisync. Using a Direct Intellisync will halt the execution of the running application and enter update mode. Using the Background Intellisync, the running application will continue to run while the data is transferred. When the transfer is finished, the switch to the new application will happen when the device resets next time. Another advantage of the Background Intellisync is that a failed/disconnected synchronization attempt can be resumed at a later time.

Before you use this feature with a programming cable for the first time, you must configure the RTCU IDE program with the correct port (COM or USB port) used. This is done in the Setup dialog.

Transfer application to RTCU device

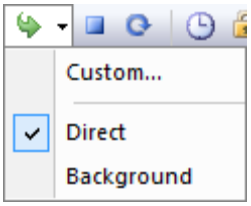


The "Transfer to RTCU" dialog is used for transferring a project build for the X32 execution architecture.

By clicking on the "Program" and "Voice" fields, you can enable transferring of either the program, voice messages or both.
Depending on the RTCU type and/or connection type, you can choose to upload the project either as a "Direct" upload or as a "Background" upload. Using a Direct upload will halt the execution of the running application and enter update mode. Using the Background upload, the running application will continue to run during the upload. When the upload is finished, the switch to the new application will happen when the device resets next time. Another advantage of the Background upload is that a failed/disconnected upload attempt can be resumed at a later time. When using the Background upload, the "Voice" section cannot be transferred, and the current Voice data in the device may be overwritten as the device uses the Voice flash memory for its operation. This means that if the application contains Voice data, and the background upload is used, the Voice data will have to be transferred separately after the background upload has finished. Also see the verCheckUpgrade() function-block.

Before you use this feature with a programming cable for the first time, you must configure the RTCU IDE program with the correct port (COM or USB port) used. This is done in the Setup dialog.

Synchronization button



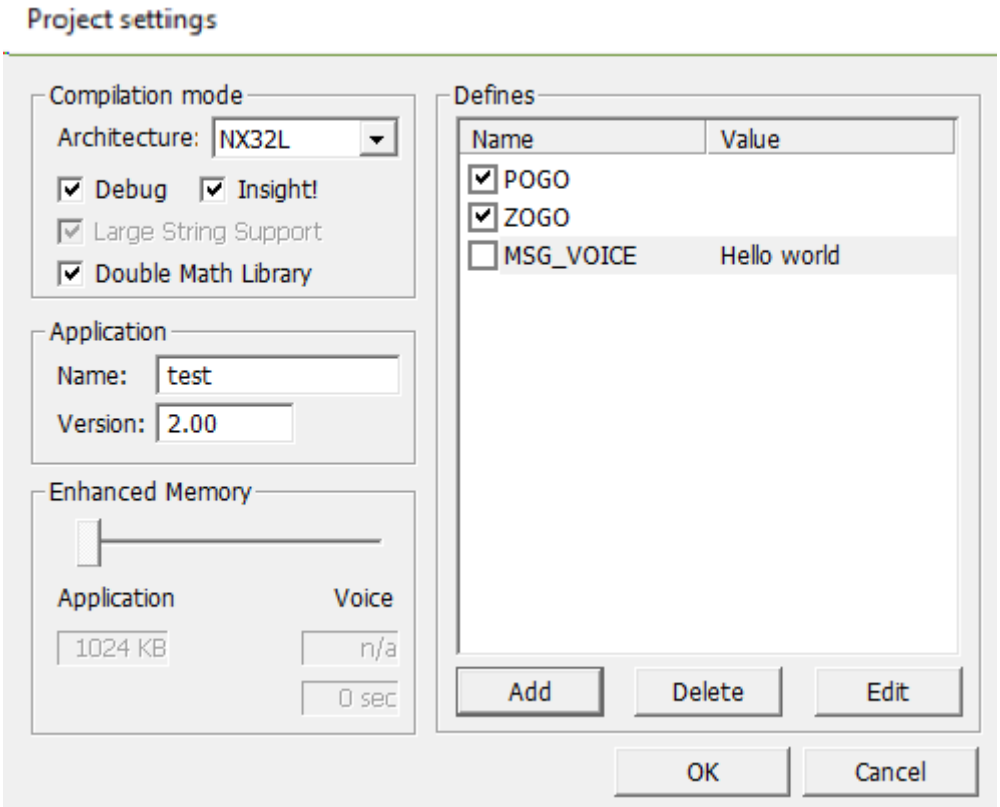
The Synchronize button on the device toolbar makes it possible to synchronize a connected RTCU device with a single click.

Clicking the down arrow will allow selection of the type of synchronization to perform when the button is clicked. If "Custom..." is selected, the Project: Transfer dialog will be shown when the button is clicked.

2.3.5.5 Project: Settings

Change settings for the active project.

When the "Project Settings" command is invoked, a dialog will be shown:



This dialog is where the settings of the project are changed.

Compilation mode
This section contains settings specific to the RTCU device.

Architecture
The execution architecture for the project.
'Large' refers to the first generation of RTCU devices: RTCU M10/M11 and RTCU A9.
'X32' refers to all second generation of RTCU devices, such as the RTCU MX2, RTCU CX1, RTCU DX4 and RTCU SX1.
'NX32' refers to the NX32 generation of RTCU devices, such as the RTCU MX2 turbo/encore/warp.
'NX32L' refers to the NX32L generation of RTCU devices, such as the RTCU NX-900, RTCU NX-400, RTCU NX-200, RTCU LX2, RTCU LX4 and RTCU LX5.

Debug
The debug feature will add additional information to the compilation and the final execution image allowing the run-time to determine in which file and at what source code line the fault occurred. Please refer to the Fault log on how this information is presented.
For certain faults, such as index error and divide-by-zero the line retrieved will be exact, and for other it will be an approximate source code line.
Besides a slightly larger execution image (approx 5%) there will be a small execution overhead that in most cases are negligible.

Insight!
Includes information to the compilation to enable *Insight*! in the RTCU IEX.

Large String Support
The LSS option extends the support for strings as follows:

- The maximum length of a STRING is increased from 254 to 16384 characters.
- The maximum length of a static string is increased from 256 to 1024 characters.
- The project can contain more than 64 Kbytes of static strings.

Double Math Library
For backward compatibility the standard setting for floating-point math functions and operations is FLOAT, but can here be changed to DOUBLE.
When this option is checked, DOUBLE will be used in math calculations and return types.

Application profile
This section contains setting specific to the application profile of the project.
The application profile can only be edited when the check mark is selected and the X32 or NX32 option above is selected.

Name
The name of the application. Maximum size is 15 characters.

Version
The version of the application.
The format of the version is "major.minor", where major can be a number between 0 (zero) and 299, and minor can be a number between 0 (zero) and 99.

Memory configuration
This section is used to define the available memory for the application and voice messages.
When moving the slider to the right, more memory is made available to the application by decreasing the memory available to voice messages.
This allows for larger applications at the expense of fewer voice messages.
The available memory for application and voice messages is displayed below the slider in Kbytes, as well as the number of seconds available for voice messages.
Note: This section is only enabled when the X32 option above is selected and the device in use supports X32 enhanced memory.

Defines
This section contains definitions of conditional compilation symbols and Macro constants.
A symbol is defined when there is a check mark in the check box. It will be undefined otherwise.
Defining a symbol here has the same effect as using #DEFINE at the top of the source code.

When typing in the value of a macro constant, it will be considered a number if only digits is used (including decimal point). If any other character is used it will be considered a string.
In case the define must be a string holding a number, add " around the number. For example: "100".
Note: a string cannot contain " as part of the text.

Create
To create a new define, press the 'Add' button, and a dialog will open where the name of the define and an optional constant value can be typed in. Alternately it is possible to create a new entry by pressing the INSERT key

Remove
Select the define and press the 'Delete' button.
Alternatively it is possible to delete an entry by pressing the DELETE key.

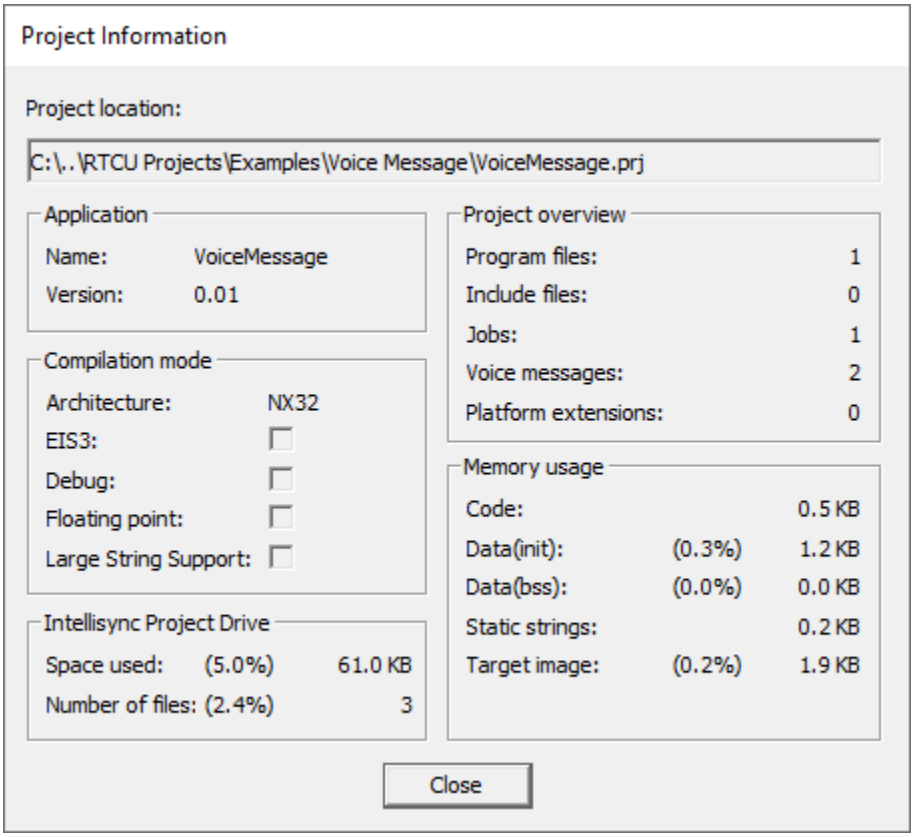
Change constant
Select the define and press the 'Edit' button and a dialog will open where the constant value can be changed.

Define / Undefined
Click on the check box to change between Defined / Undefined.

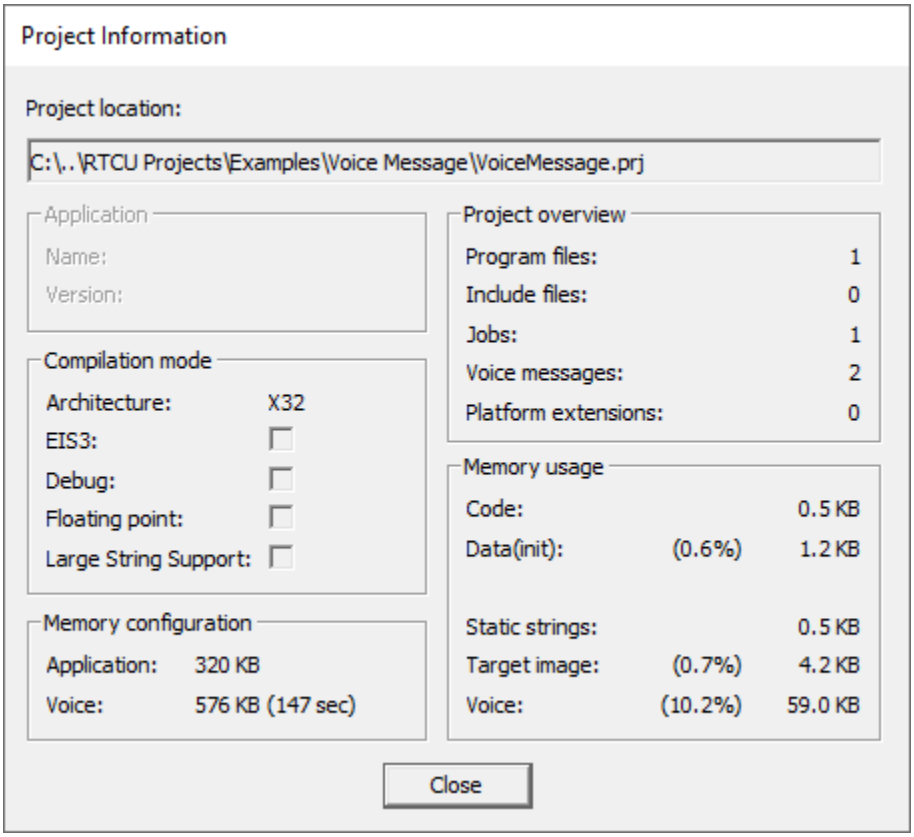
2.3.5.6 Project: Information

This command shows information for the active project.

When the "Project Information" command is invoked, the following dialog will be shown for projects built for the NX32 and NX32L compilation modes:



When a project built for the X32 compilation mode is used, the following dialog is shown:



Project location
The absolute path to the project file.

Application
This section contains the application profile of the compiled application. If no application profile is available in the compiled application the section will be disabled.

Name
The name of the application.

Version
The version of the application.

Compilation mode
This section contains information about the compilation mode used for the project.

Architecture
The execution architecture that the project is built for.
X32, NX32 and NX32L are available. Large is for legacy/backward compatibility.

EIS3
The project uses features from EIS3.
Only firmware V5.00 and V1.30 supports EIS3.

Debug
The debug feature is enabled in the project.

Floating point
The project uses floating point instructions.

Large String Support
The Large String Support feature is enabled in the project.

Intellisync Project Drive
This section contains information about the usage of the Intellisync Project Drive.
This section is only available for the NX32 and NX32L execution architectures.

Space used
The space used for application, voice messages and user files.
Also expressed in percentage of the total capacity.

Number of files
The number of files on the project drive.
Also expressed in percentage of the maximum allowed number.

Memory configuration
This section contains the available memory for the application and voice messages.
This section is not available for the NX32 and NX32L execution architectures.

Application
The available memory for the target image.

Voice
The available memory for voice messages.
Also expressed in number of seconds.

Project overview
This section contains an overview of the files in the project.

Program files
The number of "Program" files (VPL) in the project.

Include files
The number of "Include" files (INC) in the project.

Jobs
The number of jobs in the project.

Voice messages
The number of voice files in the project.

Platform extensions
The number of platform extensions in the project.
This information is only available for the NX32L compilation mode.

Memory usage
This section contains information about the amount of memory used by the compiled application.

Code
The size of the compiled application.

Data(init)
The size of the data in the compiled application.
Also expressed in percentage of the total capacity.

Data(bss)
The size of the uninitialized data section in the compiled application. Uninitialized data is variables that are not explicitly set to a value by the application.
Also expressed in percentage of the total capacity.
The data in this section does not increase the size of the target image.
This information is only available for the NX32L compilation mode.

Static strings
The size of the static strings in the compiled application.

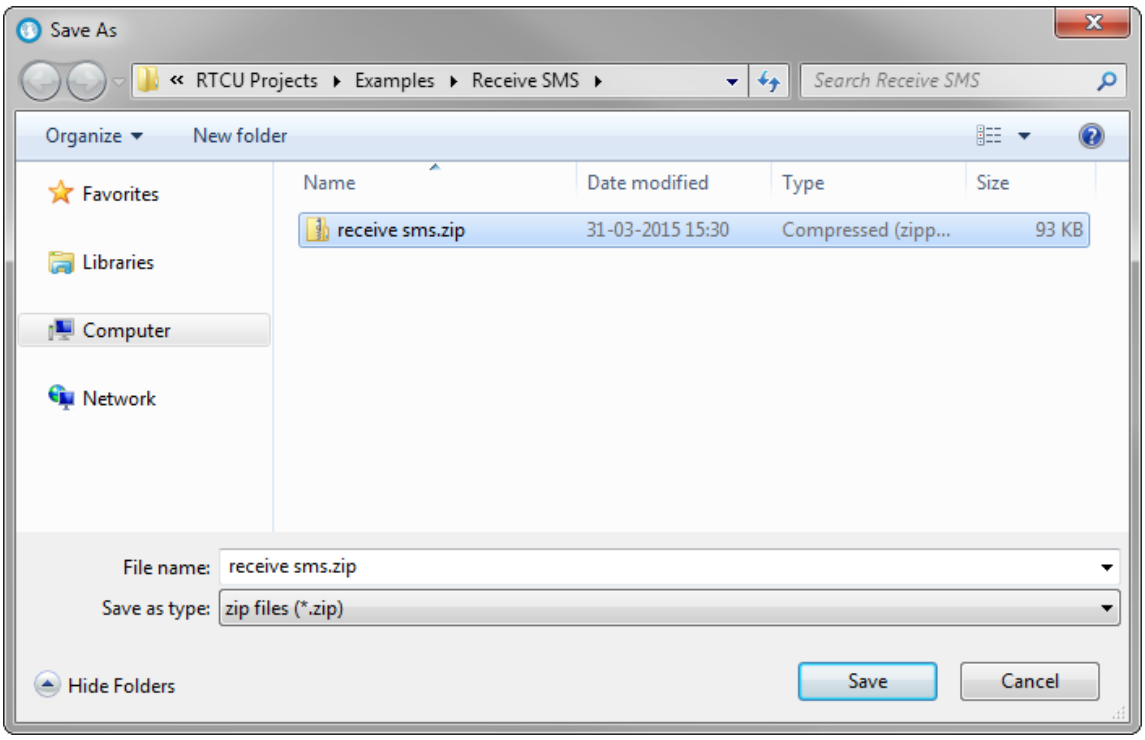
Target image
The total size of the compiled application.
Also expressed in percentage of the total capacity.

Voice
The total size of the included voice messages.
Also expressed in percentage of the total capacity.
This information is not available for the NX32 and NX32L compilation modes.

2.3.5.7 Project: Compress and save

This command saves the active project as a compressed (ZIP) file at a specified location.

When the "Compress and save" command is invoked, a dialog will be shown:

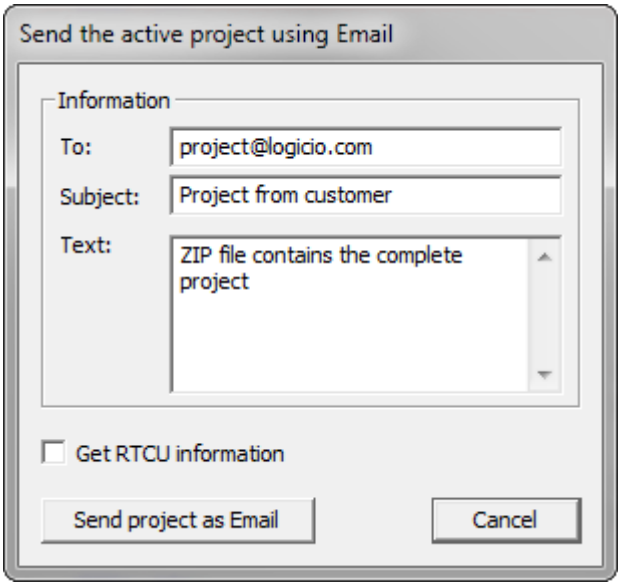


Type or select the file name the project should be compressed and saved to, then press "Save".

2.3.5.8 Project: Send project

Sends the active project as a compressed (ZIP) file via email.

When the "Project Send" command is invoked, a dialog will be shown:



In this dialog, a number of entry fields have already been filled out. You can make changes to the individual fields. This is useful for sending a project to customer support, to a colleague, or to transfer a modified project to your customer.

Depending on the type of email program that you use, it might be necessary for you to do a "Send/Receive" in your email program in order to actually make the email program send the email.

2.3.5.9 Project: Open containing folder

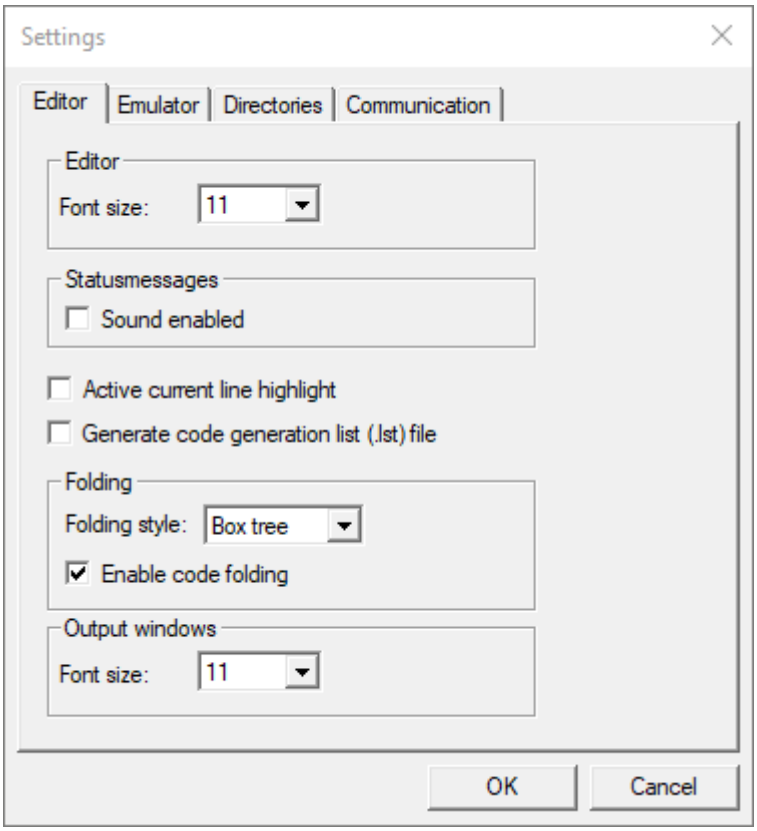
Opens the folder containing the project in Windows Explorer.

2.3.6 Menu item: Settings

2.3.6.1 Menu item: Settings

This dialog is where the settings for the RTCU IDE is configured.

Editor



The "Editor" page shows the configuration options for the editor.

Font size
Sets the font size of text in the editor window.

Sound enabled
Enable/disable sound when various actions are completed (such as uploading, etc).

Active current line highlight
Enable/disable highlight of the current line in the editor window.

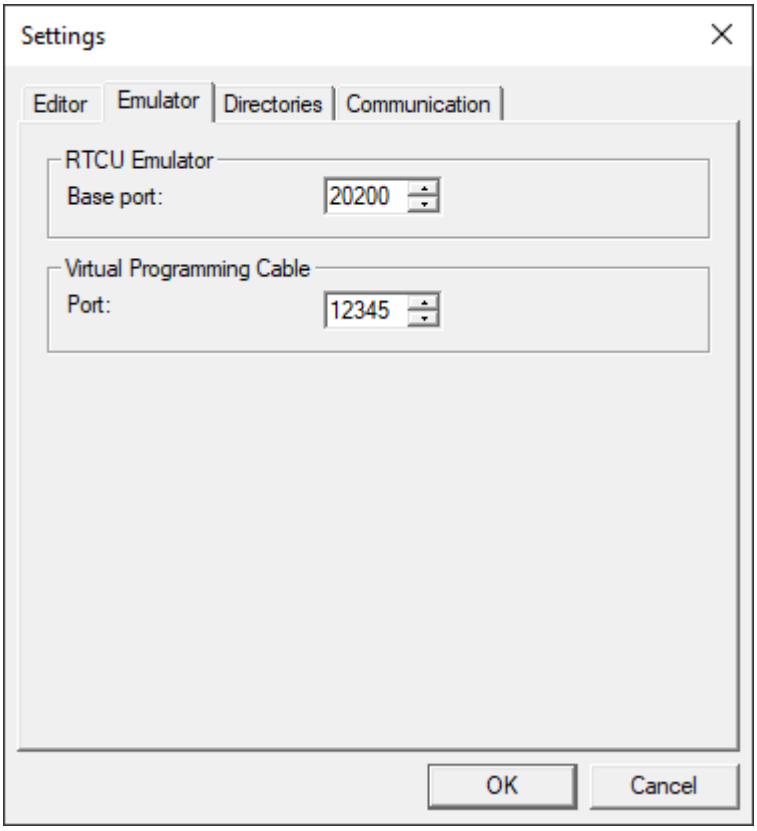
Folding style
The style used in the margin to show folded and not folded code.

Enable code folding
Enable/disable the option of folding the code in the editor window.

Generate code generation list (.lst) file
When this option is selected the VPL compiler will output the virtual machine assembly code to a file with the same name as the VPL file, but with an '.lst' extension.
The list file is useful to determine the stack-layout when using the RTCU M2M Platform.

Output windows Font size
Sets the font size of text in the Device output, Build and Find windows.

Emulator



The "Emulator" page shows the configuration options for the RTCU Emulator.

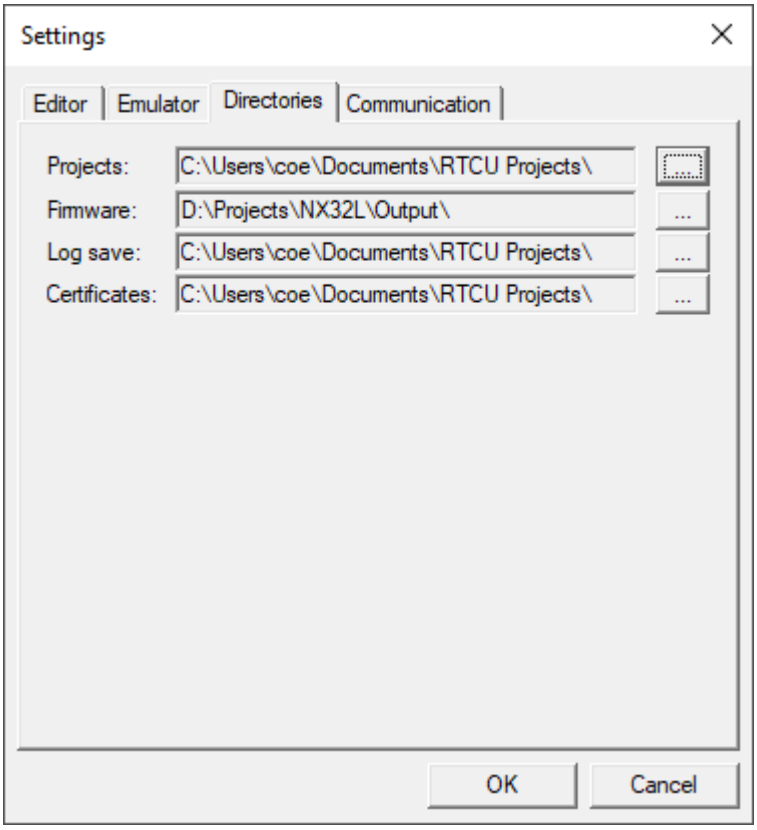
RTCU Emulator

Base port
Set what port range to use when communicating with the RTCU Emulator when started from the Emulator menu. The range consists of 16 ports, starting at the specified number. All 16 ports are not used at the same time, but must still be available and not be used by other programs.

Virtual Programming Cable

Port
Set the IP port of the socket used by the Virtual Programming Cable to connect the RTCU Emulator to the RTCU IDE.

Directories



The "Directories" page shows the configuration options for the directories used.

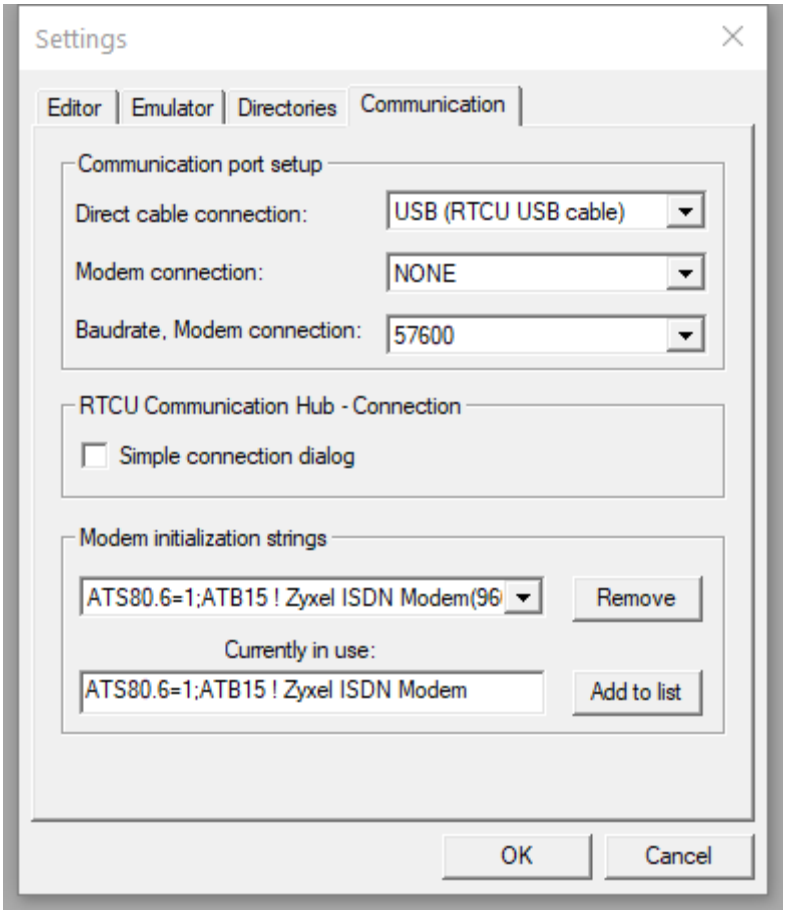
Projects
This is the directory where new projects are created.

Firmware
This is the directory where downloaded firmware is stored.

Log save
This is the directory where log files are saved.

Certificates
This is the directory where the RTCU IDE expects to find certificates.

Communication



The "Communication" page shows the configuration options for communication with devices.

Direct cable connection
Sets the serial port used to communicate with a connected RTCU device.

Modem connection
Sets the serial port used for a modem connected to a remote RTCU device.

Baudrate. Modem connection
Sets the baud rate used to communicate with the modem.

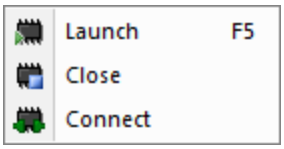
Simple connection dialog
This option selects the simplified RTCU Communication Hub Connection Dialog.

Modem initialization strings
Sets the initialization string send to the modem when starting a remote connection.
Using the drop-down list, it is possible to select different modem initialization strings, depending on the type of modem you are using to connect to a remote modem.
If the initialization string your particular modem needs is not listed, you can type the string in the entry field just below the drop-down list and then press the "Add to list" button to insert that in the list for future reference. It is possible to add a descriptive text after the initialization string by adding an "!" (exclamation mark) just before the comment. If more AT commands are needed for the proper initialization, commands can be separated with the ";" (semicolon) between commands.

2.3.7 Menu item: Emulator

2.3.7.1 Menu item: Emulator

The "Emulator" menu is used for interact with the RTCU Emulator.
If the RTCU Emulator is not installed, the content of this menu is not available.



The individual items:

- Launch
- Close
- Connect

2.3.7.2 Emulator - Launch

Launch an instance of the RTCU Emulator.

Please refer to the RTCU Emulator section of the manual for an explanation of the RTCU Emulator.

2.3.7.3 Emulator - Close

Close an RTCU Emulator instance previously launched by the RTCU IDE.

Please refer to the RTCU Emulator section of the manual for an explanation of the RTCU Emulator.

2.3.7.4 Emulator - Connect

When connect is selected, the RTCU IDE will emulate a programming cable which the RTCU Emulator can connect to.
Allowing the RTCU IDE to communicate with the emulated device as if it was a physical device.

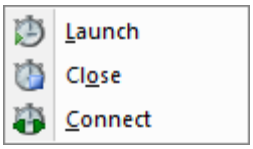
It is not possible to make a cable connection to a physical device, when the emulated programming cable is enabled.

Please refer to the RTCU Emulator section of the manual for an explanation of the RTCU Emulator.

2.3.8 Menu item: IEX

2.3.8.1 Menu item: IEX

The "IEX" menu is used to interact with RTCU Instrumented Execution.
If RTCU Instrumented Execution is not installed, the content of this menu is not available.



The individual items:

- Launch
- Close
- Connect

2.3.8.2 IEX - Launch

Launch the RTCU Instrumented Execution.

Please refer to the RTCU Instrumented Execution section of the manual for an explanation of RTCU IEX.

2.3.8.3 IEX - Close

Closes the RTCU IEX instance previously launched by the RTCU IDE.

Please refer to the RTCU Instrumented Execution section of the manual for an explanation of RTCU IEX.

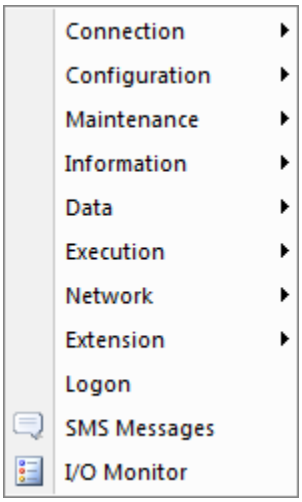
2.3.8.4 IEX - Connect

When connect is selected, the RTCU IDE will try to make any connected device connect to the RTCU IEX.
The connection will only be established if there is a high speed cable connection to the device and the device has support for IEX.

Please refer to the RTCU Instrumented Execution section of the manual for an explanation of RTCU IEX..

2.3.9 Menu item: Device

2.3.9.1 Menu item: Device



- Connection
- Configuration
- Maintenance
- Information
- Data
- Execution
- Network
- Extension
- Logon
- SMS Messages
- I/O Monitor

2.3.9.2 Device: Connection

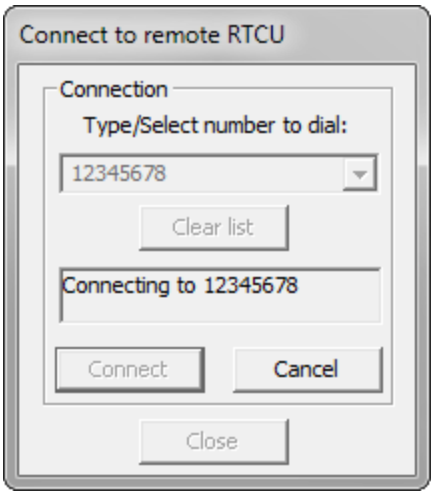
2.3.9.2.1 Device: Connection

By using the "Device: Connection" menu, it is possible to handle all aspects of connecting and managing a remote RTCU device by using a dial-up connection or the RTCU Communication Hub.

The individual items:

- Modem
- RTCU Communication Hub

2.3.9.2.2 Connection: Modem



By using this dialog, it is possible to connect via a standard Hayes-compatible modem to an RTCU device. The connection will be made through the GSM module of the RTCU device. The dialog will stay on top of the screen as long as there is an active connection.

For establishing a modem connection, there are two distinct possibilities. These are as follows:

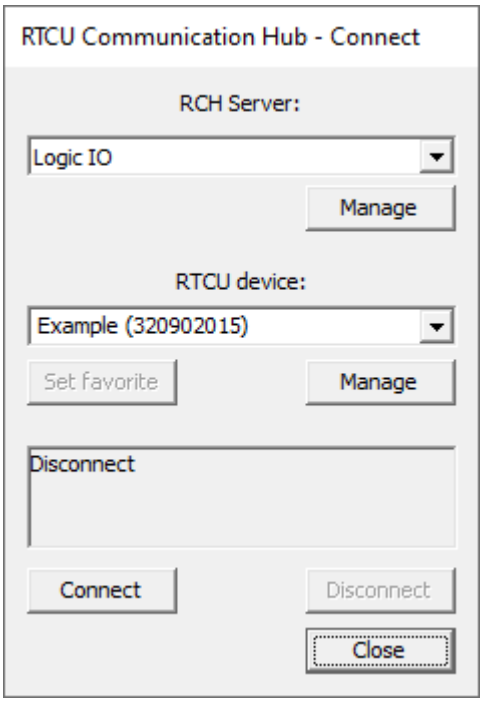
- RTCU device using a SIM-card with a dedicated data number (usually a paid option). In this case, the data connection can be established using a standard analogue modem. Calling the standard number will simply result in the call terminating as a voice.
- RTCU device using a standard SIM-card with only one call number shared for data and voice calls. To ensure that the call bearer information is present, a V110 ISDN or GSM CSD connection must be used to establish the connection.

Please note that country code typically is prefixed with 00 (e.g. 0045 for Denmark).

Note: Modem/CSD connections are considered obsolete and is not supported by most Mobile networks. The latest NX32L generation of devices also do not support Modem/CSD connections.

2.3.9.2.3 Connection: RTCU Communication Hub

2.3.9.2.3.1 Connection: RTCU Communication Hub



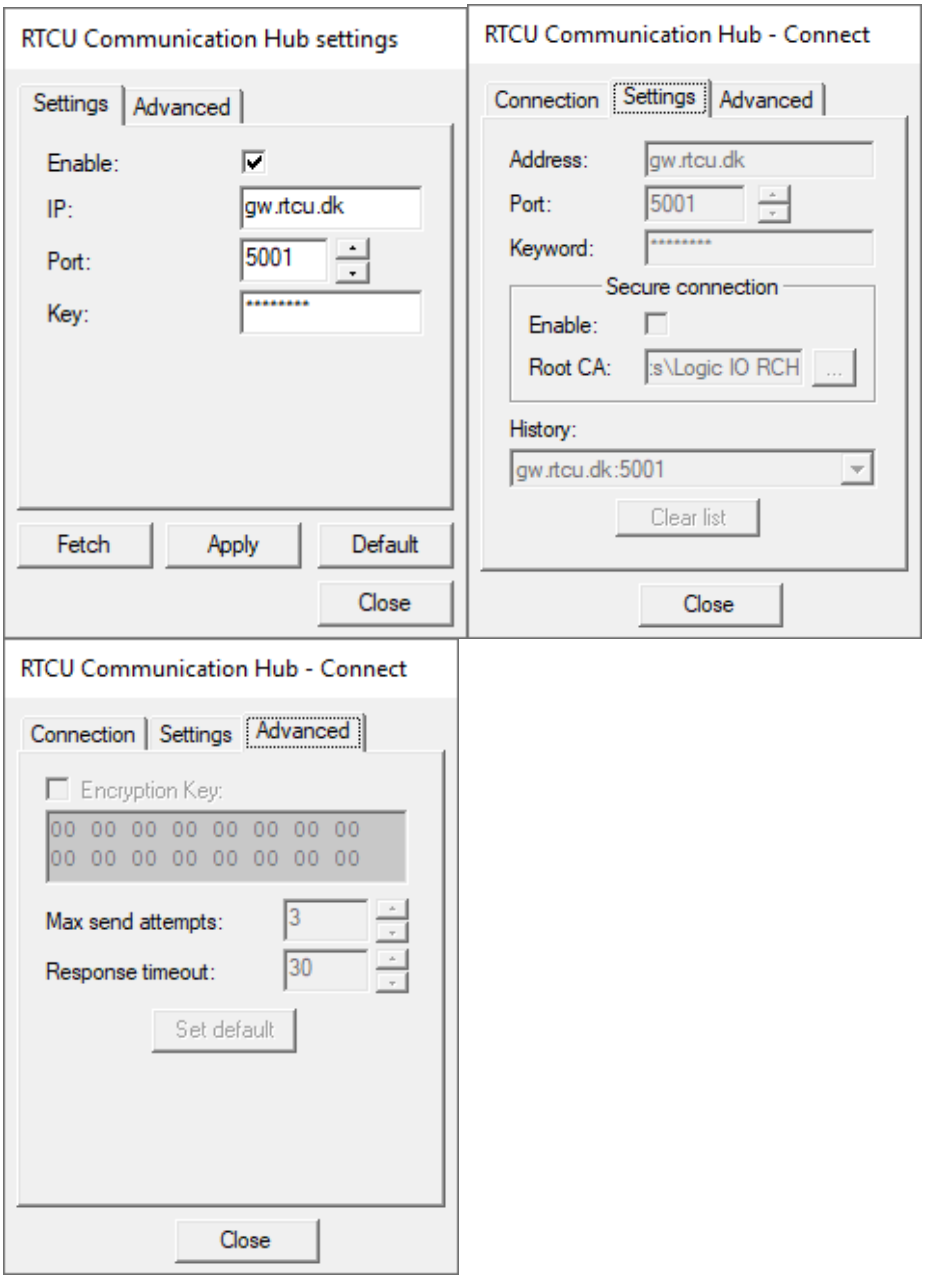
By using this dialog, it is possible to connect through the Logic IO RTCU Communication Hub to a remote RTCU device - provided that the RTCU device is connected to a suitable IP network.

RCH server:
Select the RCH server used to connect to the RTCU device.
The 'Manage' button opens the server management dialog, which allows to creating new servers, edit servers, etc.

RTCU devices:
Select the RTCU device to connect to.
It is possible to save your favorite RTCU devices.
These favorites can later be selected from the drop-down.
The 'Set favorite' button creates a new favorite client, using the Node ID and RCH server selected.
The 'Manage' button opens the client management dialog, which allows to create new favorites, edit favorites, etc.

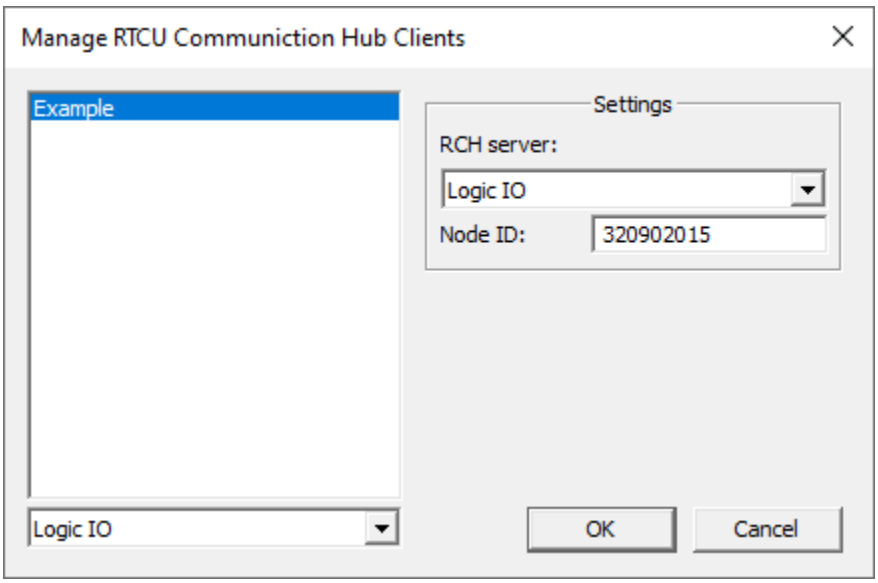
The 'Connect' button opens a connection to the RTCU device using the selected RCH server.
The 'Disconnect' button closes a connection.

As an alternative to the standard connection dialog, it is also possible to select a simpler dialog that focuses on a single RCH Server with less frequent access to many device:



The selection of connection dialog is set in the menu Settings.

2.3.9.2.3.2 Client manager



This dialog is used to manage the favorite devices.
Any changes made will only take effect when the OK button is pressed.

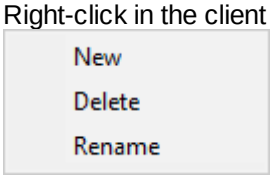
Only the clients for the selected server are listed.
The server can be selected below the list.

The settings group shows the parameters of the client:

Node ID
This is the node ID of the RTCU device, it corresponds with the serial number.

RCH Server
This is the RCH server to which the RTCU device is connected.
The drop-down list all available RCH servers,

Right-click in the client list for the menu:

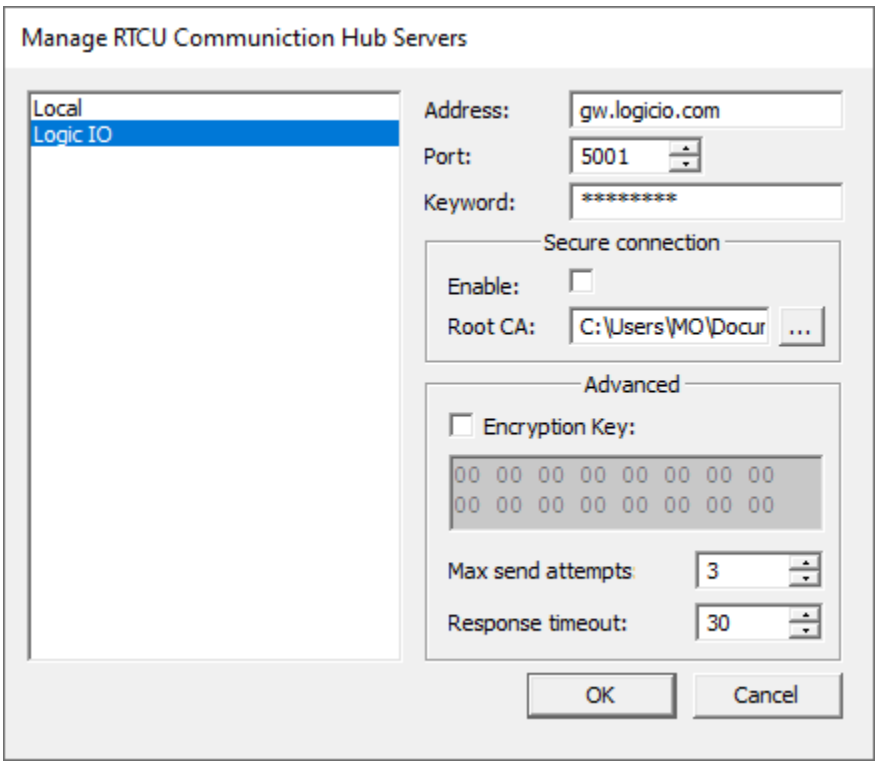


New
This creates a new favorite client.
The new client will use the parameters from the settings group.

Delete
This deletes the selected client.

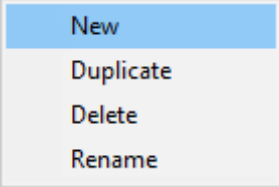
Rename
This changes the name of the selected client.

2.3.9.2.3.3 Server manager



This dialog is used to manage the RCH servers.
Any changes made will only take effect when the OK button is pressed.

Right-click in the server list for the menu:



New
This creates a new server.
The new server will use the default settings.

Duplicate
This creates a new server.
The new server will use the settings from the selected server.

Delete
This deletes the selected server.

Rename
This changes the name of the selected server.

The settings group shows the parameters of the server:

Address
The IP address (or symbolic name) of the server.

Port
The port number to be used for communicating with the server.

Keyword
The key value to be used when communicating with the server.

Secure connection
Control whether a Standard connection or Secure connection is used to connect to the server.

Root CA
This is the path to the Root CA certificate necessary to establish a secure connection to the server.

Encryption key
The 128-bit long key is used for the encryption of data sent to and received from the server.
The encryption in the device can be enabled/disabled using the Encryption Key check box.
It is not possible to use the encryption option with secure connections.

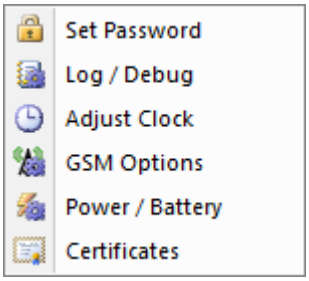
Maximum send attempts
This is the maximum number of send request attempts before the send fails.

Response timeout
This is the time spent waiting for a response, and it is specified in seconds.

2.3.9.3 Device: Configuration

2.3.9.3.1 Device: Configuration

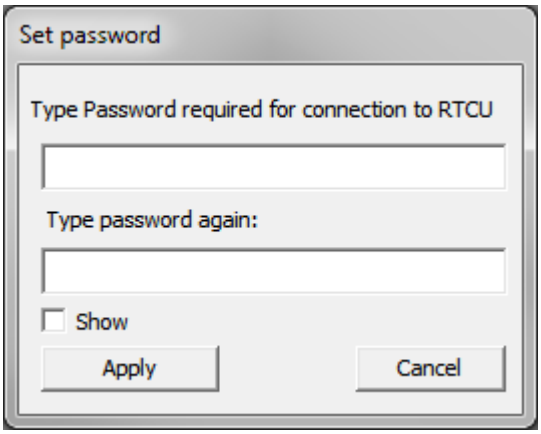
By using the "Configuration" menu, it is possible to query and set the parameters of a connected RTCU.



The individual items:

- Set password
- Log / Debug
- Adjust Clock
- GSM Options
- Power / Battery
- Certificates

2.3.9.3.2 Configuration: Set password

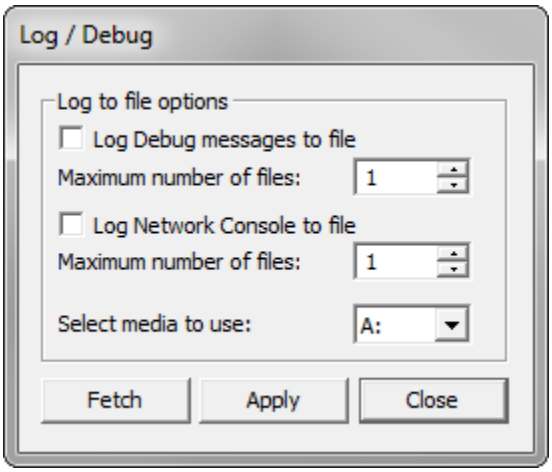


By using this dialog, it is possible to change the password for a connected RTCU device. It is a requirement that the password is entered every time communication is initiated with an RTCU device. The "Logon dialog" is then displayed. The password can be empty. If that is the case, the "Logon dialog" will not be shown, and the RTCU IDE is immediately logged on to the RTCU device.
The 'show' check-box will show the password and disable the second entry field.

When settings the password the following rules apply:

- All passwords must be at least 8 characters. Only ASCII characters can be used.
- The password must comply with a minimum of three of the following:
  - A minimum of 1 upper-case character.
  - A minimum of 1 lower-case character.
  - A minimum of 1 digit.
  - A minimum of 1 special character.

2.3.9.3.3 Configuration: Log / Debug



This dialog is used to configure various device parameters.

Log to file options
This option enables the debug messages and the network console messages to be logged in files on the selected media. These messages are the same as shown in the debug dialog and the Network Console.

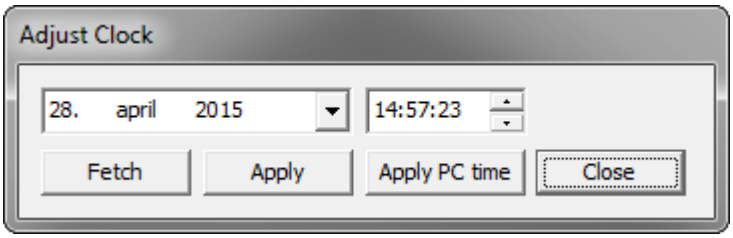
The files have the following format: YMMDD###.LOG where Y is the last digit of the year (2022 = 2), MM is the month, DD is the day, and ### is a number between 0 and 999.
To prevent log files from filling up the media, the files are limited in size - and there is a set number of files. If there are too many files, the oldest files will be deleted.
When the file with the number 999 reaches it maximum size. no more messages will be logged until midnight, at which point the count is reset and a new file is created.

The maximum number of files parameter determines how many files are saved in the log. Each file has an approximate size of 512 KByte.

The select media to use parameter determines which media the log files are stored on.
Please note that this option is not available on all targets.

This is the same as the VPL function DebugSetLogParam.

2.3.9.3.4 Configuration: Adjust Clock

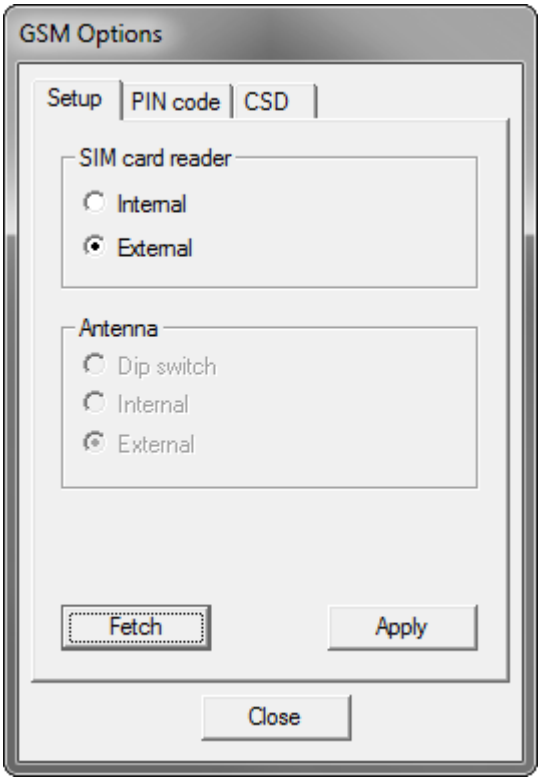


Using this command, you can query the Real-Time Clock on a connected RTCU device, and you can adjust the clock of the device.
Using the "Apply PC time" button, it is possible to set the clock of the RTCU device to the current time of your PC.

2.3.9.3.5 Configuration: GSM Options

This dialog is used to configure the GSM parameters of the connected RTCU device.
Each page handles a different part of the configuration.

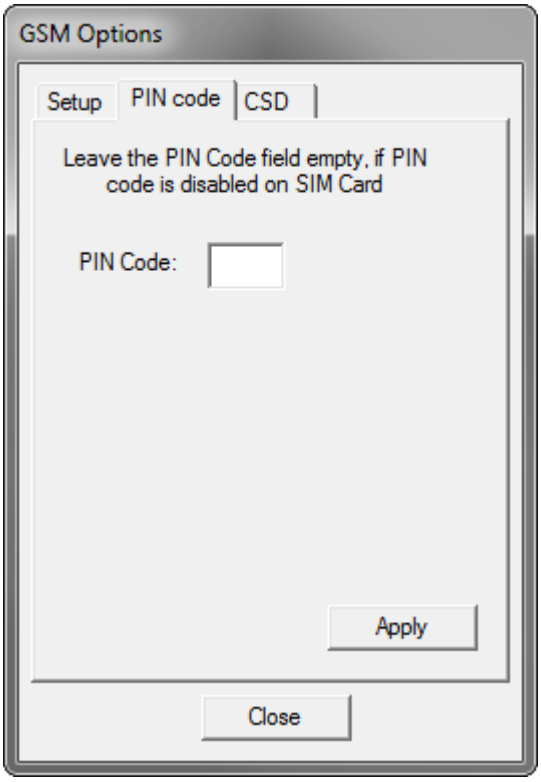
Setup page



SIM card reader
This option sets the selected SIM card reader on a device with multiple SIM card readers. This is the same as the VPL function gsmSelectSIM.

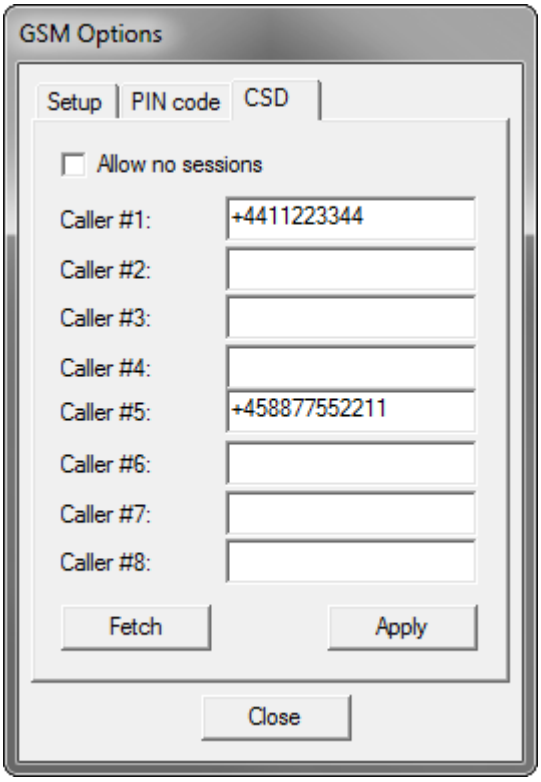
Antenna
This option sets the selected GSM antenna to use on a device with switchable antennas. This is the same as the VPL function gsmSetAntennaMode.

PIN code page



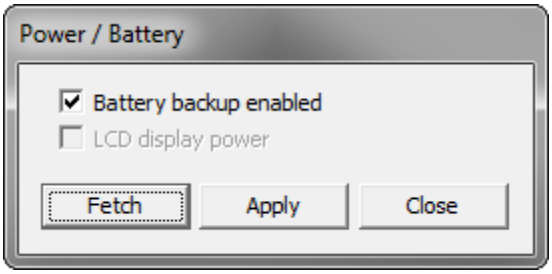
By using this page, it is possible to enter the PIN code that the RTCU device must use in order to activate the GSM module. If the PIN code is disabled on the SIM Card, the PIN code field must be blank. Please note that this dialog is just for entering the PIN code that has been assigned to the SIM Card. It is under no circumstances used for changing this code. This would have to be done from a "normal" mobile phone or with the gsmSimSetPIN function. If the correct PIN code is not entered here, the RTCU device be unable to connect to the GSM network.

CSD Page



Using this page, it is possible to view and modify the list of callers that are allowed to create a CSD session with the connected RTCU device. Please note that this list is ONLY used when a data call is issued to an RTCU and NOT when a voice call or an SMS message is initiated to a device. If the "Allow no sessions" is enabled, no callers are allowed at all. The contents of all the eight number fields will, however, be preserved.

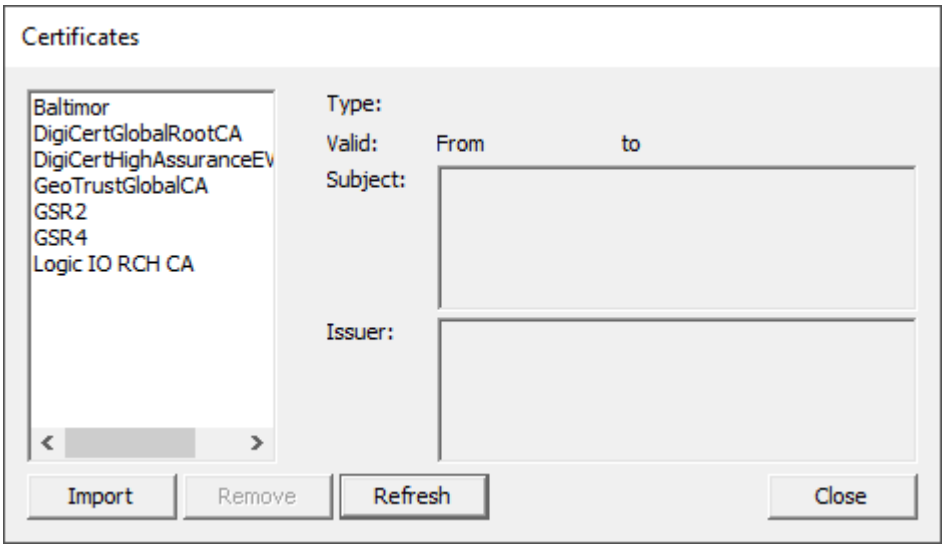
2.3.9.3.6 Configuration: Power / Battery



Battery backup enabled
This option sets the behaviour of the device when external power fails. This is the same as the VPL function pmPowerFail.

LCD display power
This option sets the behaviour of the LCD displays when the device starts (either by reset or if external power is applied).
This is the same as the VPL function displayPower.
This option is disabled if the connected device does not have the LCD display option.

2.3.9.3.7 Configuration: Certificates



This dialog is used to manage the certificates in the RTCU device.
Certificates are used for establishing secure network connections. (See Security features for more information)

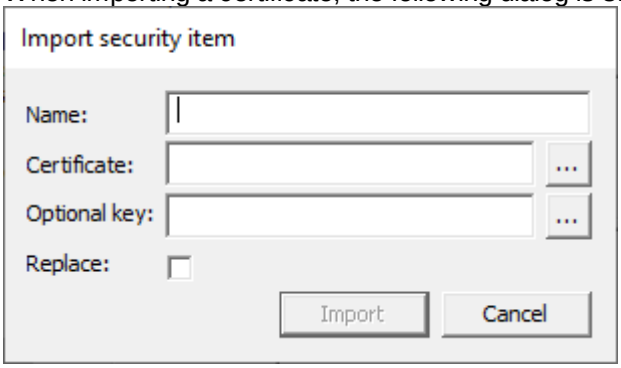
Certificates in the device is listed by name to the left, and information about the selected certificate is displayed to the right.

Refresh
This refreshes the information of certificates from the device.

Remove
This removes the selected certificate from the device.

Import
This imports a certificate to the device.

Import dialog
When importing a certificate, the following dialog is shown:



Name
This is the name to identify the certificate.

Certificate

The file name of a certificate. This file may optionally also include a private encryption key for the certificate.
The file must be encoded in PEM format. (See the PEM file format)

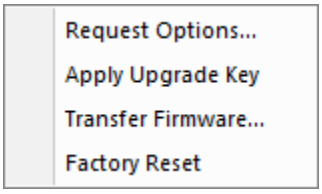
Optional key
The file name of a private encryption key, to include with the certificate.
The file must be encoded in PEM format. (See the PEM file format)

Replace
When this option is set the new certificate will replace an existing one if present.

2.3.9.4 Device: Maintenance

2.3.9.4.1 Device: Maintenance

By using the "Maintenance" menu, it is possible to manage options of a connected RTCU, upgrade firmware etc.



The individual items:

- Request options
- Apply upgrade key
- Transfer firmware
- Factory reset

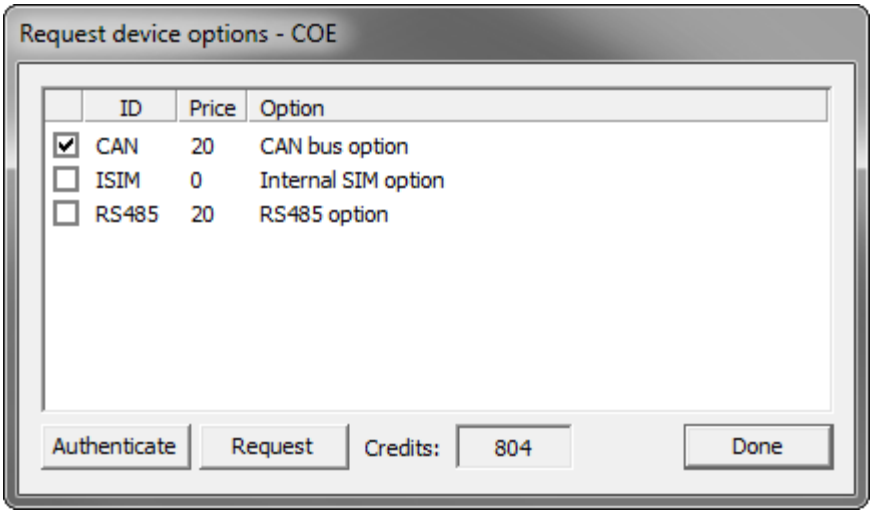
2.3.9.4.2 Maintenance: Request options

The RTCU concept utilizes a state-of the-art option concept, which allows actual hardware options to be applied on-demand to the device during installation time - or even over-the-air after installation.
Using for example the RTCU CX1 warp, additional digital I/O or communication options(RS232/FMI/1-wire) can be requested as an option at any given time.

When an option is requested, the options server at Logic IO is contacted through the internet by utilising the supplied account credentials. If there are sufficient credits for the requested option(s), the transaction will be made, and the options activated in the device.
To use the request option functionality, it is therefore necessary to have an account with a positive credit established at Logic IO with a given username/password.

The VPL program which runs inside the RTCU device can also request options using the boardRequestOption.

When the user has gone through the authentication process via the options server and a device is connected to the RTCU IDE, the list of available options for the specific device is listed:



Using this window allows you to request and apply options to a connected device.

List of options
This is a list of the options that are available for the connected RTCU device.
The list will be updated when the window is opened or when a new device with a different target is connected.

ID
This is the ID of the option. The ID can also be used with the boardRequestOption function to request the option from the device.

Price
This is the number of credits required to purchase the option.

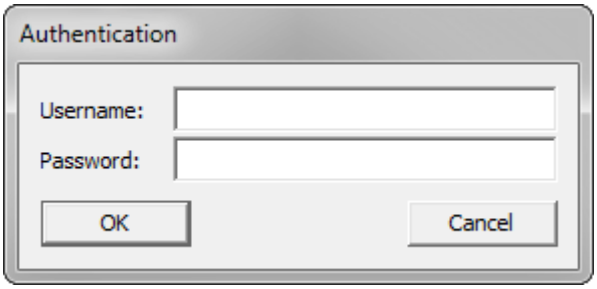
Option
This is the name of the option.

Credits
This is the number of credits available to purchase options.

Request
Pressing this button will request the selected options for the connected device and then update the device according to your selected options.
The window will ask for confirmation before the options are requested and the device updated.
Accepting at this point will deduct the purchase from the account.
Please note that once an option is requested for a device, that option can be included in future updates to the device without additional cost.

The request action will automatically be activated when a new device with the same target is connected. This allows easy bulk operation on a larger number of devices.

Authenticate
Pressing this button will change the authentication used when requesting the options.
For security reasons the credentials are not stored when the RTCU IDE is closed.

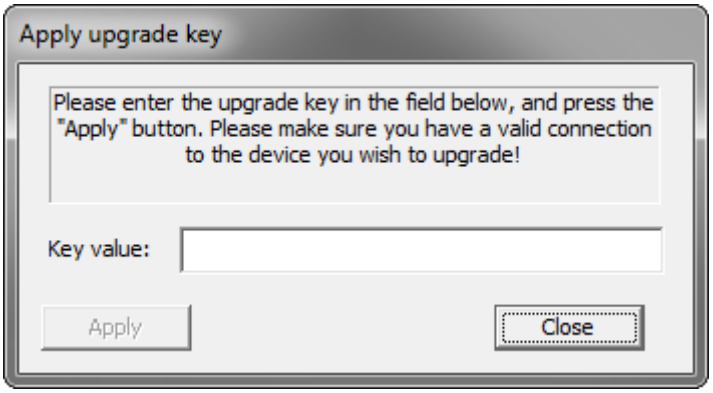


Username
This is the username for the account used.

Password
This is the password for the account used.

NOTE: Using this function will deduct the credits from the account used and constitutes a non-refundable purchase.

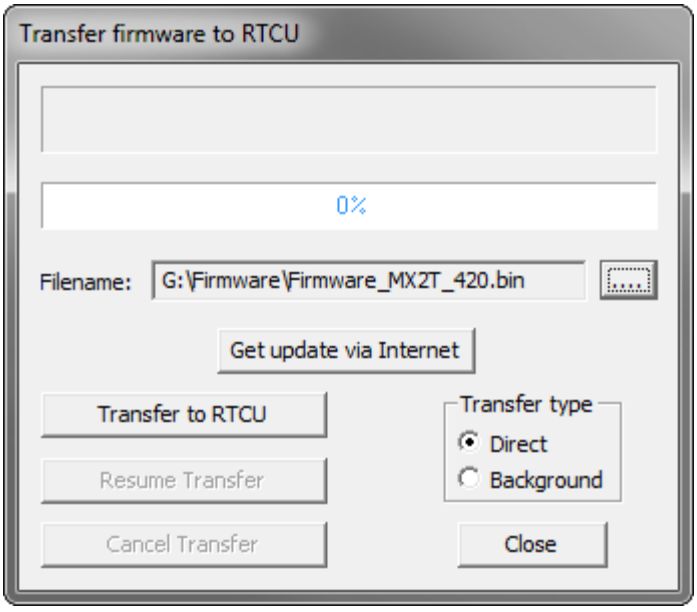
2.3.9.4.3 Maintenance: Apply upgrade key



By using this window, it is possible to enable certain features for the RTCU device. This key must be written in the entry field, and then by pressing the "Upgrade" button, the feature will be enabled in the connected RTCU device.
The upgrade key is device specific and based on the serial number of the device.
This functionality has been superseded by the Request options dialog.

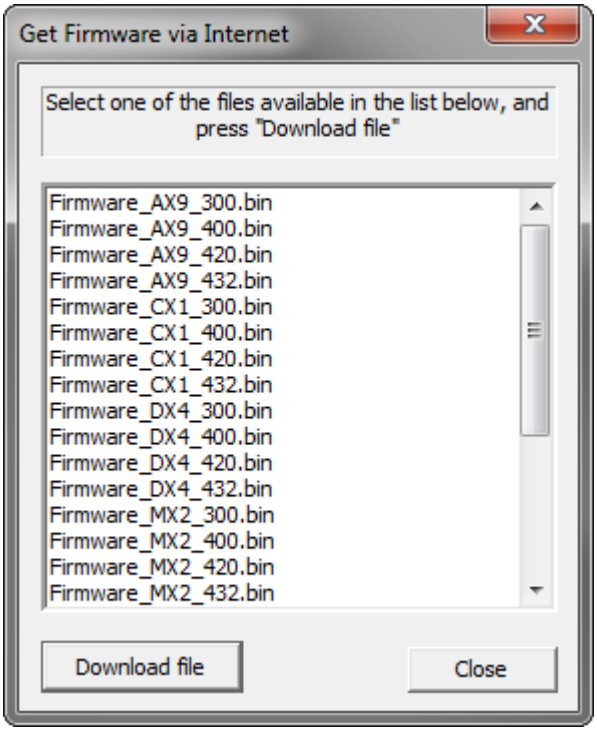
2.3.9.4.4 Maintenance: Transfer firmware

By using "Transfer firmware", it is possible to upload a new firmware revision to a connected RTCU. It is possible to select a file, or to get a list of available versions, by clicking the "Get update via Internet". This requires you to have a connection to the internet.



Depending on the type of RTCU and/or the connection type, you can choose to upload the firmware either as a "Direct" upload or as a "Background" upload. By using a Direct upload, you will halt the execution of the running application. Update mode will then be entered. If using the Background upload, the running application will continue to run during the upload. When the upload has finished the actual "switch" to the new firmware, this process will be finished when the device resets next time. One other advantage of the Background upload is that a failed/disconnected upload attempt can be resumed at a later time. When using the Background upload, the Voice section cannot be transferred, and the current Voice data in the device may be overwritten as the device uses the Voice flash memory for its operation. This means that if the application contains Voice data, and the background upload is used, the Voice data will have to be transferred separately after the Background upload has finished. Also see the verCheckUpgrade() function-block.

If you press the "Get update via Internet", you will be presented with a list of available firmware files as seen below:



It is very important that you select the correct file.

For Linux-based RTCU devices (LX / NX) there are several types of firmware files:

Runtime-Firmware.
This is the primary firmware that implements most of the functionality of the RTCU platform. This can best be compared with the single firmware on non-Linux based devices. This firmware file is most frequently updated.

System-Firmware.
This main firmware includes the complete Linux system and all the necessary files to run the system. It also includes a runtime firmware that can later be upgraded. This firmware is not frequently updated.

Monitor-Firmware.
This firmware is a "backup" firmware used if the System-firmware is not present or is under update. This firmware is rarely updated.

2.3.9.4.5 Maintenance: Factory reset

This will restore the device back to the factory default settings.

The factory default setting are:

- The persistent FLASH (except for the extended flash) memory is cleared.
- The persistent FRAM memory is cleared.
- The data logger is cleared.
- The TCP/IP settings are cleared.

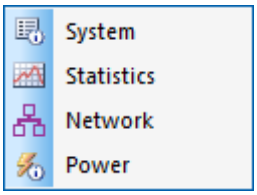
- The RTCU Communication Hub settings are cleared.
- Battery operation is disabled.
- Debug and network log to file is disabled.
- The internal flash drive is formatted.
- The time in the device is set to the PC time.

The application present in the device will not be changed.
Please note that this can be a lengthy operation that can take up to 5 minutes to complete.

2.3.9.5 Device: Information

2.3.9.5.1 Device: Information

By using the "Device - Information" menu, it is possible to query information from a connected RTCU.



The individual items:

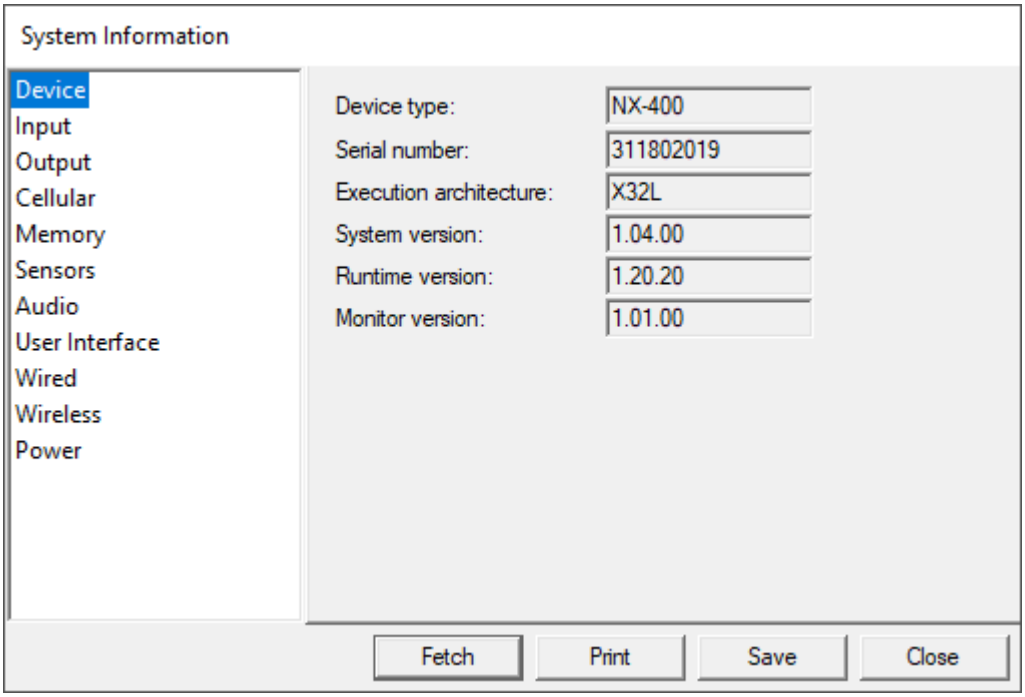
- System
- Statistics
- Network
- Power

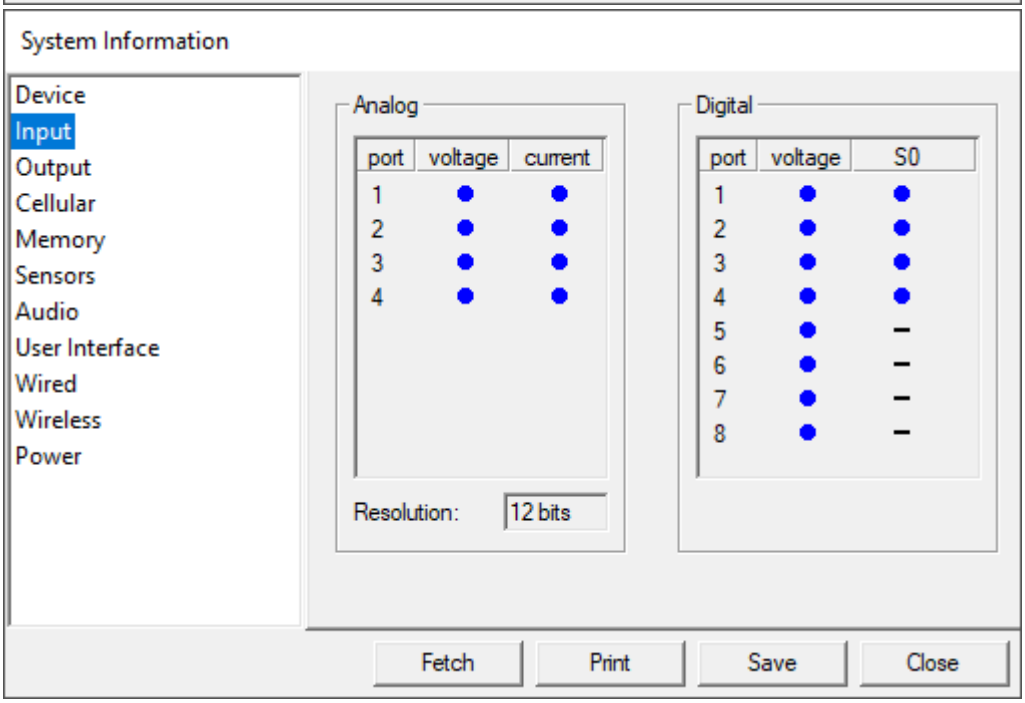
2.3.9.5.2 Information: System

This dialog can be used to query various system information parameter of the connected RTCU device.
The information available includes serial-number, device-type, peripherals available such as Cellular, GNSS, WLAN, CAN, Ethernet, number of I/O's etc.

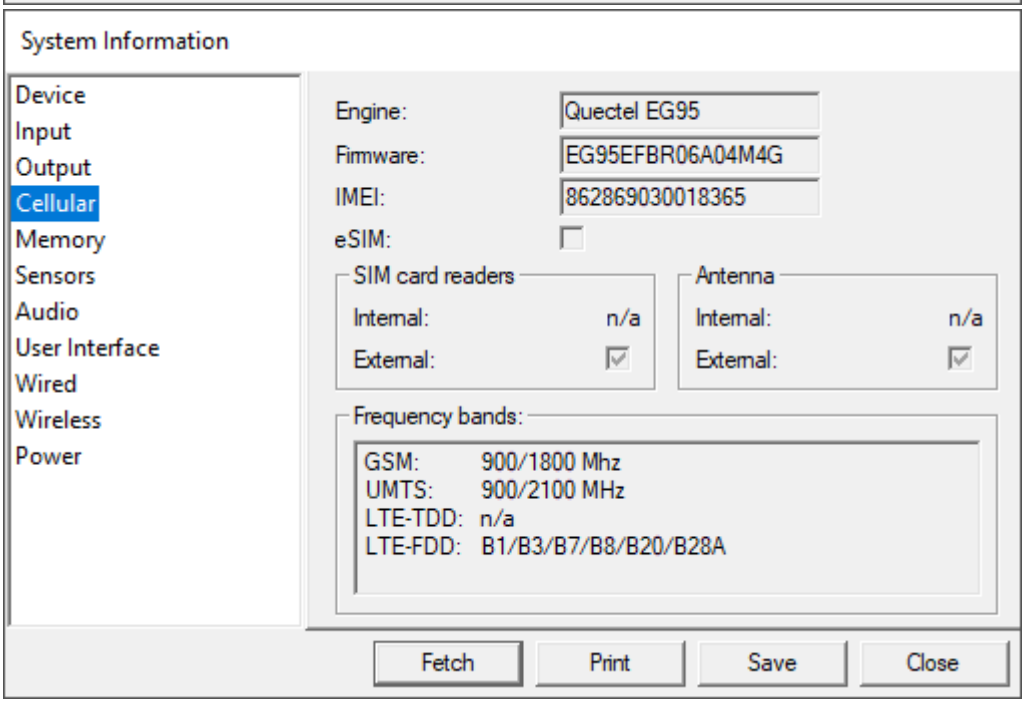
It is possible to print the system information with the "Print" button and equally it can be saved to a file using the "Save" button.

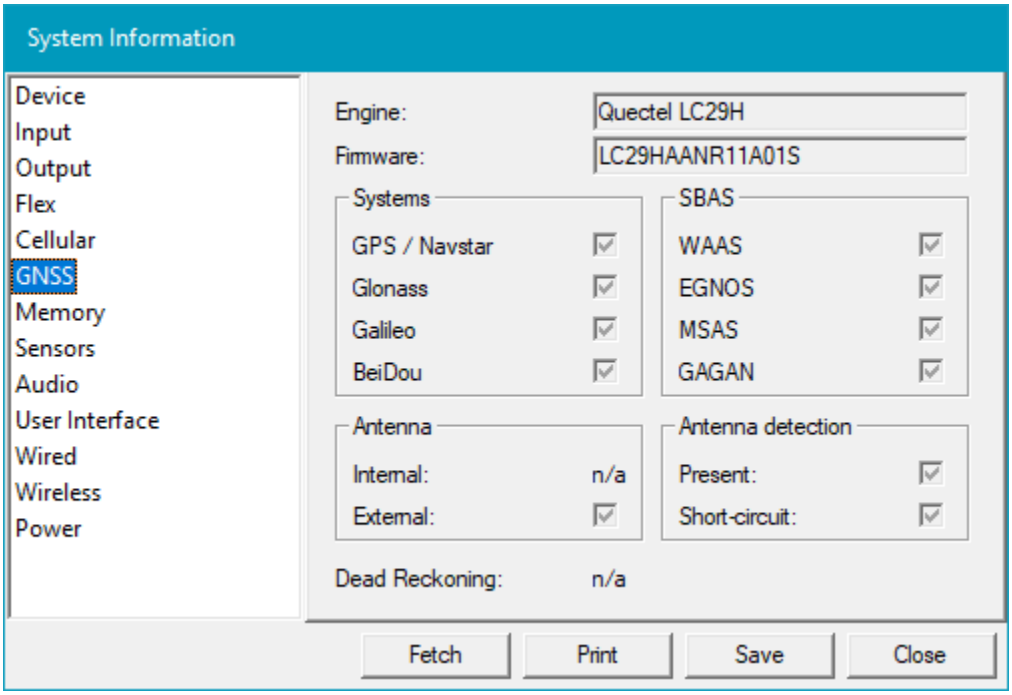
The system information utilizes the following symbols with the listed semantics:
- ● - The item is present.
- ○ - The item is optional,
- – - The item is not available.

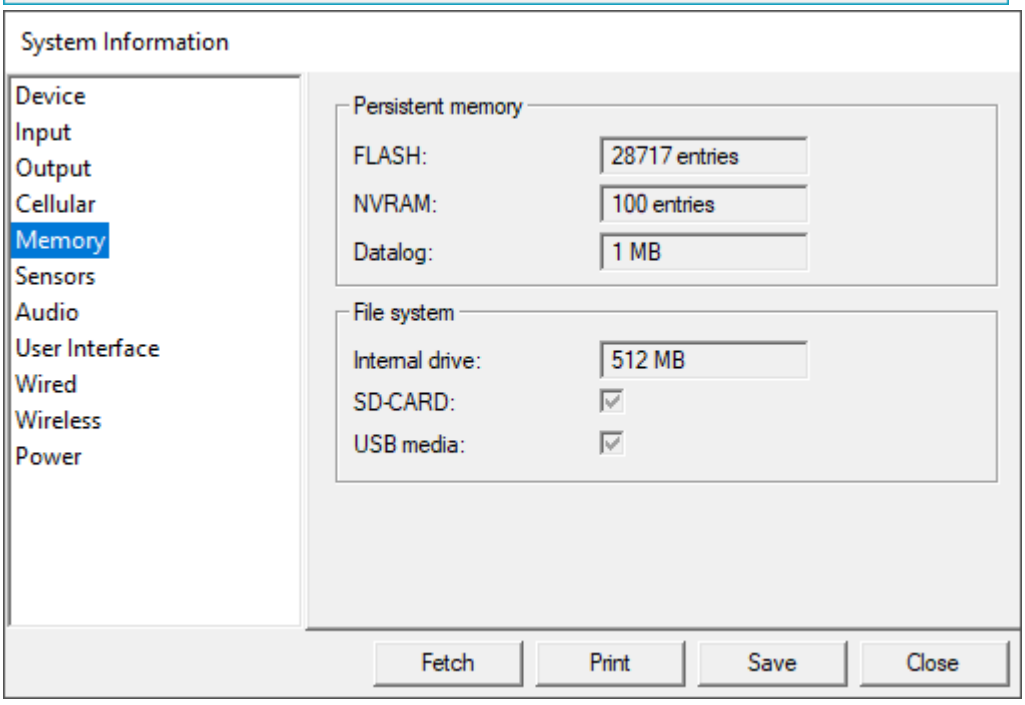


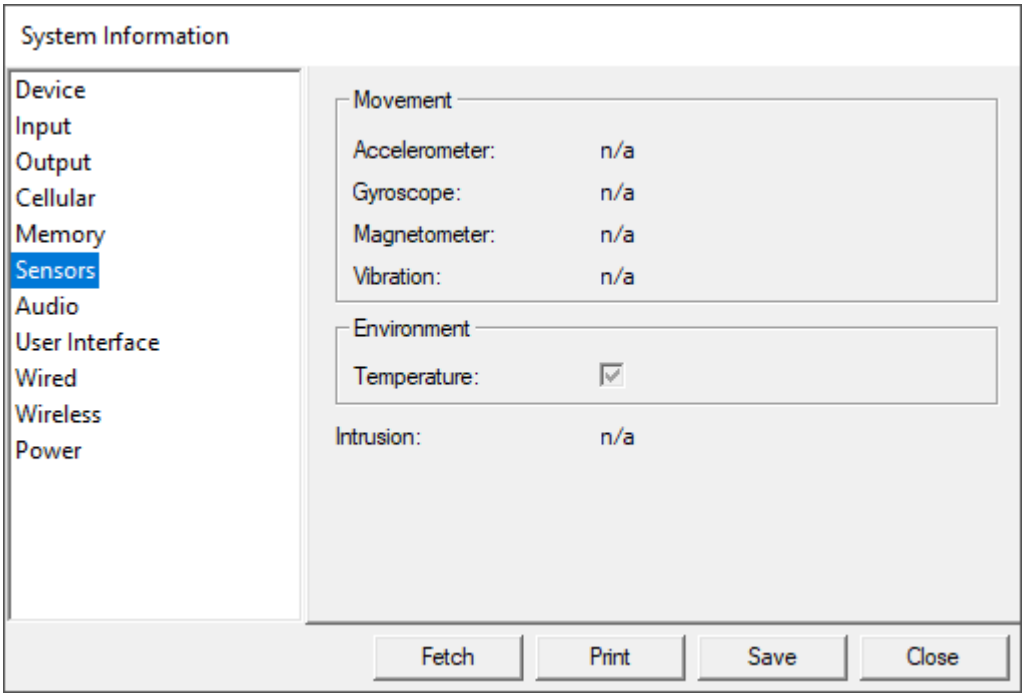


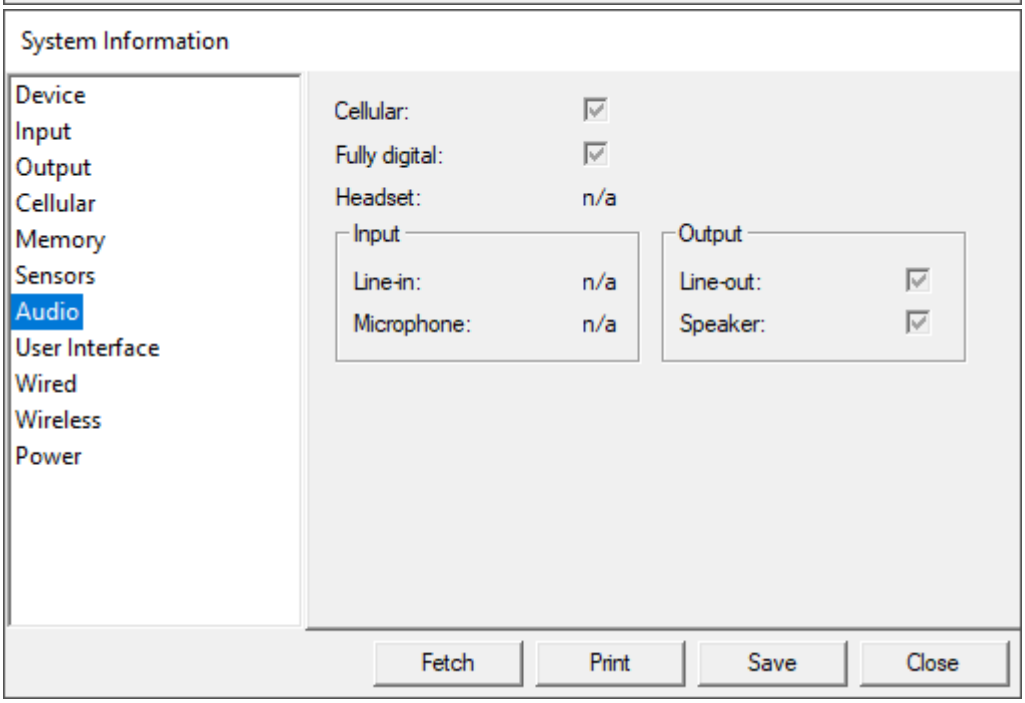

System Information
Device
Input
Output
Cellular
Memory
Sensors
Audio
User Interface
Wired
Wireless
Power
Analog
port
voltage
current
1
2
3
4
Resolution:
12 bits
Digital
port
solid state
relay
1
2
3
4
5
6
7
8
Fetch
Print
Save
Close

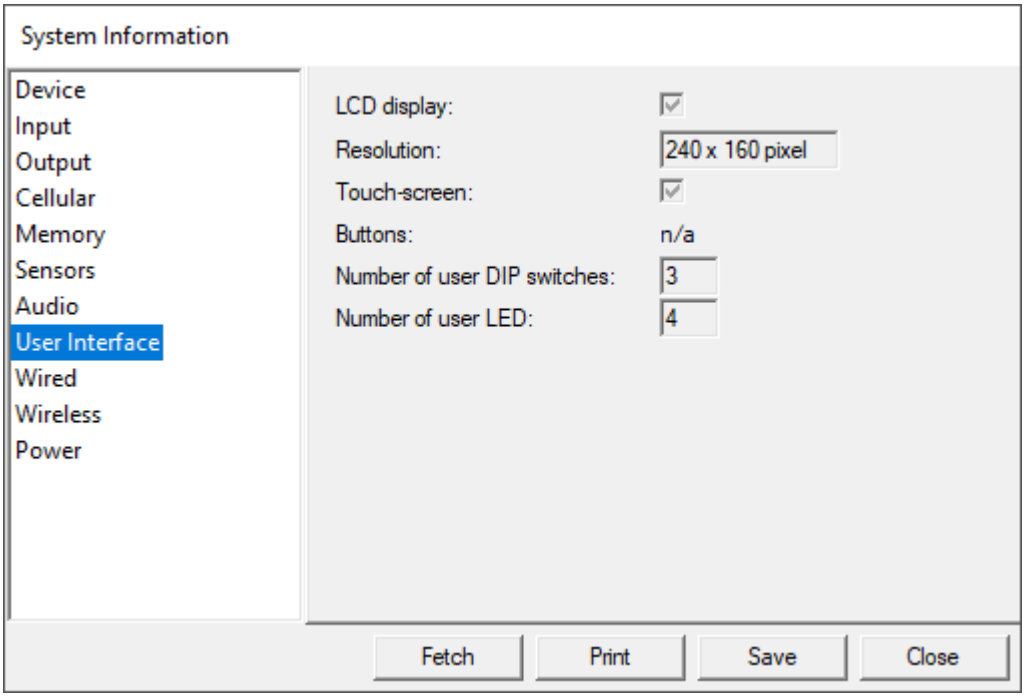

System Information
Device
Input
Output
Cellular
Memory
Sensors
Audio
User Interface
Wired
Wireless
Power
Engine:
Quectel EG95
Firmware:
EG95EFBR06A04M4G
IMEI:
862869030018365
eSIM:
SIM card readers
Internal:
n/a
External:
Antenna
Internal:
n/a
External:
Frequency bands:
GSM:
900/1800 Mhz
UMTS:
900/2100 MHz
LTE-TDD:
n/a
LTE-FDD:
B1/B3/B7/B8/B20/B28A
Fetch
Print
Save
Close

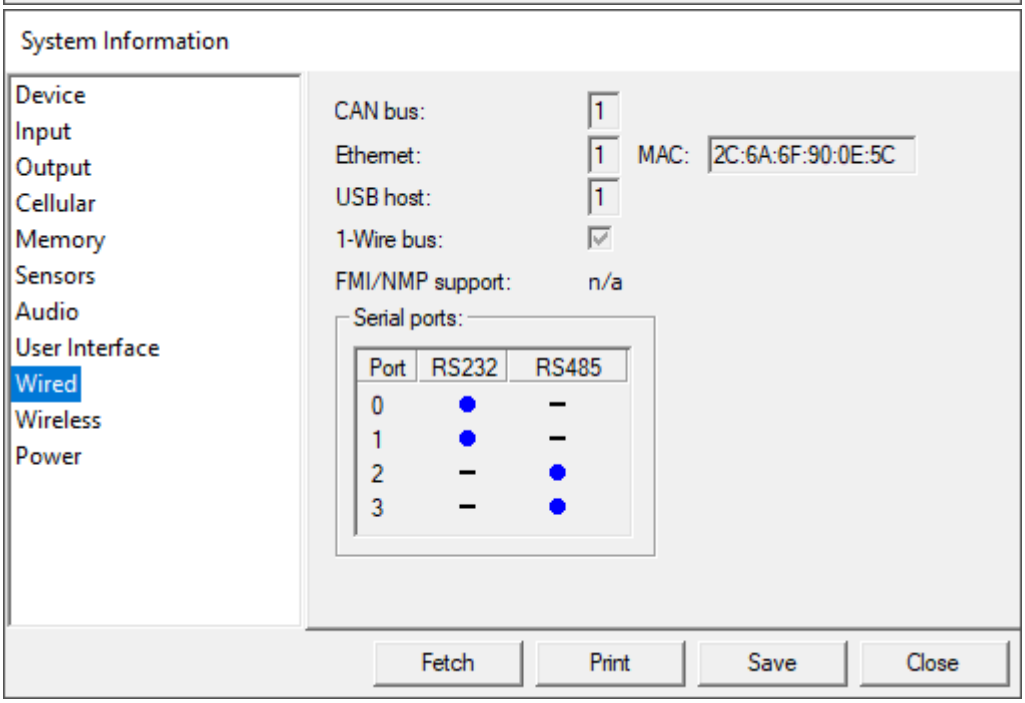


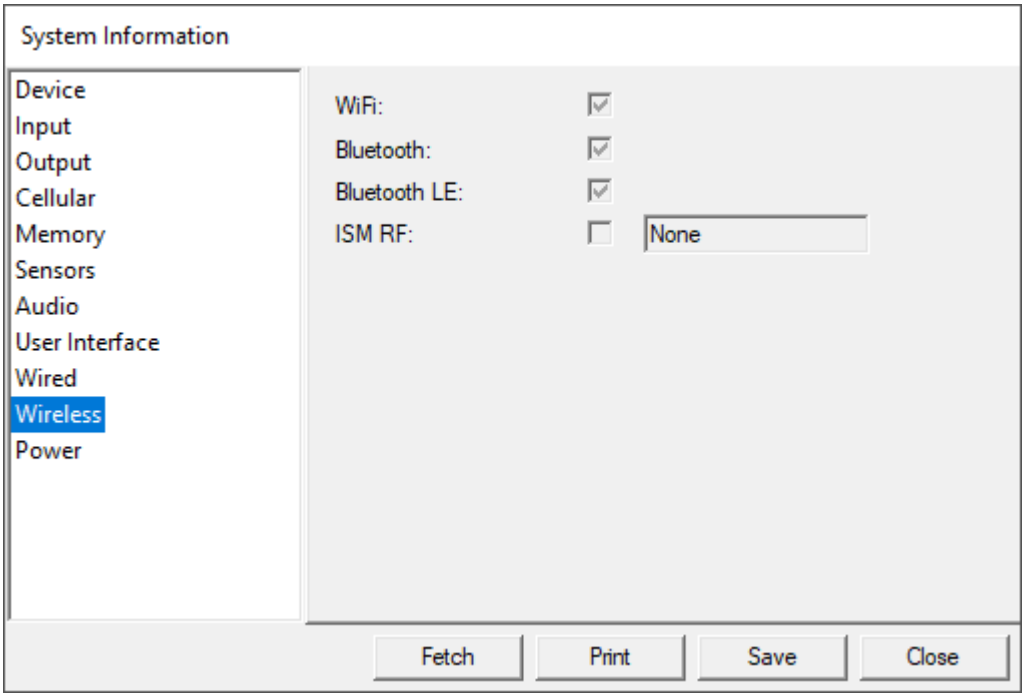


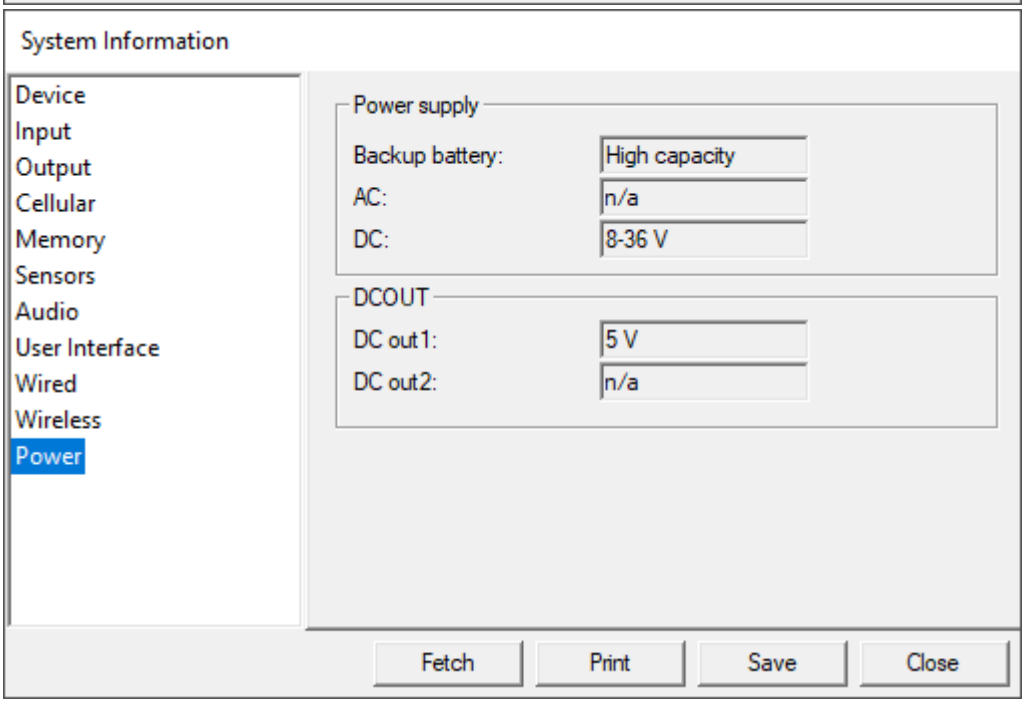

System Information
Device
Input
Output
Cellular
Memory
Sensors
Audio
User Interface
Wired
Wireless
Power
Movement
Accelerometer: n/a
Gyroscope: n/a
Magnetometer: n/a
Vibration: n/a
Environment
Temperature:
Intrusion: n/a
Fetch
Print
Save
Close


System Information
Device
Input
Output
Cellular
Memory
Sensors
Audio
User Interface
Wired
Wireless
Power
Cellular:
Fully digital:
Headset: n/a
Input
Line-in: n/a
Microphone: n/a
Output
Line-out:
Speaker:
Fetch
Print
Save
Close

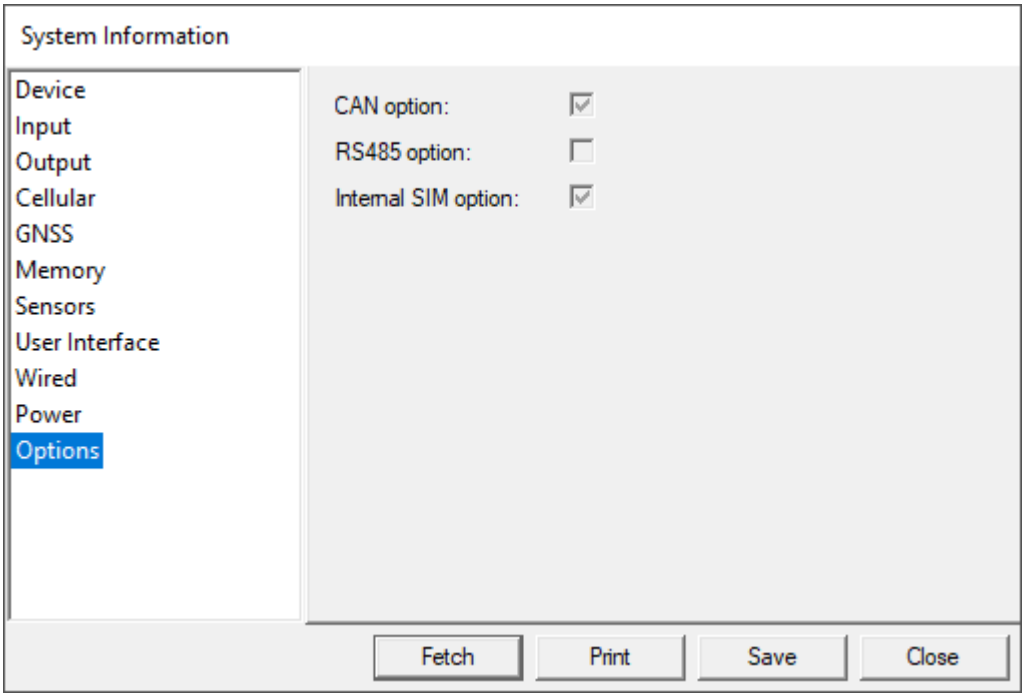






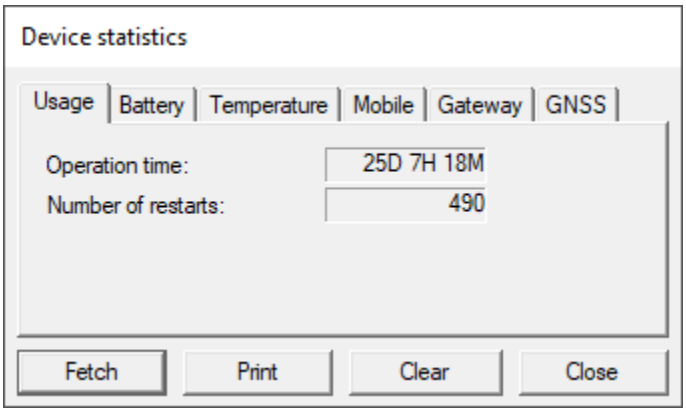


When the system information is requested for a device supporting additional options(see Request options), a page with the options is included.



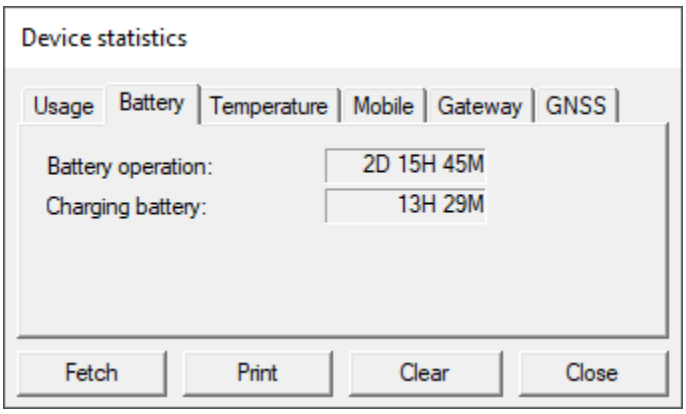
2.3.9.5.3 Information: Statistics

This dialog can be used to query system statistics of the connected RTCU device.
The device statistics contains lifetime information reset only during the production of the device, and it cannot be reset.



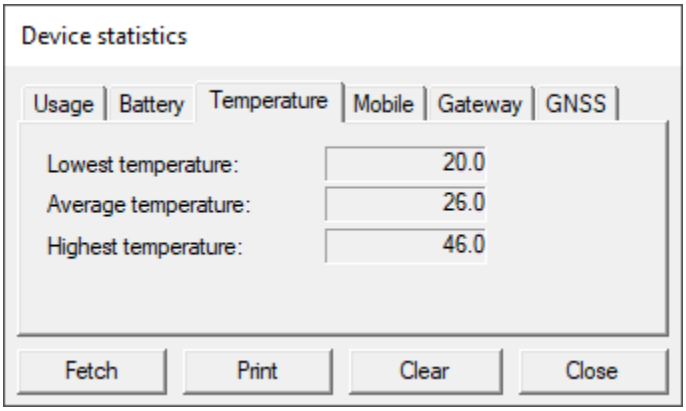
Operation time
This is the total amount of time that the RTCU device has been operating.

Number of restarts
The number of times the device has rebooted.



Battery operation
This is the total time of running from the backup battery instead of from external power.

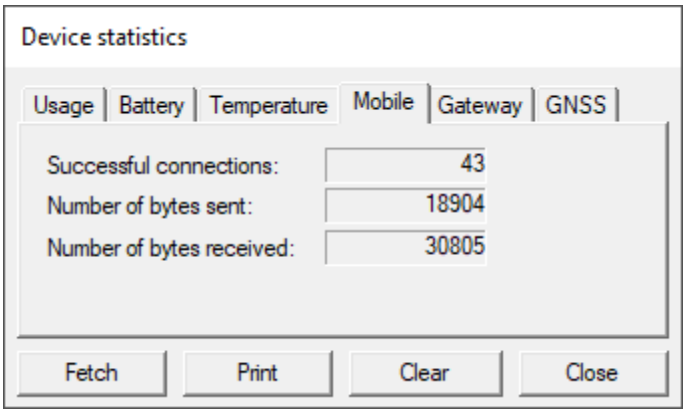
Charging battery
This is the total time the backup battery has been charged.



Lowest temperature
This is the lowest temperature measured in the RTCU device.

Average temperature
This is the average temperature measured in the RTCU device.

Highest temperature
This is the highest temperature measured in the RTCU device.

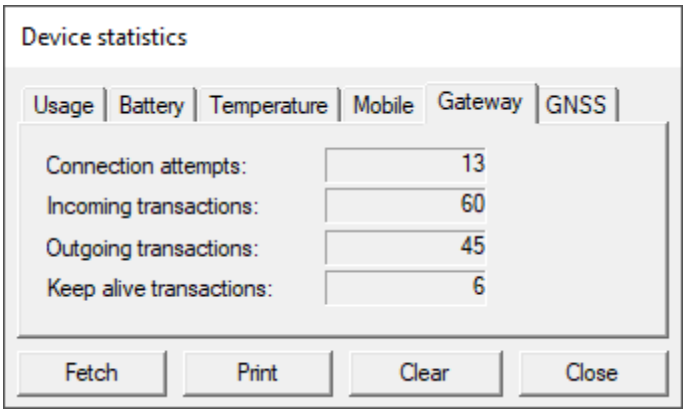


Successful connections

This is the number of times the RTCU device successfully connected to a mobile network.

Number of bytes sent
This is the number of bytes sent through the mobile network connection.

Number of bytes received
This is the number of bytes received from the mobile network connection.

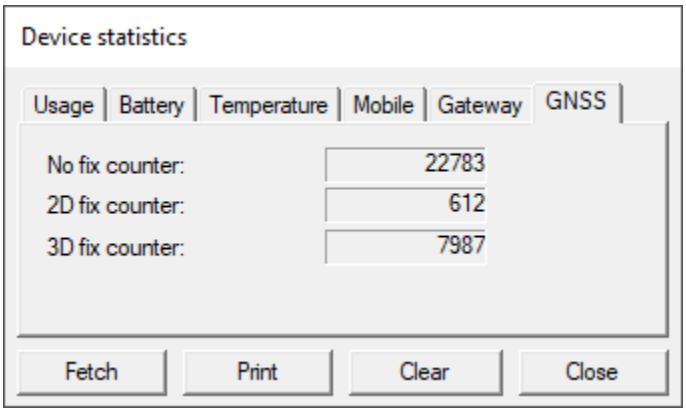


Connection attempts
This is the number of times the RTCU device has tried to connect to the RTCU Communication Hub.

Incoming transactions
This is the number of incoming transactions.
Incoming transactions are generated when the RTCU device receives data from the RTCU Communication Hub.

Outgoing transactions
This is the number of outgoing transactions.
Outgoing transaction are generated when the RTCU device sends data to the RTCU Communication Hub.

Keep alive transactions
This is the number of keep alive transactions the RTCU device has sent through the RTCU Communication Hub.



No fix counter
This is the number of times the GNSS module has reported no fix.

2D fix counter
This is the number of times the GNSS module has reported a 2D fix.
(GPS mode = 2)

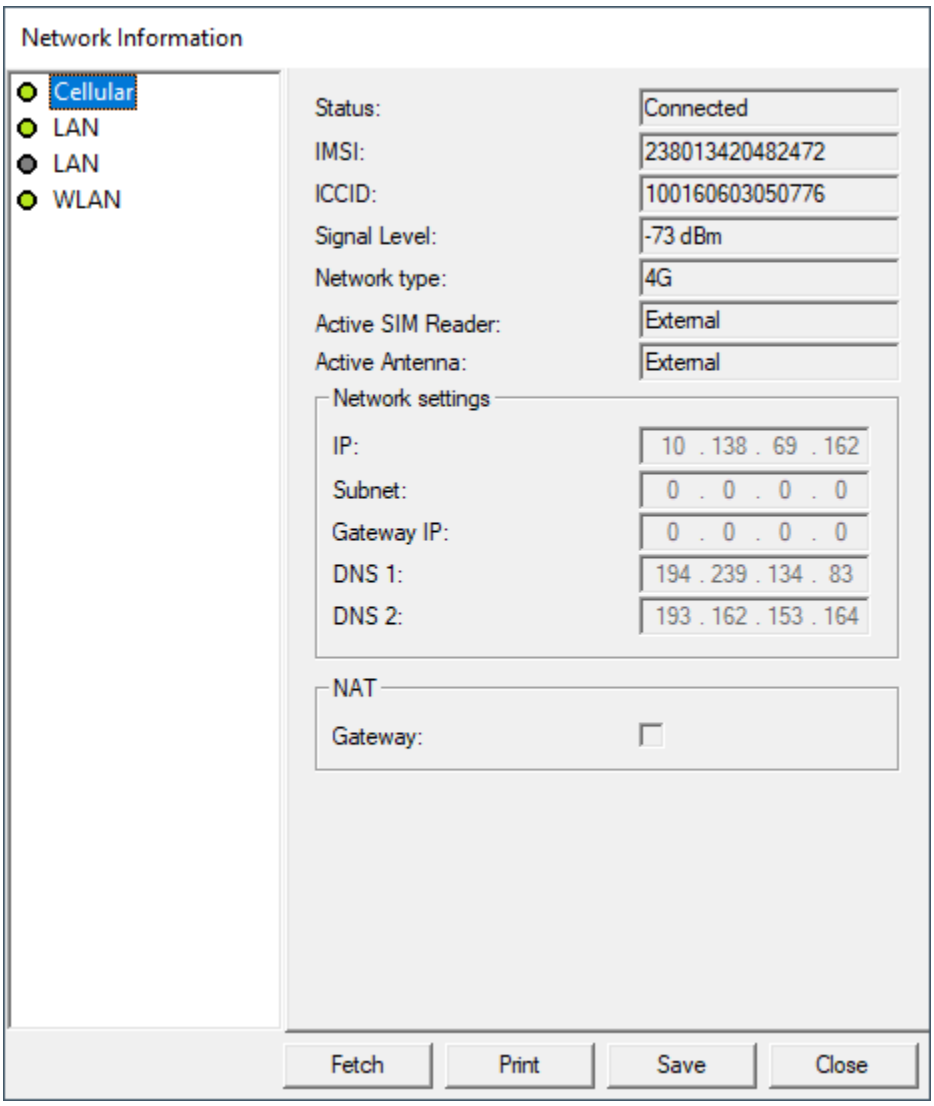
3D fix counter
This is the number of times the GNSS module has reported a 3D fix (mode = 3 or 4).

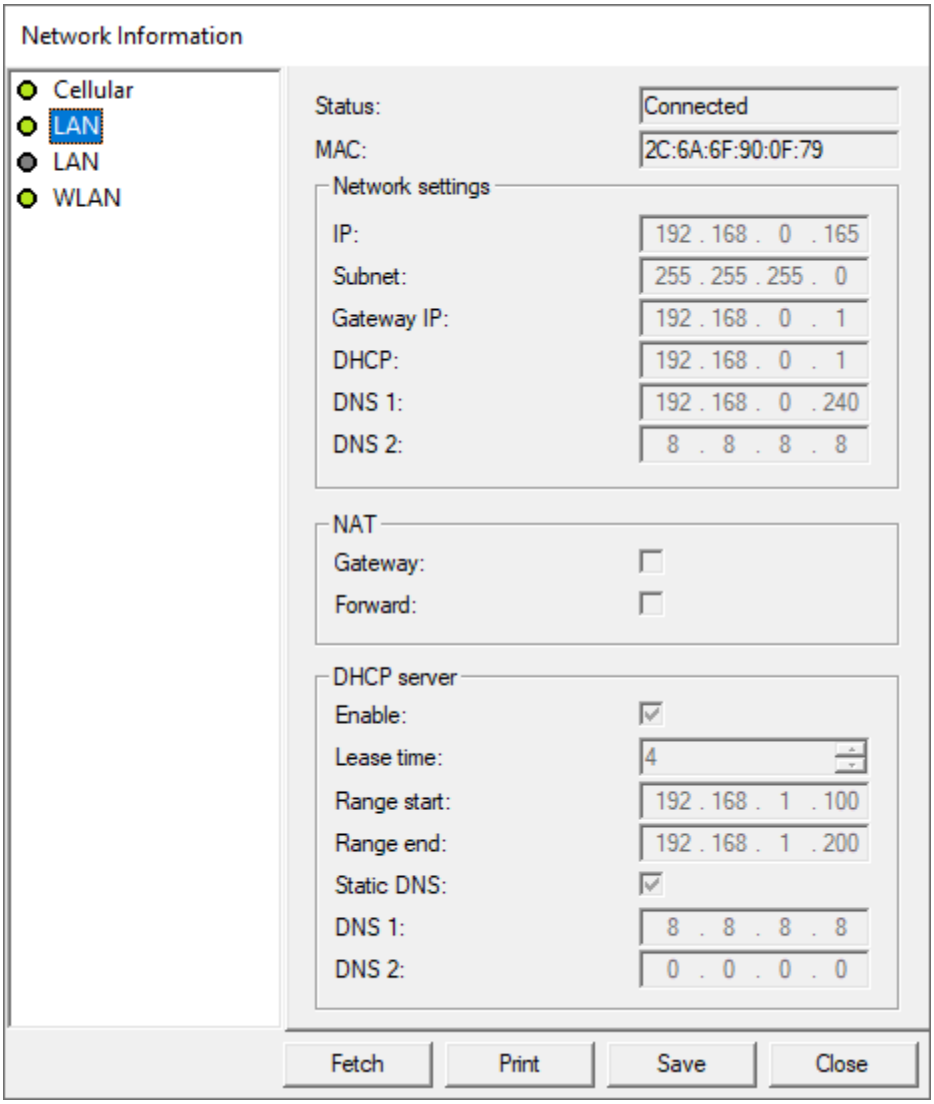
2.3.9.5.4 Information: Network

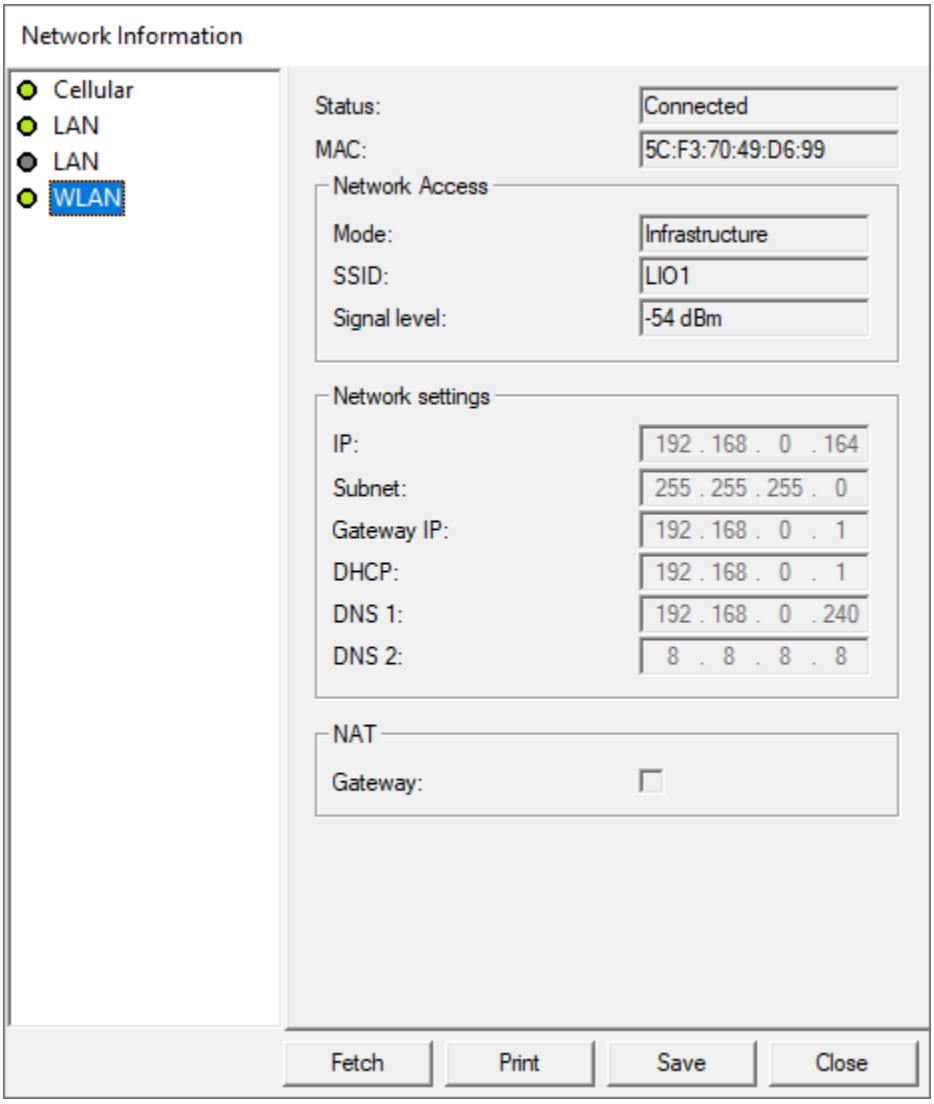
This dialog shows information about the available networks of the connected RTCU device.

Preceding the name of the network status icon is shown with the following semantics:

- (Grey) - The network is not present or not powered on (Cellular only).
- (Blue) - The network is not open.
- (Red) - The network is not connected.
- (Yellow) - The network link is up, but not connected.
- (Green) - The network is connected







Status:
The status of the connection.

No power	- The network is not powered on. (Cellular only)
Not present	- The network is not present.
Not open	- The network is not open.
Not connected	- The network is not connected.
Link present	- The network link is up, but not connected.
Connected	- The network is connected.

IMSI:
The IMSI number of the SIM card used. Only the last 15 characters are included.

ICCID:
The ICCID number of the SIM card used.

Signal Level:
The signal strength of the connection to the network.

Network Type:
The service type used by the cellular network.

Active SIM Reader:
The SIM card reader used.

Active antenna:
The GSM antenna used.

Mode:
The operation mode of the wireless network.
Infrastructure - The device is connecting to an access point.

SSID:
The SSID of the connected wireless network.

MAC:
The MAC address of the network.

IP:
The IP address of the network.

Subnet:
The subnet of the network.

Gateway:
The IP address of the RTCU Communication Hub.

DHCP:
The IP address of the DHCP server; if the DHCP server is not used the address will be zero.

DNS 1:
The IP address of the primary DNS server.

DNS 2:
The IP address of the secondary DNS server.

NAT
This group contains the NAT features used by the interface. (See network for information)

Forward:
If this is set, socket connections will be forwarded to the Gateway interface.

Gateway:
If this is set, the interface will act as a gateway to the internet.

DHCP server
This group contains the parameters used by the DHCP server for the interface (See network for information)

Enable:
If this is set, the DHCP server will listen for requests from the interface

Lease time:
The number of hours before a client must renew their address.

Range start:
The range of dynamic addresses starts with this address.

Range end:

The range of dynamic addresses ends with this address.

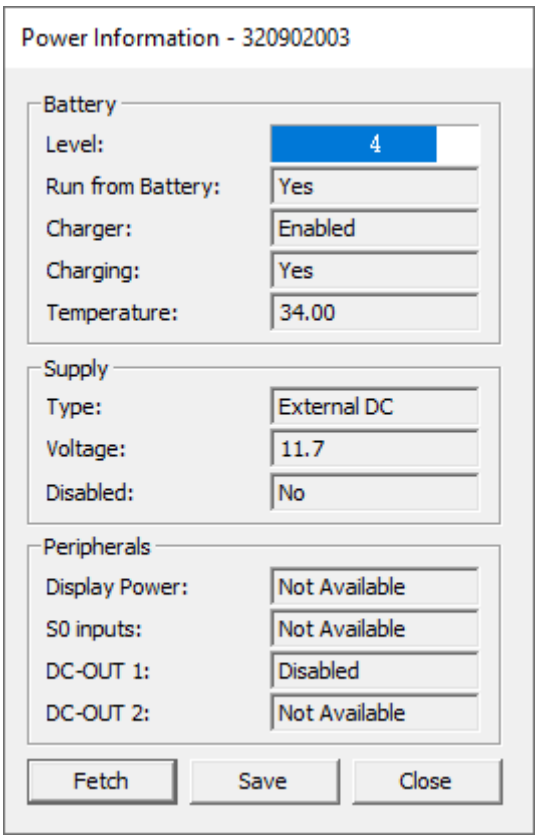
Static DNS:
If this is set, the DNS servers with the following parameters are used:

DNS 1:
The IP address of DNS server 1.

DNS 2:
The IP address of DNS server 2.

2.3.9.5.5 Information: Power

This dialog shows information about the power supply of the connected RTCU device.



Battery
This group contains information about the backup battery and the battery charger.

Level:
The power level of the battery, given as a number between 0 and 5. Where 5 is maximum power.
(This is the same as the VPL function batPowerLevel)

Run from Battery:
This shows whether the device will continue to operate on internal battery when external power is lost (power fail).

Charger:
This shows whether the charging of the on-board battery is enabled or disabled.

Charging:
This shows whether the on-board battery is being charged or not.
(This is the same as the VPL function batIsCharging)

Temperature:
The measured temperature inside the device in degrees Celsius.
(This is the same as the VPL function boardTemperature)

Supply
This group contains information about the power supply

Type:
The type of power (battery, DC, or AC) the device is currently operating on
(This is the same as the VPL function boardSupplyType)

Voltage:
The measured voltage the device is supplied with, in volt.
(This is the same as the VPL function boardSupplyVoltage)

Disabled:
This shows whether the external power is turned off.

Peripherals
This group contains information about the peripheral features

Display Power:
This shows whether the display power is turned on.

S0 inputs:
This shows whether the IEC62053-31 compliant S0 interface is enabled.

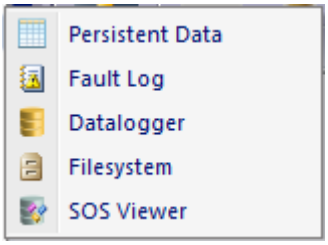
DC-OUT 1:
This shows whether the DC-OUT external power supply is enabled.

DC-OUT 2:
This shows whether the DC-OUT 2 external power supply is enabled.

2.3.9.6 Device: Data

2.3.9.6.1 Device: Data

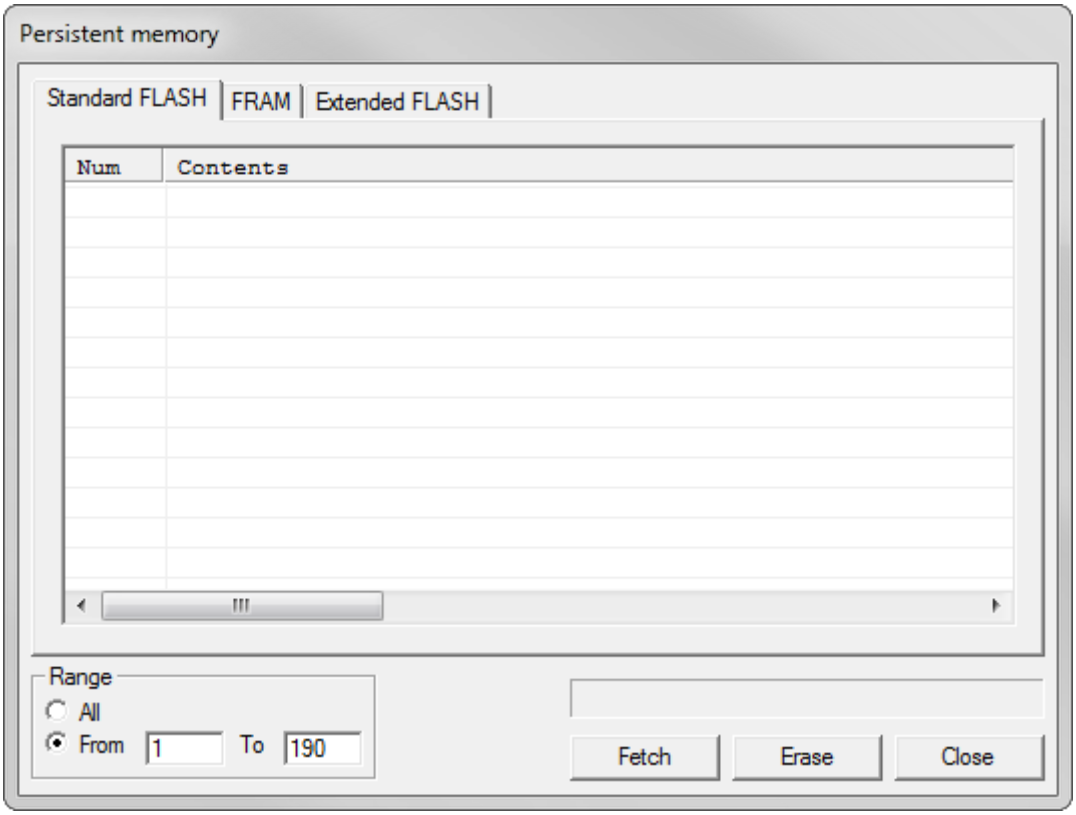
The "Device: Data" menu contains a number of functions that allows the user manipulate/view different types of data in a connected RTCU device.



The individual items:

- Persistent Data
- Fault log
- Datalogger
- Filesystem
- SOS Viewer

2.3.9.6.2 Data: Persistent



By using this dialog, it is possible to view and modify the contents of the persistent memory, FLASH, and FRAM in a connected RTCU device.

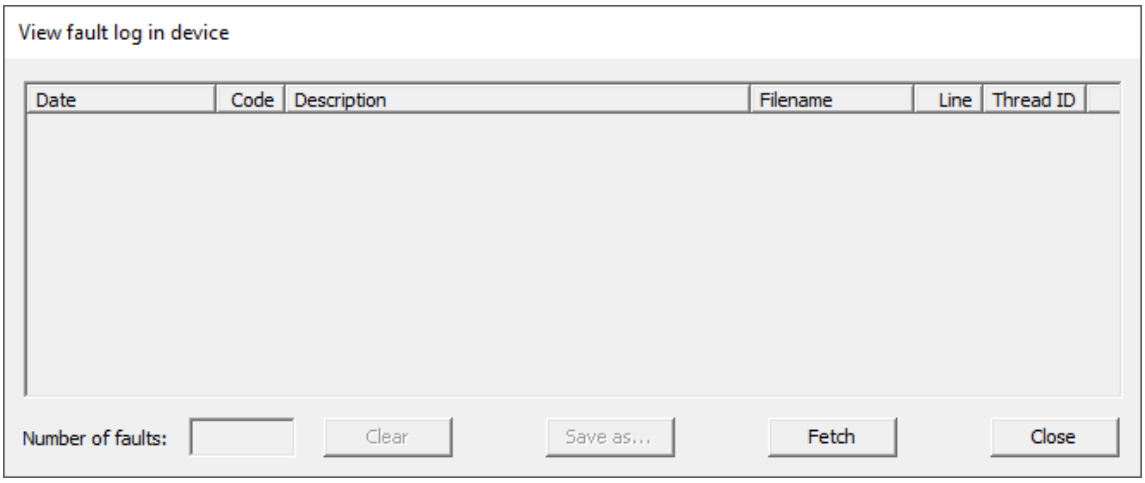
Press the "Fetch" button to fetch all or a range of persistent strings.
Press the "Erase" button to clear all or a range of persistent strings.

Use the controls in the "Range" to determine whether all persistent strings or just a range of persistent strings are Fetched or Erased.
By right-clicking inside the list, a menu will appear. By using this menu, it is possible to modify, delete, and create new entries in the persistent memory.

Note:
In standard FLASH, the range spans from 1 to 192.
In FRAM, the range spans from 1 to 20 or from 1 to 100 (on NX32 / NX32L devices)
The extended FLASH range depends on the device in use and may span up to 32600 entries.

Please refer to the Persistent Memory section for more information.

2.3.9.6.3 Data: Fault Log



By using this dialog, it is possible to view the fault log for a connected RTCU device. The fault log list shows when and what types of errors occurred in the RTCU device. When an error occurs within an RTCU device, it will be signalled on the status LED of the device, and an entry will be written in the fault log of the device. This fault record can be viewed using this dialog, and it can also be cleared.

For a list and explanation of the fault codes, please refer to the Troubleshooting section, Fault codes.

Date
This is the time and date when the fault occurred.

Code
This is a code number to identify the fault.

Description
This is a short text to describe the fault.

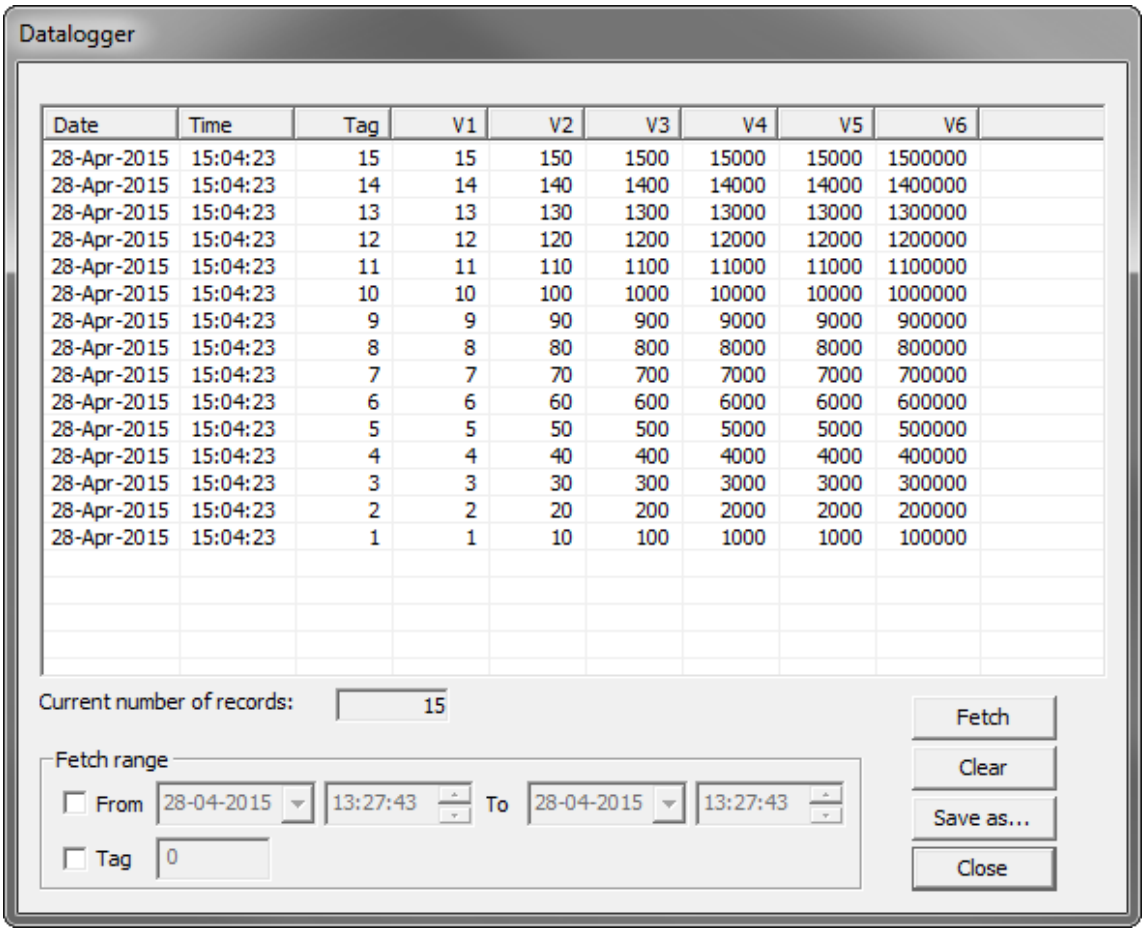
Filename
This is the name of the source file where the fault occurred.
(only available if the application is build with the Debug option)

Line
This is the approximate line number in the source file where the fault occurred.
(only available if the application is build with the Debug option)

Thread ID
This is the ID of the VPL thread where the fault occurred. This ID is identical to the one returned by the thGetID function.
(only available if the application is build with the Debug option)

The Debug option is enabled in the Project settings before building the application.

2.3.9.6.4 Data: Datalogger



By using this window, it is possible to view the contents of the Datalogger in a connected RTCU device.
The list of records will automatically adjust itself with respect to the number of items logged in each record.

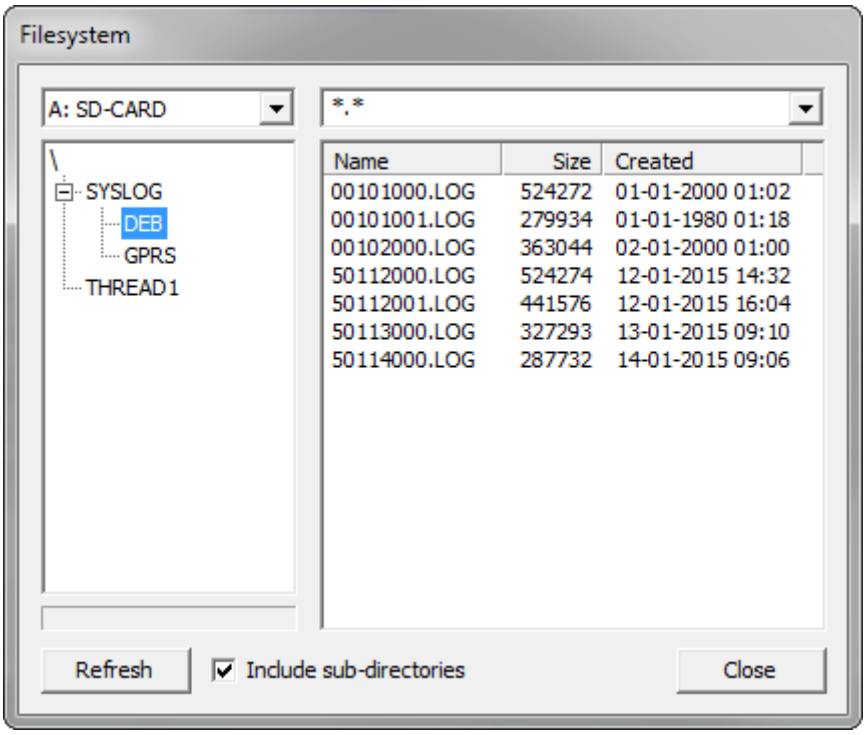
The "Fetch range" section allows filtering the records fetched.

- "From/To" limits the records to a range determined by a "Start" and "Stop" time.
- "Tag" limits the records to a particular Tag.

The "Clear" button empties the Datalogger.
The "Save as..." button saves the data in a comma-separated file for easy viewing in various spreadsheet programs etc.

Please refer to the Datalogger section for more information.

2.3.9.6.5 Data: Filesystem



The "Filesystem" window allows for remote access to the file system of the RTCU devices.

The file system is displayed in a way that is similar to Windows Explorer with the directories to the left in a tree structure and files to the right in a detailed list.

The drop-down list above the directory tree is used to select which media is displayed.

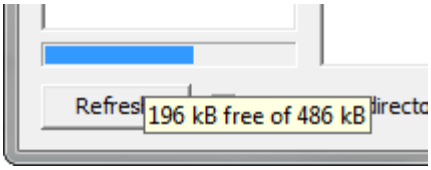
The drop down above the file list is used to filter the files transferred when the Refresh button is pressed. For example only *.log files could be fetched saving bandwidth and reducing the time for update. The last 5 filters are stored, as well as a default filter.

If the selected media is not available on the connected device, the directory tree will say "Not Available".

Refresh
This refreshes the information of files and sub-directories of the selected directory.
Refresh the root directory ("\" or "No card" or "Not Present") to update the info of all directories and files.
The file filter is applied to the files, and supports the wildcard characters * (zero or more characters) and ? (one character).
Removable media connected on the USB host will be reported as not present, until the usbHostEnable function is called to enable the USB host port.

Include sub-directories
If this box is unchecked, only the content of the selected directory will be refreshed when "Refresh" is clicked.

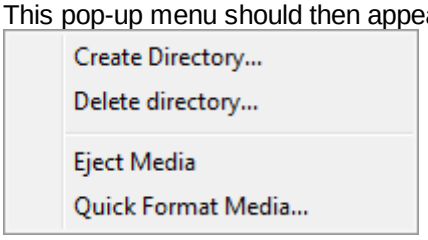
Media Space
At the bottom of the directory tree is a bar which shows how much of the total space on the selected media is used.
This information is updated when using Refresh.



By holding the cursor above the bar will show the space in kB.
If the information is not available, ?? will be shown instead of the value. For example, before pushing refresh the text will show: "?? kB free of ?? kB".

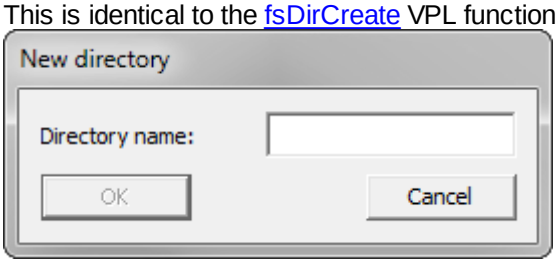
On a compressed media, as found on NX32L devices, the free space reported is the actual physical memory available without compression taking into account. The actual compression ratio depends on the content of the data stored.
Please note that free space is not available for an SD-CARD; this is because it can take a long time to update for large cards.

Directory functions
The directory functions are used by right-clicking in the directory tree.
This pop-up menu should then appear:



Note: some of the functions require a directory to be selected (eight-click on the directory name).
The functions are described below:

Create Directory
This create a new directory at the selected directory.
This is identical to the fsDirCreate VPL function.

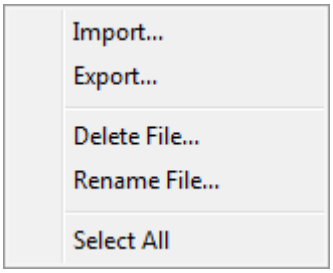


Delete Directory
This deletes the selected directory.
This is identical to the fsDirDelete VPL function.
It is also possible to delete a directory by using the "delete" key.

Eject Media
This ejects the media from the device.
This is identical to the fsMediaEject VPL function.

Quick format media
This quick formats the media. All directories and files are lost when this function is used.
This is identical to the fsMediaQuickFormat VPL function.

File functions
The file functions are used by right-clicking in the file list.
This brings up the following pop-up menu:



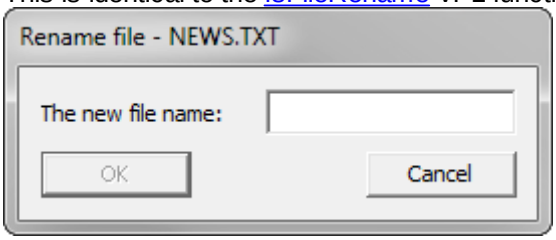
Note: Some of the functions require a file to be selected (right-click on the file name).
The functions are described below:

Import...
This imports files to the selected directory.

Export...
This exports the selected files from the device.

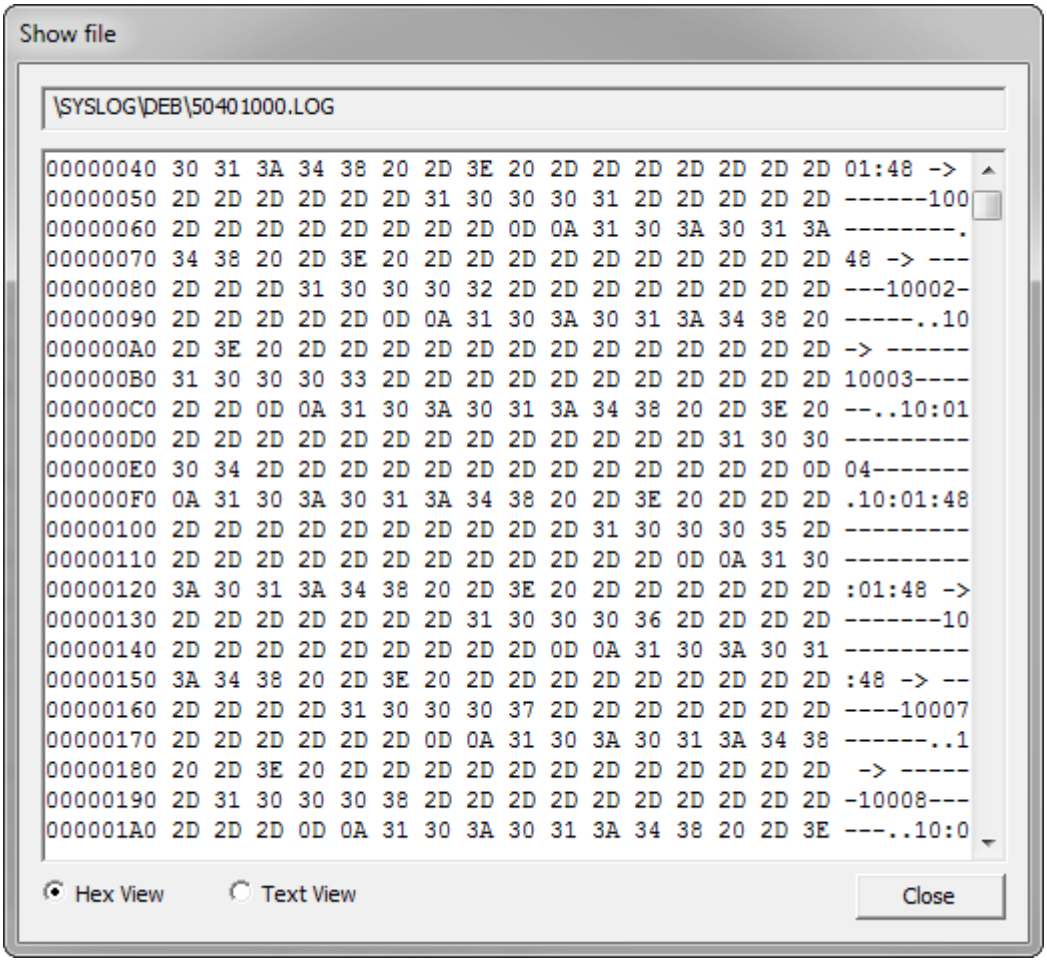
Delete File
This deletes the selected files from the device.
This is identical to the fsFileDelete VPL function.
It is also possible to delete files by using the "delete" key

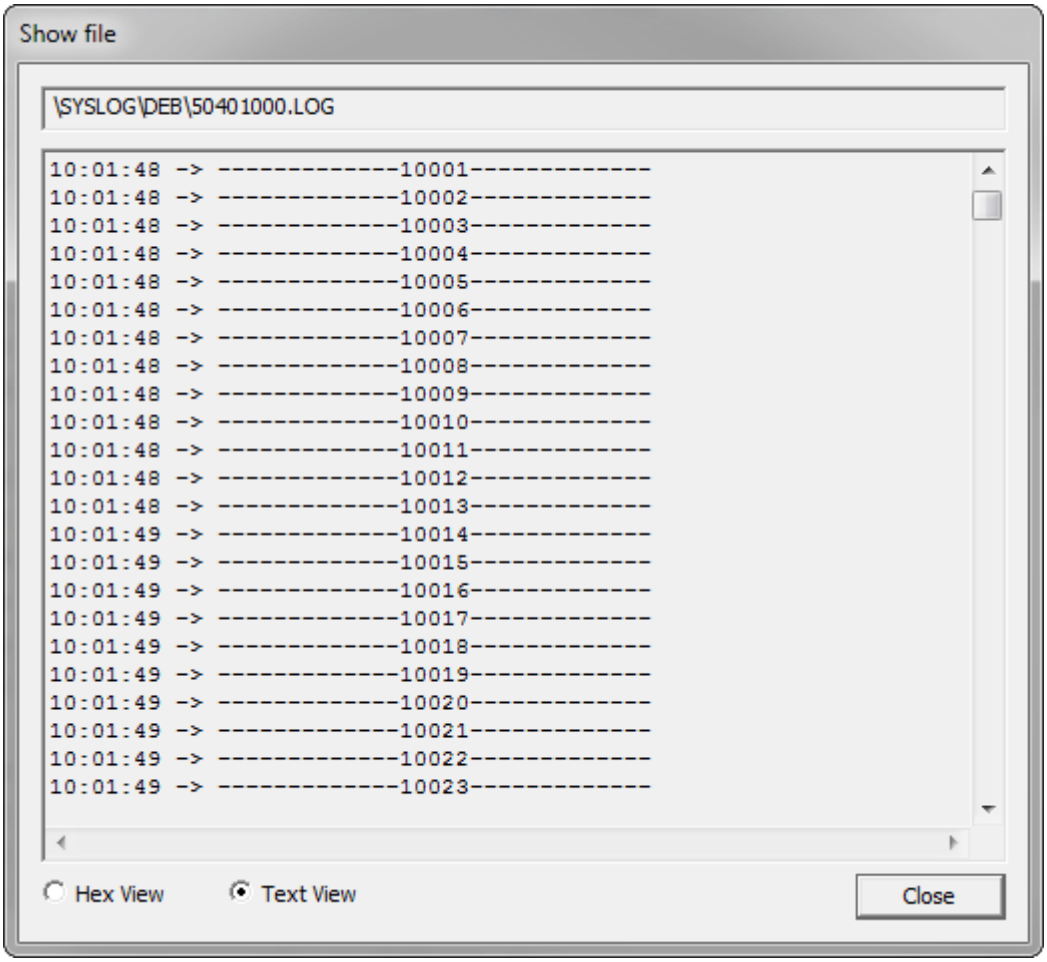
Rename File
This makes it possible to rename the selected file. If more than one file is selected, only the first one is renamed.
This is identical to the fsFileRename VPL function.



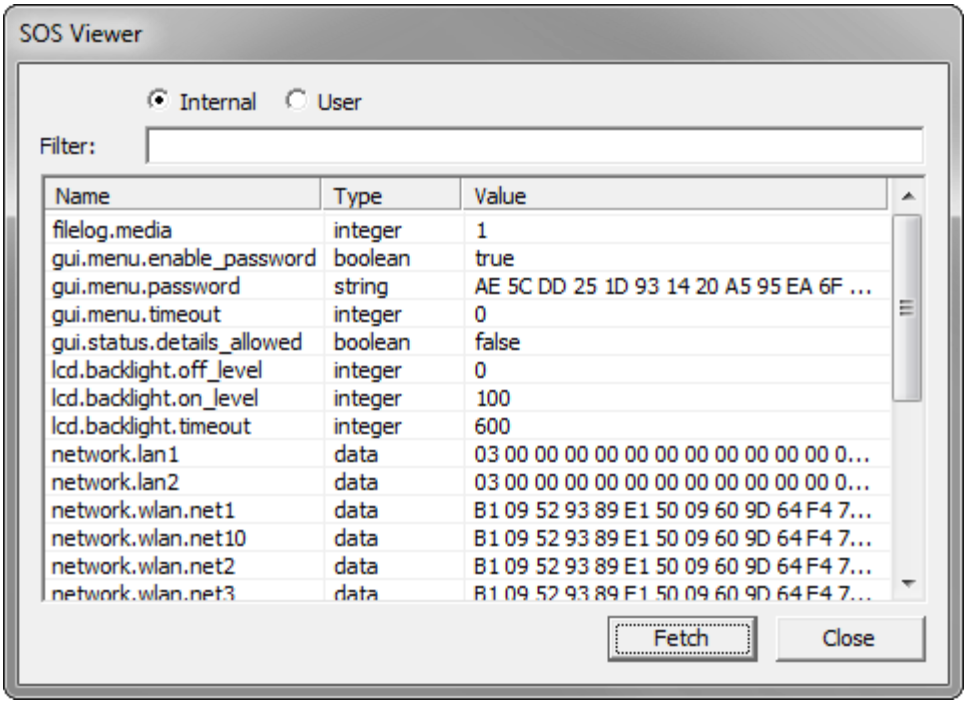
Select All
This selects all files in the directory.

Show file
It is possible to see the contents of a file by double-clicking on its name.


Show file
\SYSLOG\DEB\50401000.LOG
00000040 30 31 3A 34 38 20 2D 3E 20 2D 2D 2D 2D 2D 2D 01:48 ->
00000050 2D 2D 2D 2D 2D 2D 31 30 30 30 31 2D 2D 2D 2D 2D ------100
00000060 2D 2D 2D 2D 2D 2D 2D 2D 0D 0A 31 30 3A 30 31 3A --------.
00000070 34 38 20 2D 3E 20 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 48 -> ---
00000080 2D 2D 2D 31 30 30 30 32 2D 2D 2D 2D 2D 2D 2D 2D ---10002-
00000090 2D 2D 2D 2D 2D 0D 0A 31 30 3A 30 31 3A 34 38 20 -----..10
000000A0 2D 3E 20 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D -> ------
000000B0 31 30 30 30 33 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 10003----
000000C0 2D 2D 0D 0A 31 30 3A 30 31 3A 34 38 20 2D 3E 20 --..10:01
000000D0 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 31 30 30 ---------
000000E0 30 34 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 0D 04-------
000000F0 0A 31 30 3A 30 31 3A 34 38 20 2D 3E 20 2D 2D 2D .10:01:48
00000100 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 31 30 30 30 35 2D ---------
00000110 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 0D 0A 31 30 ---------
00000120 3A 30 31 3A 34 38 20 2D 3E 20 2D 2D 2D 2D 2D 2D :01:48 ->
00000130 2D 2D 2D 2D 2D 2D 2D 31 30 30 30 36 2D 2D 2D 2D -------10
00000140 2D 2D 2D 2D 2D 2D 2D 2D 2D 0D 0A 31 30 3A 30 31 ---------
00000150 3A 34 38 20 2D 3E 20 2D 2D 2D 2D 2D 2D 2D 2D 2D :48 -> --
00000160 2D 2D 2D 2D 31 30 30 30 37 2D 2D 2D 2D 2D 2D 2D ----10007
00000170 2D 2D 2D 2D 2D 2D 0D 0A 31 30 3A 30 31 3A 34 38 ------..1
00000180 20 2D 3E 20 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D -> -----
00000190 2D 31 30 30 30 38 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D -10008---
000001A0 2D 2D 2D 0D 0A 31 30 3A 30 31 3A 34 38 20 2D 3E ---..10:0
Hex View
Text View
Close



2.3.9.6.6 Data: SOS Viewer



The "SOS viewer" window provides access to the System Object Storage in an NX32L device. Two different sets of objects can be selected using the radio buttons:

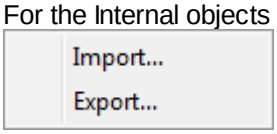
The **Internal** objects are used internally in the system and provides access to advanced settings, some of which are not directly available elsewhere. These objects can be modified but new objects can not be created.
The **User** objects are fully custom objects that can be used for storing application settings or resources. New objects can be created either from the SOS Viewer or by using the SOS functions.

The list shows all the objects that match the current filter.
Note that some objects, such as `gui.menu.password`, are encrypted, and should not be changed directly, instead export and import can be used to apply known settings from another device.

Fetch
This fetches the objects that matches the filter.

Filter
The filter textbox filters the objects based on the name. Pressing enter in the dialog or pressing "Fetch" fetches the matching objects.
It supports the wildcard characters * (zero or more characters) and ? (one character).
If no wildcard is specified, it will fetch all objects that contain the filter value.
If no filter is specified at all, all entries will be fetched.

Object functions
The object functions are used by right-clicking in the list.
For the Internal objects the following pop-up menu appears:



For the User objects the following pop-up menu appears:

Note: some of the functions require an object to be selected (right-click on the object).

The functions are described below:

Import...
Import... imports objects from an XML file into the SOS, either to copy the configuration from another device, or to be able to import a custom configuration.
When clicking Import..., a dialog is shown to select the file to import.
Once the file has been selected, the import dialog will be shown.

Export...
Export... exports the shown objects to an XML file, which can be used to copy settings from one device to other devices.
It can also be used to use other tools to edit the objects or to be able to choose between different configurations.

Create
This shows a dialog to create a new object.

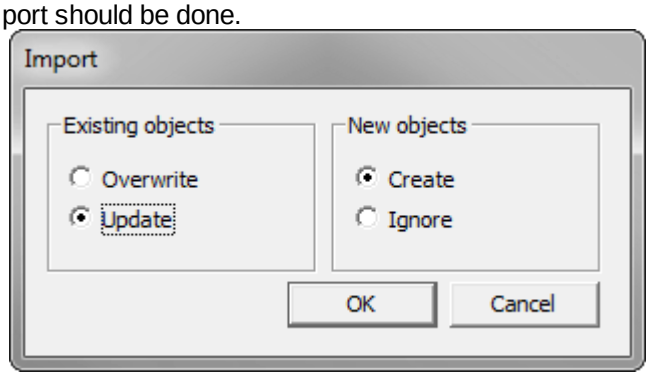
Remove
This deletes the selected object.

This is identical to the sosObjectDelete VPL function.
It is also possible to delete an object by using the "delete" key.

Remove All
This deletes all the objects that matches the current filter.

Import Dialog

The import dialog is shown once a file has been selected for import and is used to select how the import should be done.



Existing objects
Determines how any existing objects with the same names should be handled.

Overwrite
The old object is deleted and a new one is created with the value, description and limits from the file.

Update
If the existing object is of the same type, the value will be updated, if it is within the limits. If the existing object is of a different type or the value is not valid, the import fails.

New objects
Determines how objects that do not match an existing object should be handled:

Create
A new object is created for the object.

Ignore
The new object is not added to the SOS.

Depending on the purpose of the import, different selections should be used:

Purpose	Existing	New
- Clone the settings of a different device, without validation.	- Overwrite	- Create
- Import a few changed settings, with validation.	- Update	- Ignore
- Import changed and new settings, with validation.	- Update	- Create

Creating objects

The create object dialog is used to create new user objects.

Name
The unique name to use for the object. Valid characters are a-z, A-Z, 0-9 . and _.
If an object exists with the same name and type, the value of it will updated instead.
If an item with same name but a different type exists, an error message will be shown.

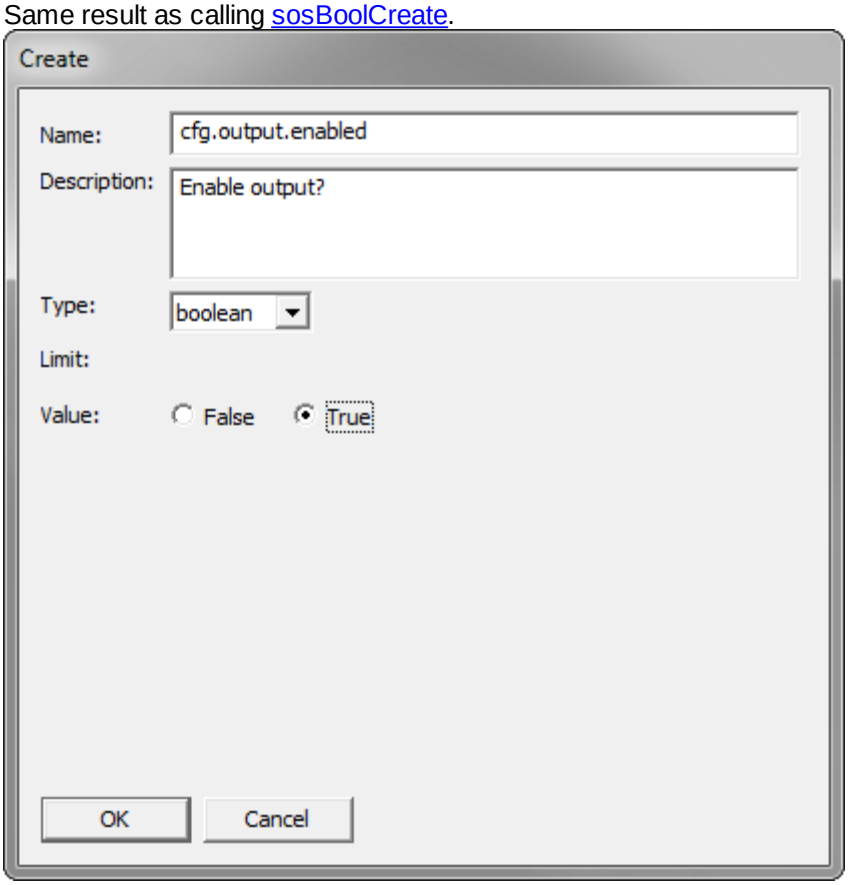
Description
The description to show when editing the value. Max 80 characters.

Type
The type of the object to create.

The appearance of the rest of the dialog changes depending on the chosen type.

Boolean

Same result as calling sosBoolCreate.

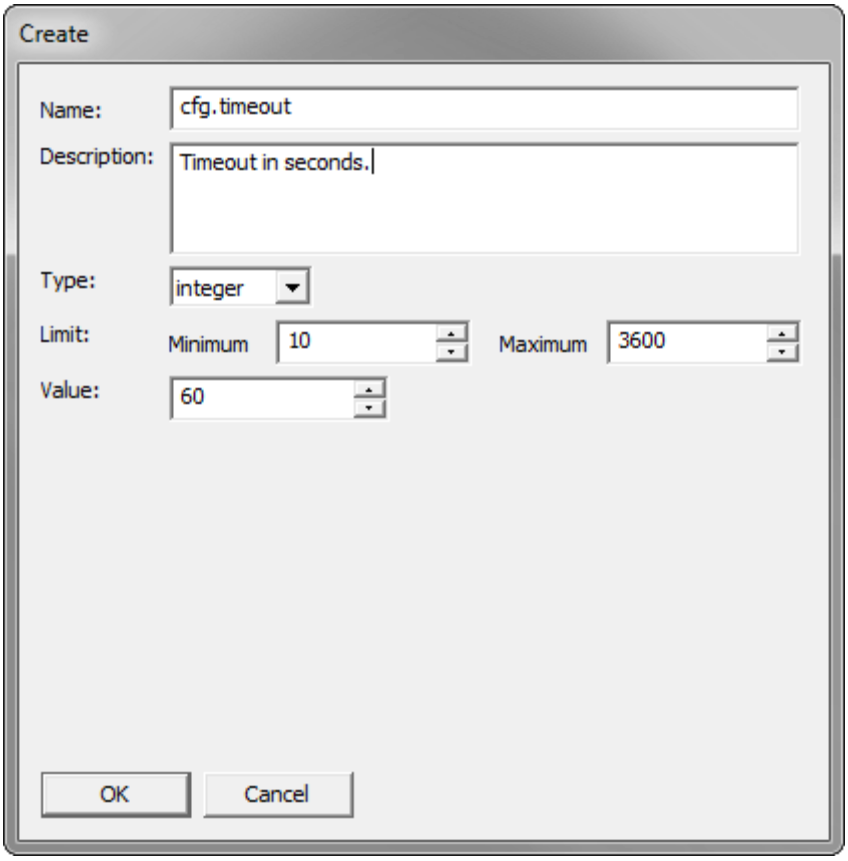


Value

The value to store.

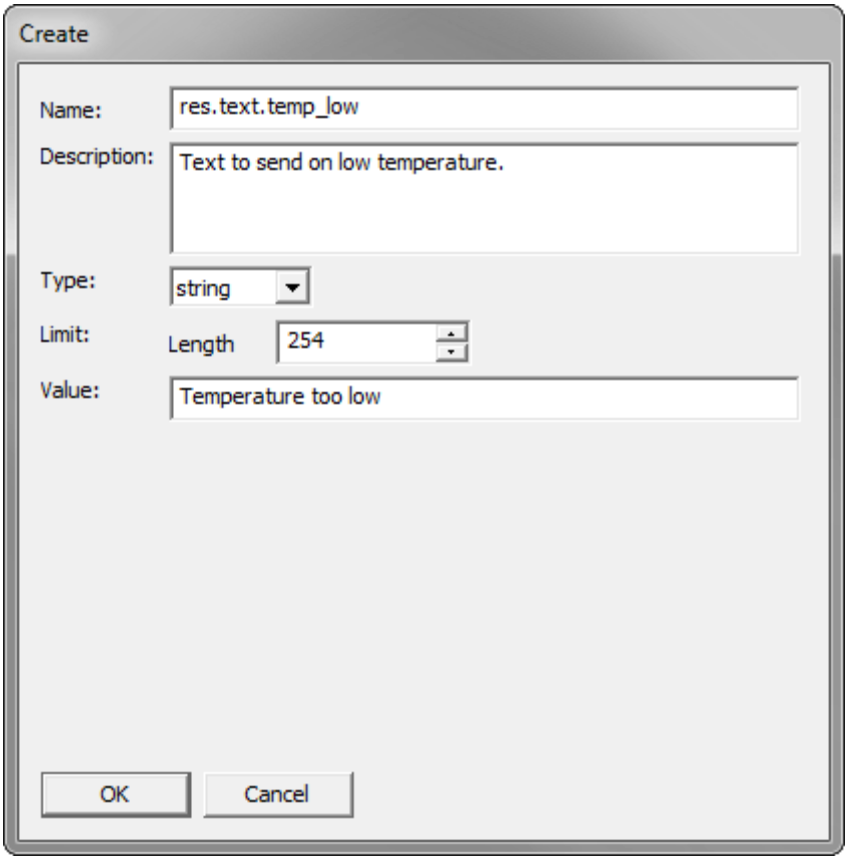
Integer

Same result as calling sosIntegerCreate.



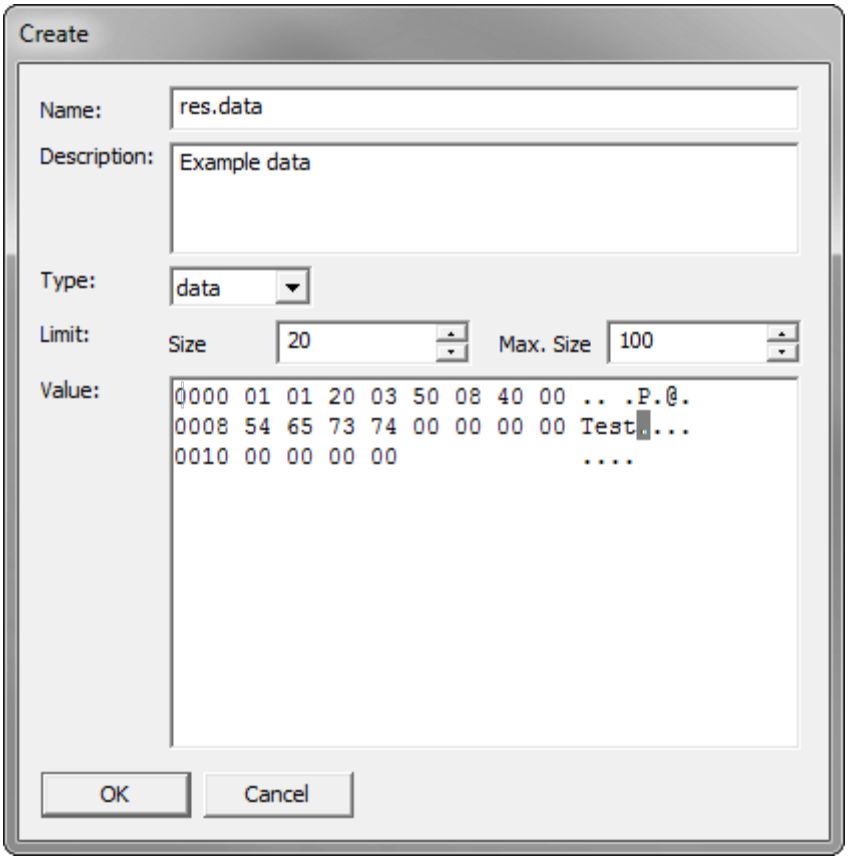
Minimum
The minimum allowed value.
Maximum
The maximum allowed value.
Value
The value to store.

String
Same result as calling sosStringCreate.



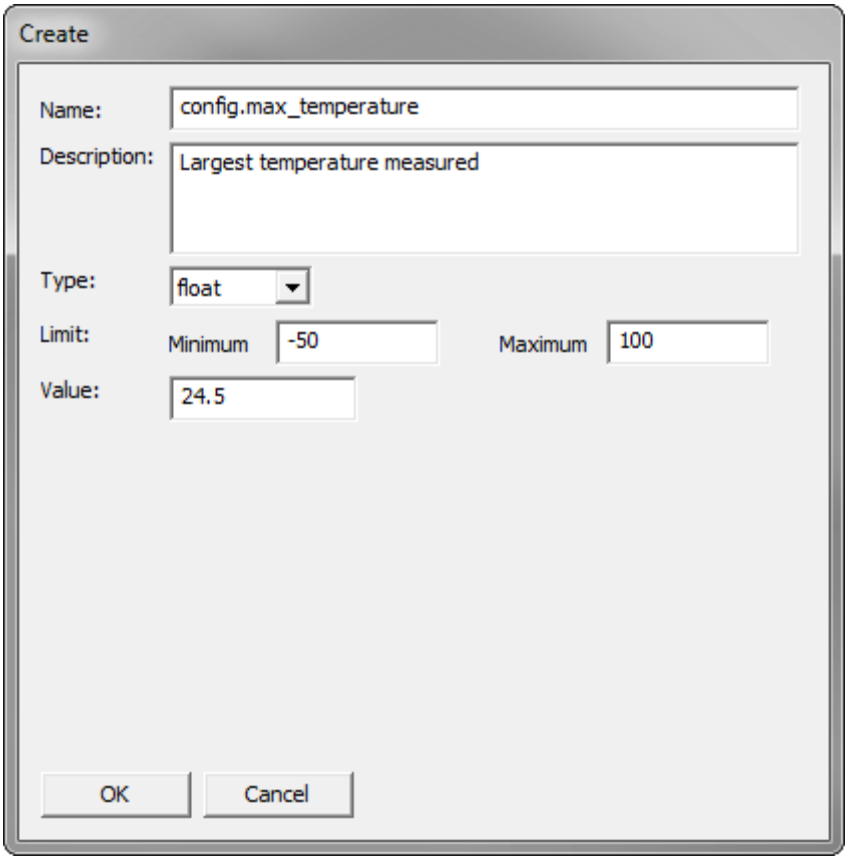
Length
The maximum allowed length of the string.
Value
The string to store.

Data
Same result as calling sosDataCreate.



Size
The amount of data to store.
Max. Size
The maximum allowed amount of data to store.
Value
The binary data to store.

Float
Same result as calling sosFloatCreate.

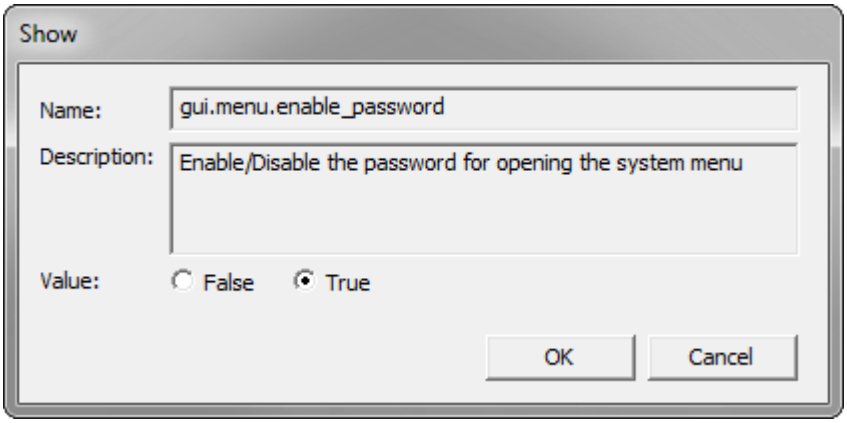


Minimum
The minimum allowed value.
Maximum
The maximum allowed value.
Value
The value to store.

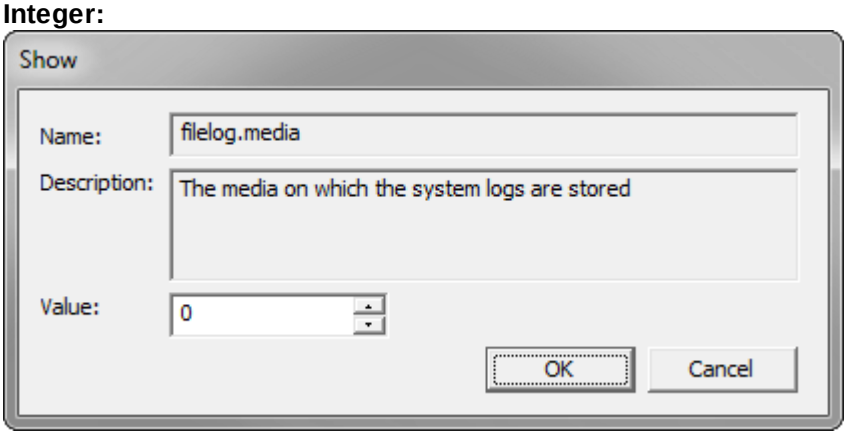
In addition to decimal values, the Minimum, Maximum and Value fields also supports values written in decimal exponent notation, e.g. "2.5e+6" for 2500000, as well as the special values "Inf" and "-Inf" or "Infinity" and "-Infinity" for the infinities and "NaN" for Not a Number. See FLOAT for more details about the special values.

Editing the value
Double clicking on an object shows a dialog to change the value of the object.
The valid range for the value depends on how the limits where configured when the object was created.

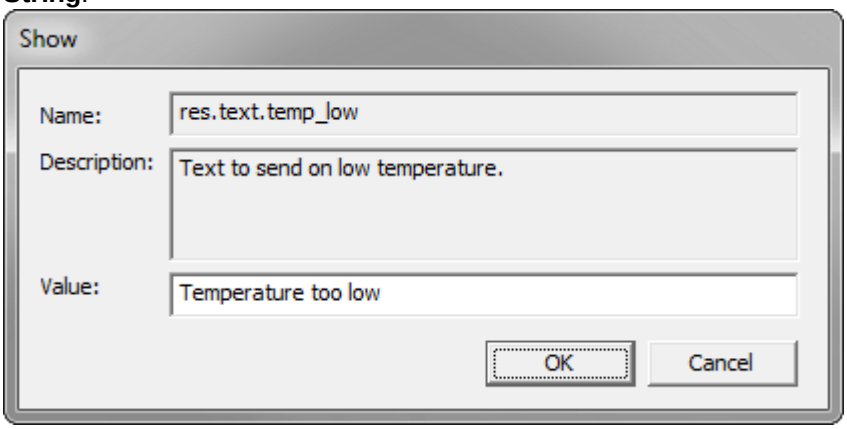
The appearance of the dialog depends on the type of data the object contains.
Boolean:



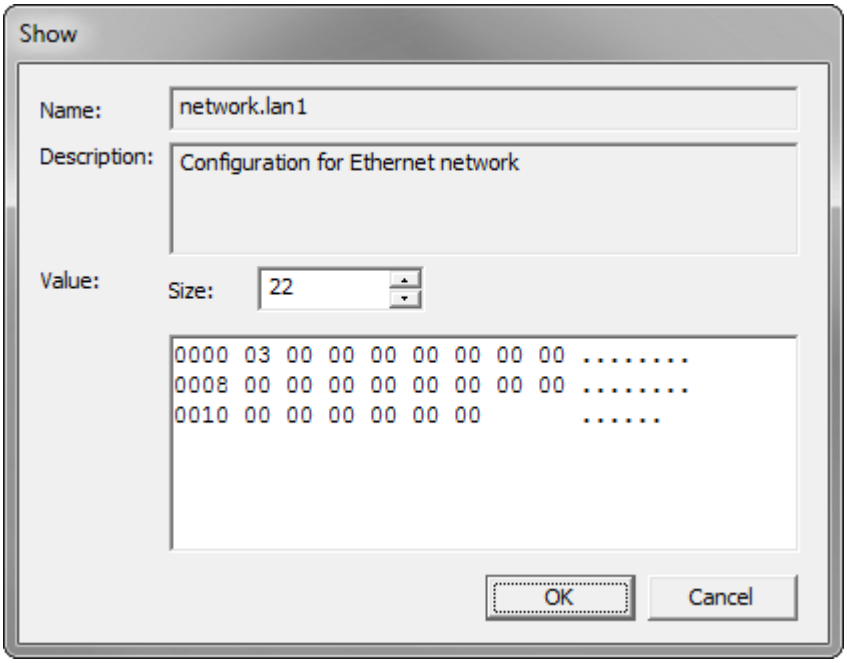
Integer:



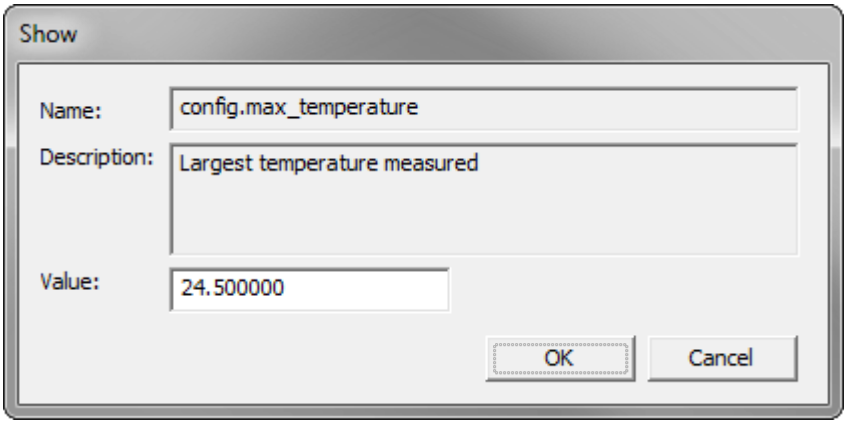
String:



Data / Encrypted String:
Note that binary Internal objects should not be changed, as it is possible that the content is encrypted.



Float:



2.3.9.7 Device: Execution

2.3.9.7.1 Device: Execution

The commands in the "Execution" menu allows you to manipulate the state of a connected RTCU device. Through this menu, it is possible to "Halt" the execution of a connected RTCU, as well as to "Reset" a device as seen below.

The individual items:

- Halt
- Reset

2.3.9.7.2 Execution: Halt

This stops the execution of VPL programs in a connected RTCU device. All outputs are set to their inactive state, and all program executions of VPL tasks/programs cease.

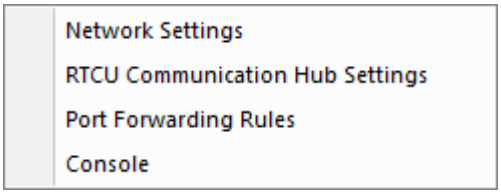
2.3.9.7.3 Execution: Reset

This command will reset the connected RTCU. This has the same effect as powering the device off and then subsequently on again. In case of a remote connection the reset will be delayed until the session is disconnected.

2.3.9.8 Device: Network

2.3.9.8.1 Device: Network

By using the following menus, it is possible to configure the network features which are found on some RTCU devices. Please consult the technical documentation for the actual RTCU device.



The individual items:

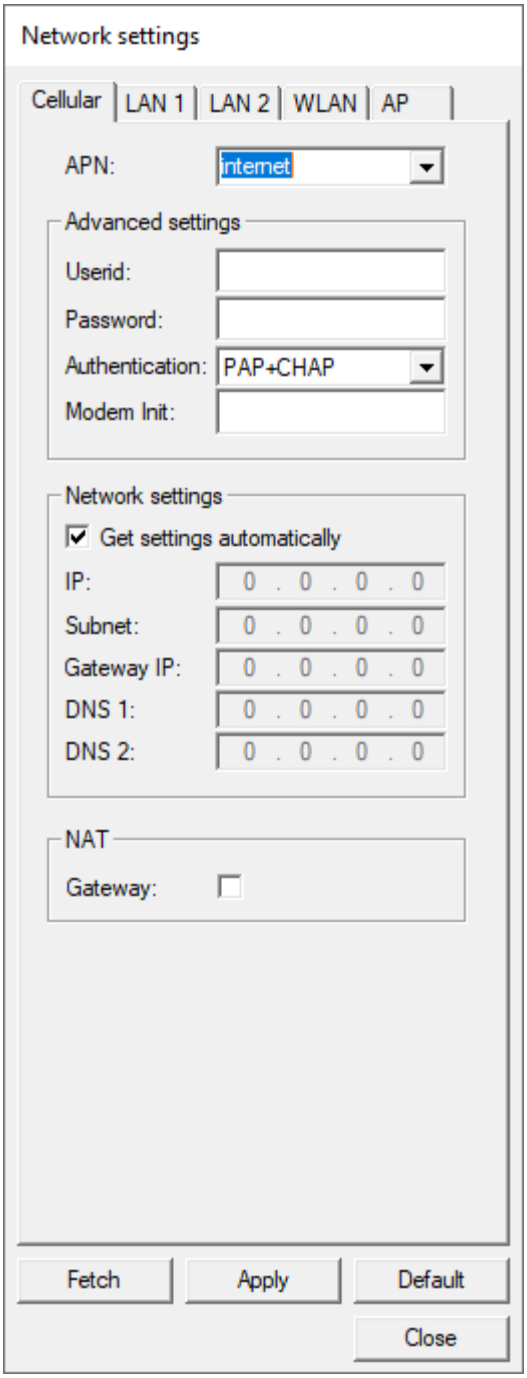
- Network Settings
- RTCU Communication Hub Settings
- Port Forwarding Rules
- Console

2.3.9.8.2 Network: Network Settings

This dialog allows one to manage the network configuration on the device. Please consult the technical documentation for the actual RTCU device regarding availability of TCP/IP.

The Cellular tab:

This tab allows one to set all relevant parameters for making a TCP/IP connection using the cellular module.
The parameters for the TCP/IP setup, must be supplied by the network operator that has supplied the SIM card.



Using the "Fetch" button it is possible to read the current configuration from the connected RTCU, and the "Apply" button will store the current configuration in the connected RTCU device.

The parameters are:

Userid:
The userid that must be used in order to establish the connection

Password:
The password that must be used in order to establish the connection

Authentication:
Select the correct authentication method (normally PAP+CHAP)

Modem init:
Type any other initialization parameters that needs to be sent to the GSM module to setup a mobile network connection. This field should normally be left blank.
When operating in a specialized GSM network environment with no DNS server it is possible to enter the parameter: `$DNS=0` (firmware V3.10 and later). This will disable the DNS look-up sanity check that is implemented in the firmware.
The default is `$DNS=1` that enables the DNS look-up check. Also refer to the netSetMonitorParm function for information on DNS lookup.

Get settings automatically
When this is selected the Network settings (IP, Subnet, etc.) are ignored.

IP:
The IP address of the RTCU (if 0.0.0.0 a dynamic address is used)

Subnet:
Subnet to be used (normally 0.0.0.0)

APN:
The APN name that is to be used for the connection
The dropdown list contains a history of APN's fetched from or written to RTCU devices.

DNS 1:
IP address of Domain Name Server 1 (if 0.0.0.0 this will be negotiated with provider)

DNS 2:
IP address of Domain Name Server 2 (if 0.0.0.0 this will be negotiated with provider)

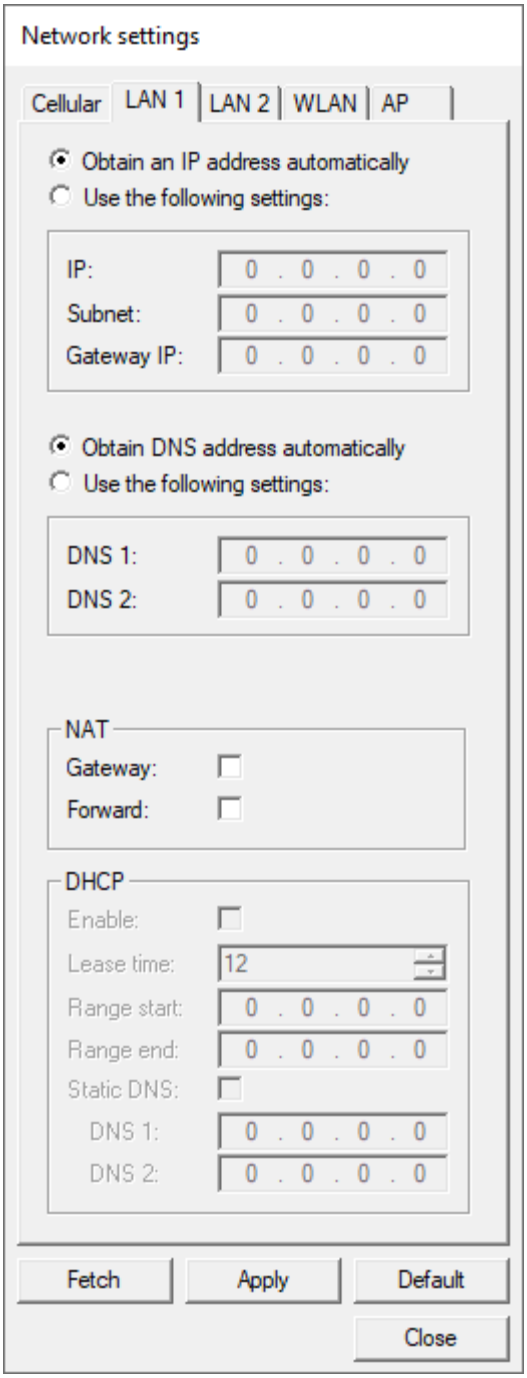
Gateway IP:
IP Address of the network gateway (normally 0.0.0.0)

NAT
This is where the NAT parameters, available for the interface, are configured. (See network for information)

> **Gateway**
> When this is selected the interface will act as a gateway to the internet.

The LAN tab(s):

This tab allows one to set all relevant parameters for making a TCP/IP connection using the LAN network interface.



If "Obtain an IP address automatically" is selected, it will use DHCP to determine the IP address, other wise it will use the static configuration with the following parameters:

IP:
The IP address of the RTCU

Subnet:
Subnet to be used

Gateway IP:
IP Address of the gateway

If "Obtain DNS address automatically" is selected, the DNS configuration provided from DHCP is used, otherwise the DNS servers with the following parameters are used:

DNS 1:
IP address of Domain Name Server 1

DNS 2:
IP address of Domain Name Server 2

NAT
This is where the NAT parameters, available for the interface, are configured. (See network for information)

Forward:
When this is selected socket connections will be forwarded to the Gateway interface.
If multiple local networks are present, socket connections will be forwarded to these as well.

Gateway:
When this is selected the interface will act as a gateway to the internet.

DHCP
This is where the DHCP server parameters are configured (See network for information)

Enable:
When this is selected, the DHCP server will listen for requests from the interface

Lease time:
The number of hours before a client must renew their address.

Range start:
The range of dynamic addresses starts with this address.

Range end:
The range of dynamic addresses ends with this address.

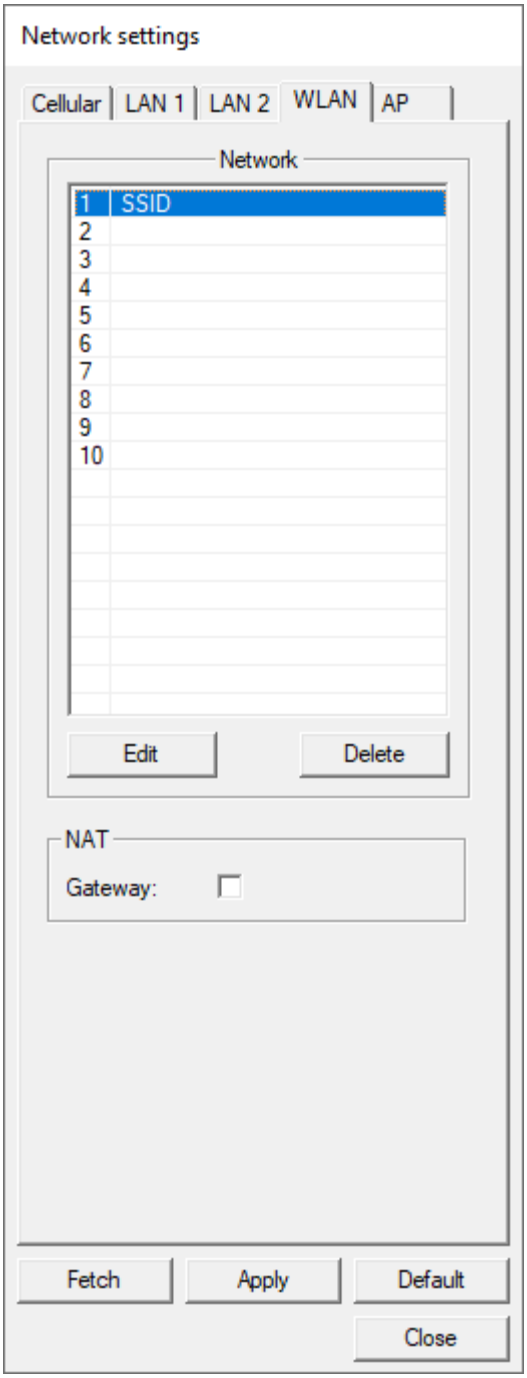
Static DNS:
When this is selected the DNS servers with the following parameters are used:

DNS 1:
The IP address of DNS server 1.

DNS 2:
The IP address of DNS server 2.

The WLAN tab:

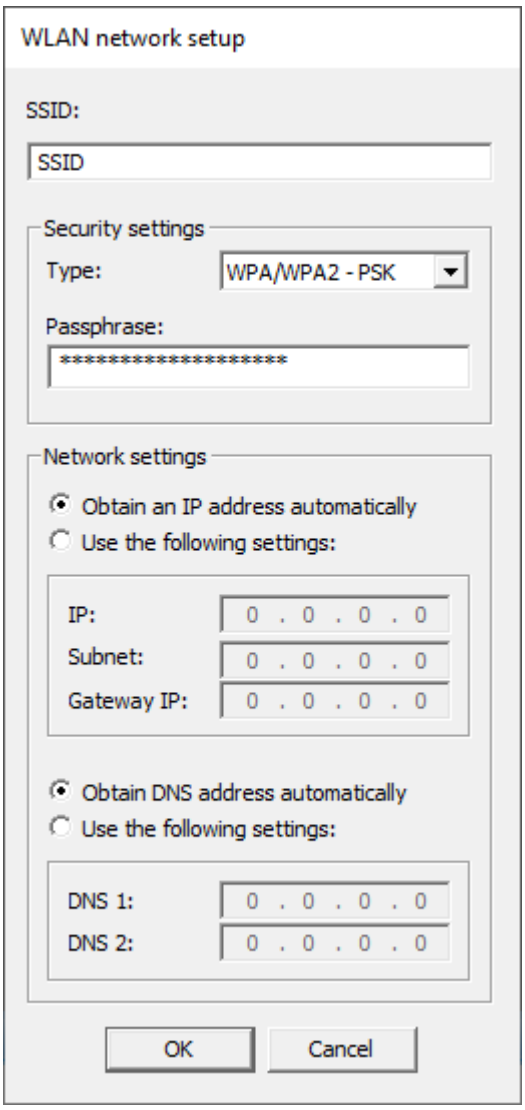
This tab allows one to manage the wireless networks that can be connected to using the WLAN network interface.



NAT
This is where the NAT parameters, available for the interface, are configured. (See network for information)

Gateway
When this is selected the interface will act as a gateway to the internet.

This dialog allows one to set all relevant parameters for making a TCP/IP connection using the WLAN network interface.



SSID:
The name of the wireless network

Security settings:
The parameters for the security of the network

If "Obtain an IP address automatically" is selected, it will use DHCP to determine the IP address, other wise it will use the static configuration with the following parameters:

IP:
The IP address of the RTCU

Subnet:
Subnet to be used

Gateway IP:
IP Address of the gateway

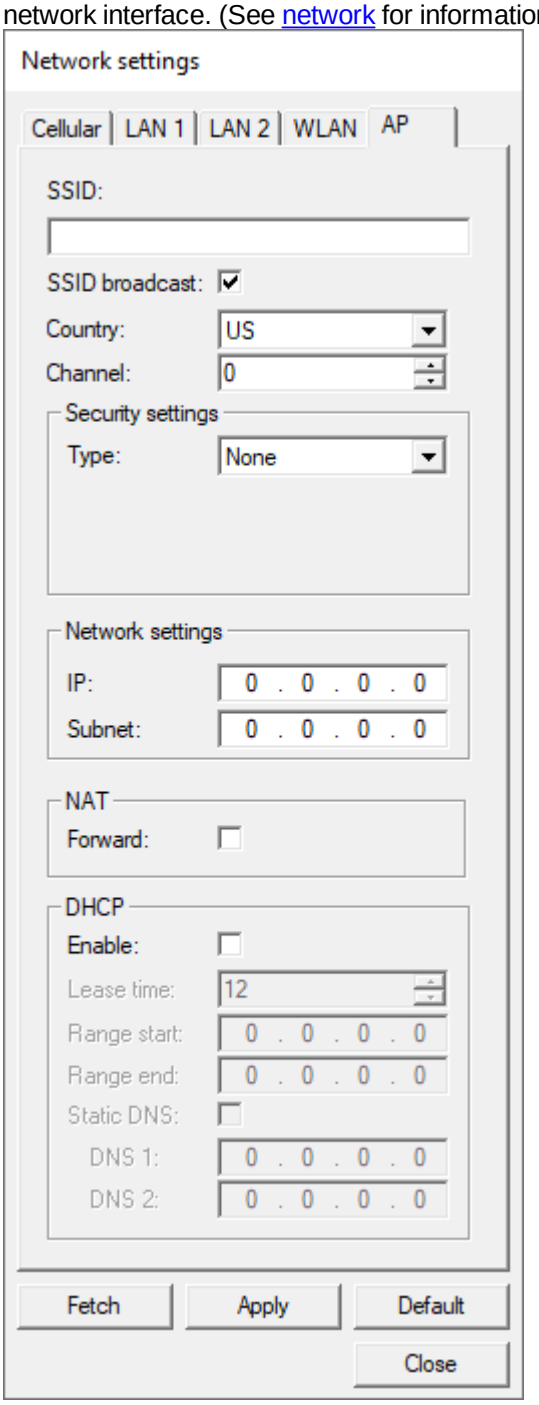
If "Obtain DNS address automatically" is selected, the DNS configuration provided from DHCP is used, otherwise the DNS servers with the following parameters are used:

DNS 1:
IP address of Domain Name Server 1

DNS 2:
IP address of Domain Name Server 2

The AP tab:

This tab allows one to set all relevant parameters for accepting TCP/IP connections using the AP network interface. (See network for information)



SSID:
The name of the wireless network.

SSID broadcast:
When this is selected, the SSID will be broadcast to all possible clients.

Country:
This is the country where the AP is located.
The ISO 3166-1 standard is used for the country code.

Channel:
The channel used to communicate.

Security settings:
The parameters for the security of the network.

IP:
The IP address of the RTCU

Subnet:
Subnet to be used

NAT
This is where the NAT parameters, available for the interface, are configured. (See network for information)

Forward:
When this is selected, socket connections will be forwarded to the Gateway interface.
If multiple local networks are present, socket connections will be forwarded to these as well.

DHCP
This is where the DHCP server parameters are configured (See network for information)

Enable:
When this is selected, the DHCP server will listen for requests from the interface

Lease time:
The number of hours before a client must renew their address.

Range start:
The range of dynamic addresses starts with this address.

Range end:
The range of dynamic addresses ends with this address.

Static DNS:
When this is selected the DNS servers with the following parameters are used:

DNS 1:
The IP address of DNS server 1.

DNS 2:
The IP address of DNS server 2.

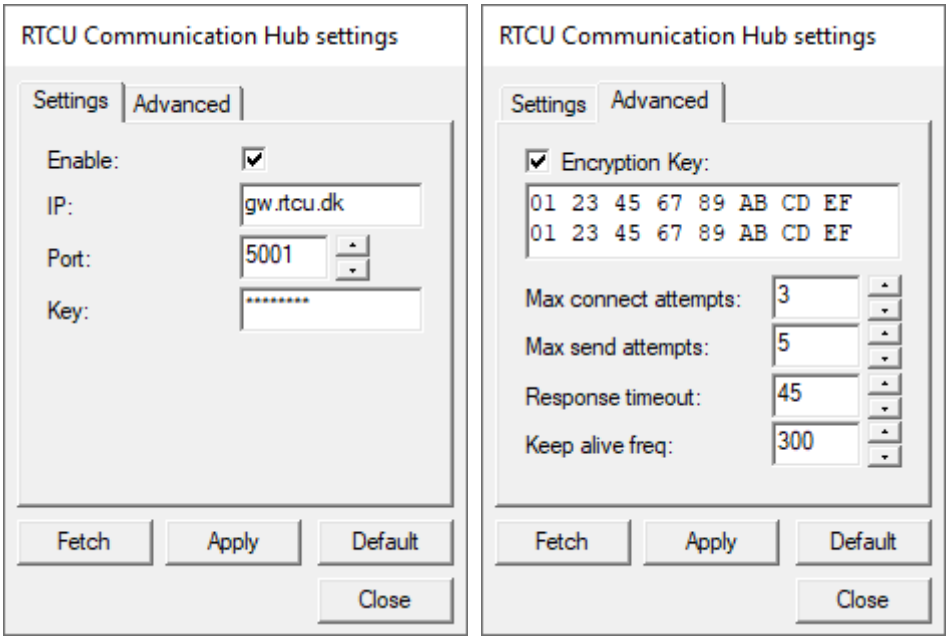
2.3.9.8.3 Network: RTCU Communication Hub Settings

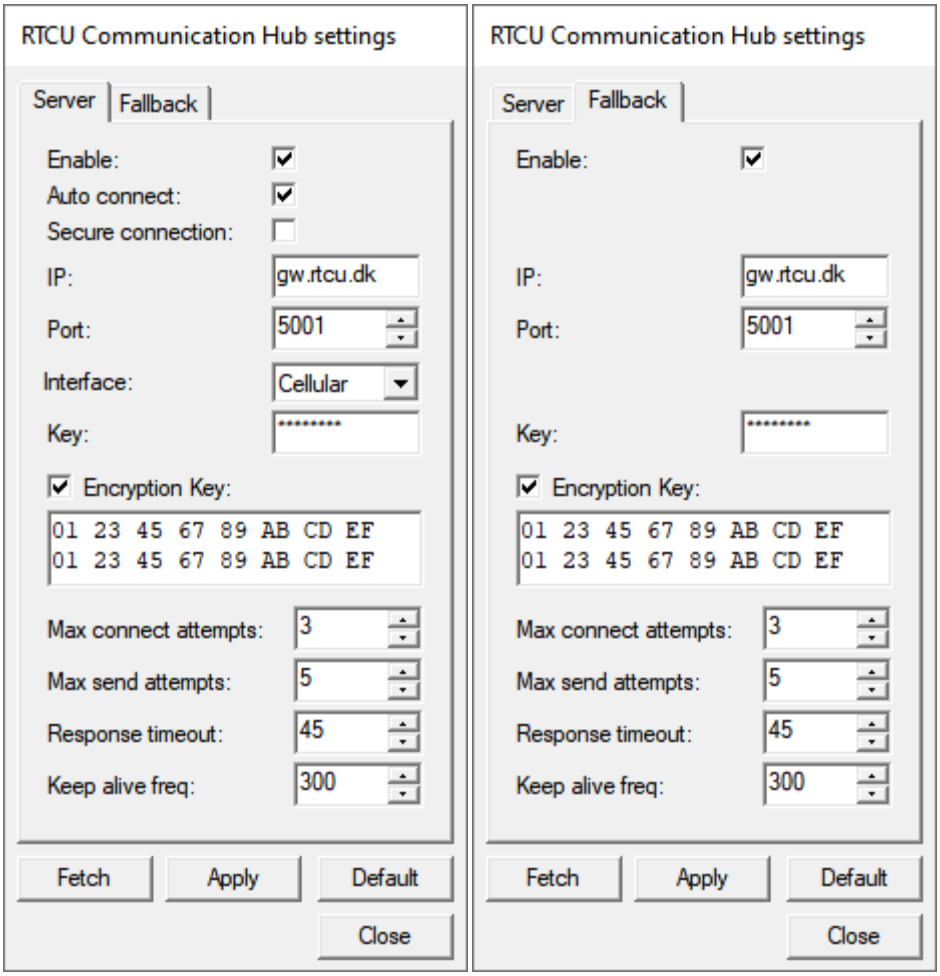
This dialog allows one to set all relevant parameters for making a connection to a RTCU Communication Hub using TCP/IP.
Please consult the technical documentation for the RTCU Communication Hub for further.

The Server and Fallback pages are displayed when NX32L architecture devices, running firmware 1.54.00 or later, are connected.
The Settings and Advanced pages, will be displayed for other devices.

The Fallback configuration is used in relation to secure connections to the RCH. It is designed to avoid connectivity loss to a device if for example the certificate has expired. In that case it will resort to the non-secure Fallback connection.
The Fallback configuration will also be used of the Server configuration is not enabled.

See gw: RTCU Communication Hub / Gateway for further details.





By using the "Fetch" button, it is possible to read the current configuration of the connected RTCU, and the "Apply" button will store the current configuration in the connected RTCU device. If "Default" is activated, a default set of parameters will be loaded into the entry fields of the dialog.

The parameters are:

Enable:
Control whether the configuration should be used.

Auto connect:
Control whether the device should connect to the server automatically when the network interface is available.

Secure connection:
Control whether a Standard connection or Secure connection is used to connect to the server.

IP:
The IP address (or symbolic name) of the RTCU Communication Hub.

Port:
The port number that is to be used for communicating with the RTCU Communication Hub (this is set in the RTCU Communication Hub).

Interface:
The network interface to use when connecting to the server. (see Network for available interfaces)

The fallback configuration will always use the cellular interface unless it is changed with gwSetMedia or rchInterfaceSet.
NX32L architecture devices in recovery and fault mode will try all the interfaces, starting with the currently selected interface.

Key:
The key value that is to be used when communicating with the RTCU Communication Hub (this is set in the RTCU Communication Hub).

Advanced parameters, no changes normally needed:

Encryption Key:
The 128 bit long key used for encryption of data sent to and received from the RTCU Communication Hub.
The encryption in the device can be enabled/disabled using the Encryption Key checkbox.
It is not possible to use the encryption option with secure connections.
Note: the encryption key cannot be read directly from the device and will replaced with all zeros if such is attempted with "Fetch from RTCU".

Maximum connect attempts:
This is the maximum number of connection attempts to the RTCU Communication Hub before the reconnecting the mobile network to the GSM network.

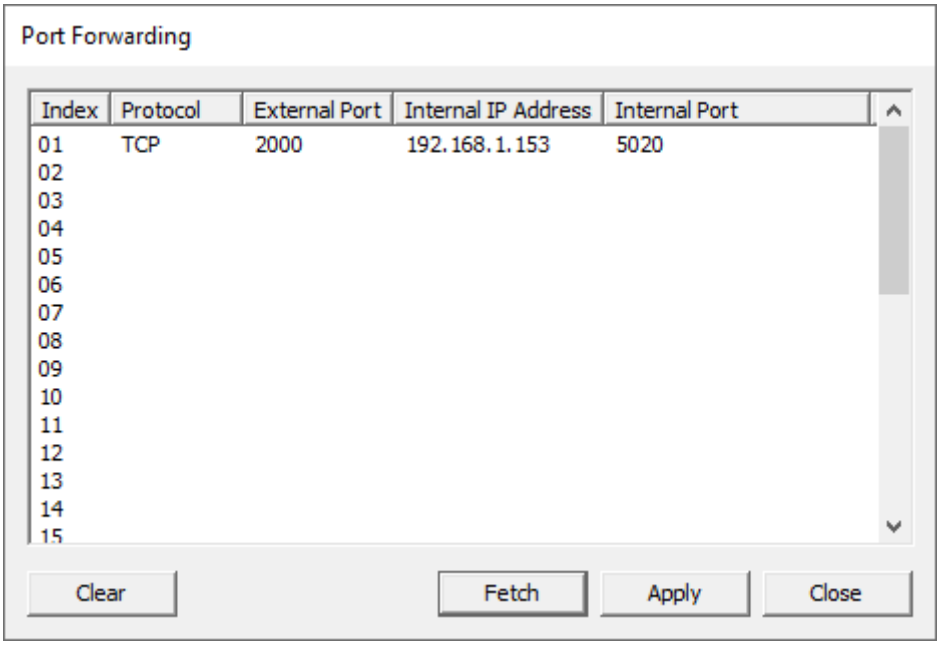
Maximum send attempts:
This is the maximum number of send request attempts before the send fails.

Response timeout:
This is the time spent waiting for a response, and it is specified in seconds.

Keep alive frequency:
Frequency for sending self-transactions through the RTCU Communication Hub (number of seconds between self-transactions).
The purpose of the self-transaction is to ensure a healthy two-way communication channel through the RTCU Communication Hub.
For applications that are only sending data from the device to the server, this frequency can safely be increased.
Setting the value to zero will disable the self-transactions completely.

2.3.9.8.4 Network: Port Forwarding

This dialog shows the port forwarding rules in an NX32L device.



By using the "Fetch" button, it is possible to read the current port forwarding rules of the connected RTCU, and the "Apply" button will store the current port forwarding rules in the connected RTCU device.
The 'Clear' button will remove all the rules from the dialog.

Double-click on a rule to edit the parameters.

The Port Forwarding rules can be modified from VPL with the netPortForwardSet function.

The parameters are:

Index
This is the rule number.

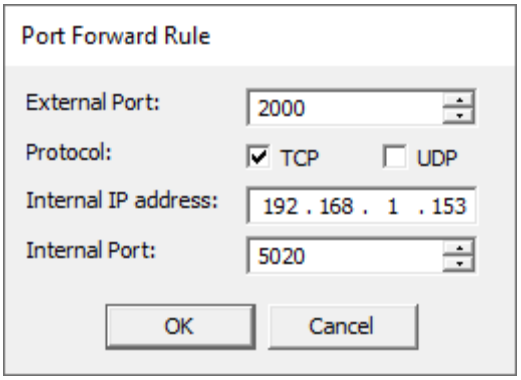
External Port
This is the port where incoming connections are forwarded from.

Protocol:
This is the protocols which are affected by the rule.

Internal IP Address:
This is the IP address where communication is forwarded to.

Internal Port:
This is the port where communication is forwarded to..

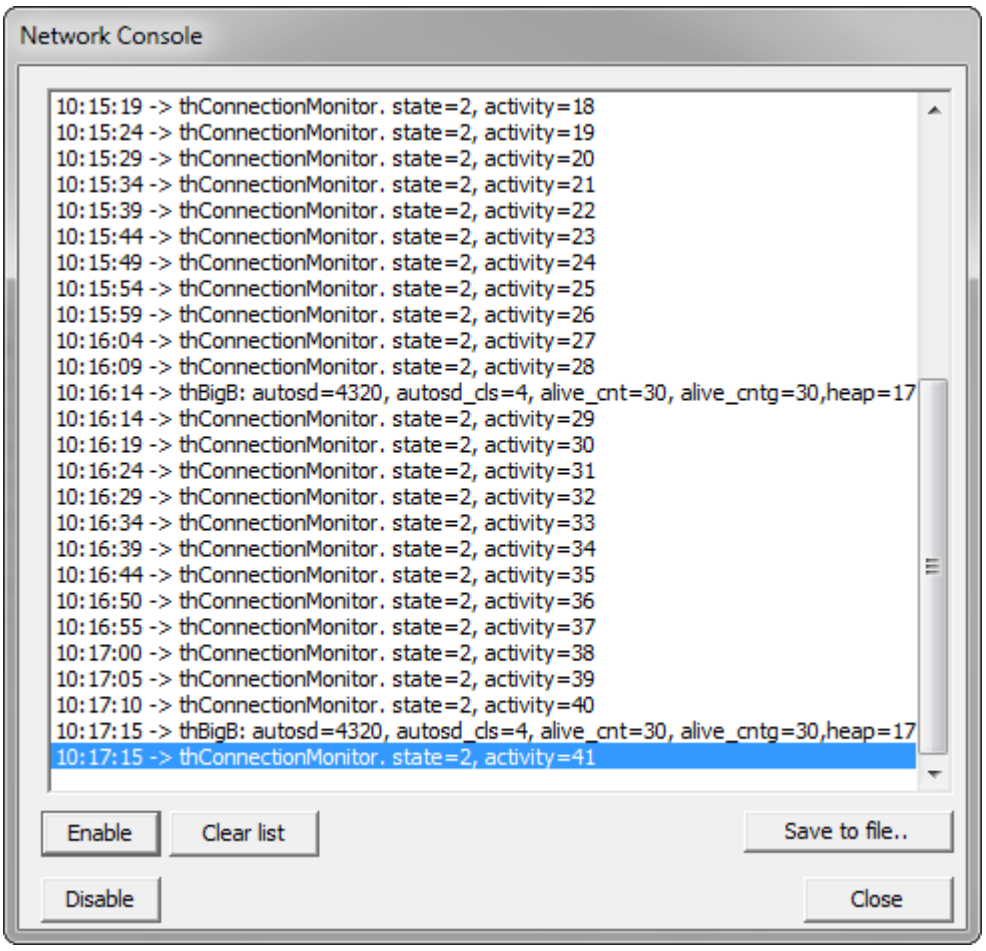
Port forward rule edit dialog:



This dialog is shown when editing a rule.
The parameters are the same as the Port forward dialog.

2.3.9.8.5 Network: Console

The "Network Console" opens up a "window" into the inner workings of the TCP/IP stack, as well as the network console on the RTCU. The Network Console is an advanced tool that will usually be operated under the guidance of Logic IO.
The console is only available during a direct cable connection to the device.



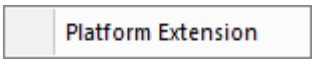
By default, the network messages will be disabled and must be enabled by pressing the "Enable" button. The "Clear list" button will clear the list box.
"Save to file" will open a dialog that allows you to select a file, onto which the complete log shown in the list box will be saved.

The actual content and interpretation of the log messages are not to be discussed further. Please contact your supplier for more information.

2.3.9.9 Device: Extension

2.3.9.9.1 Device: Extension

Platform extensions are supported under the NX32L execution architecture and allows the user to develop and install Programs and Modules.
Please refer to the Platform Extension functions API section for further details.

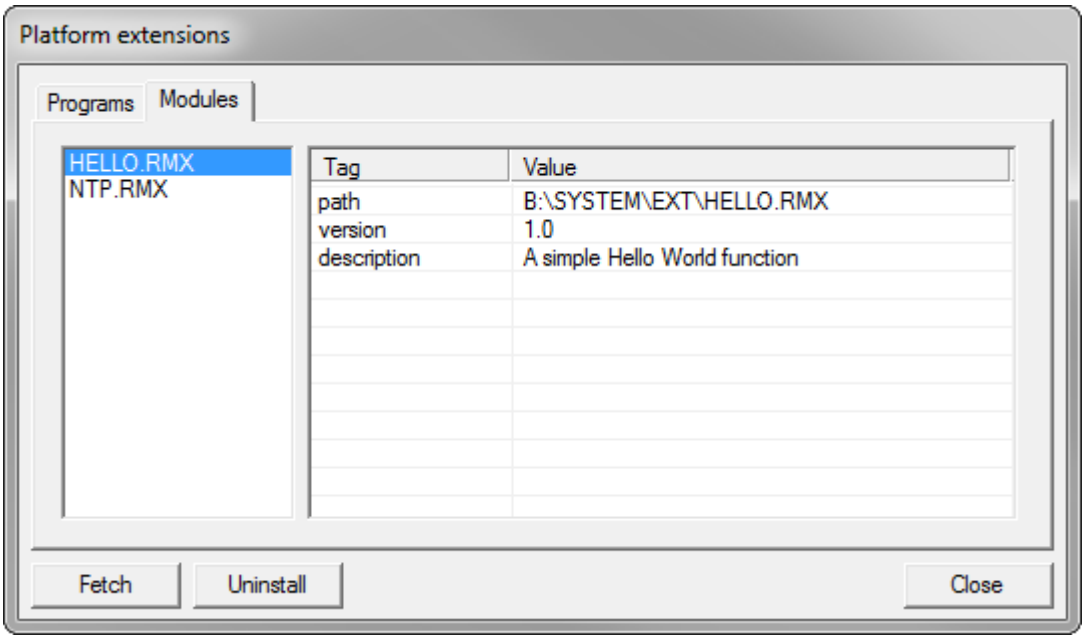


The individual items:

- Platform Extension

2.3.9.9.2 Extension: Platform Extension

This dialog allows one to manage the platform extensions on device.



The Programs tab lists the RTCU program extensions that are currently installed in the device. Similarly the Module tab lists the RTCU module extensions that are currently installed in the device.
Both programs and modules are managed by the application through the extension API.

The program and modules are listed to the left, and a list of property tags for the selected program is listed on the right.

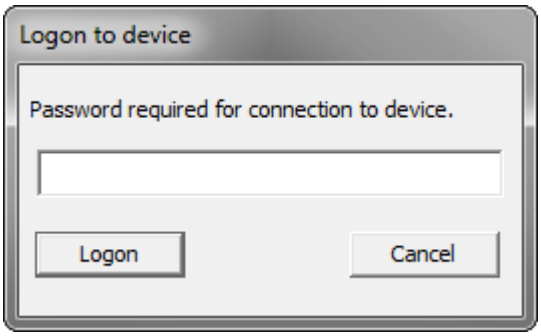
For more information see the Extension overview.

Platform Extensions are installed on a device as part of the project transfer.
See project view for information on how to add platform extensions.

Fetch
This fetches the information about all the programs and modules which is installed on the device.

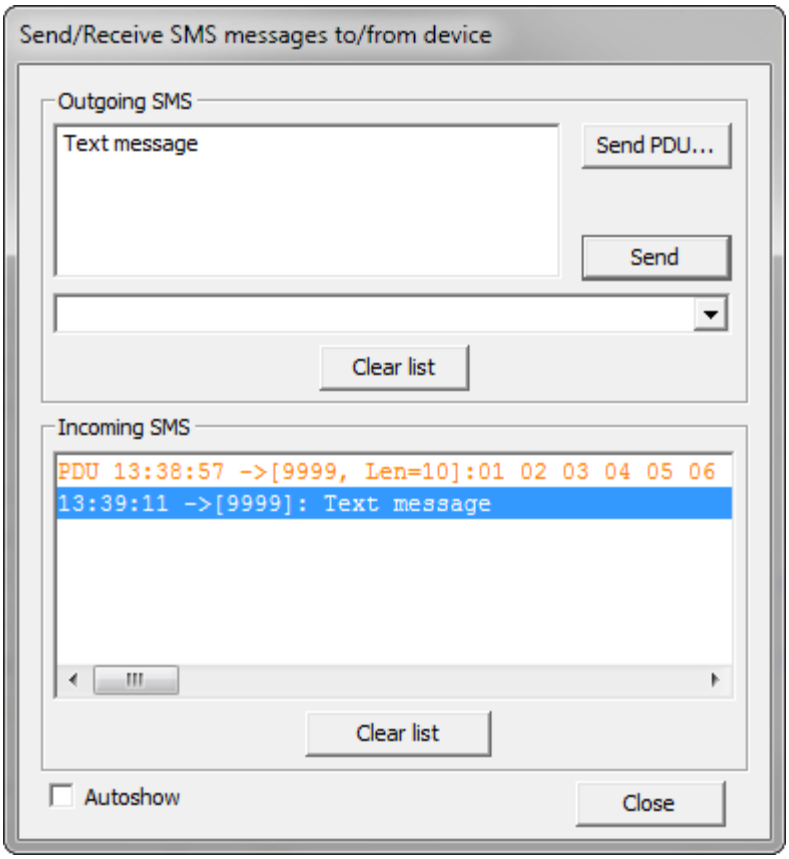
Uninstall
This will uninstall the selected program or module from the device.
Uninstalling a program will not stop it if it is executing.

2.3.9.10 Device: Logon



This function allows you to issue a logon to a connected RTCU device. Only after a successful logon (or if the password field of the connected RTCU device is empty) is it possible to interact with the RTCU device. It is possible through the "Set password" function, which allows you to change the password.

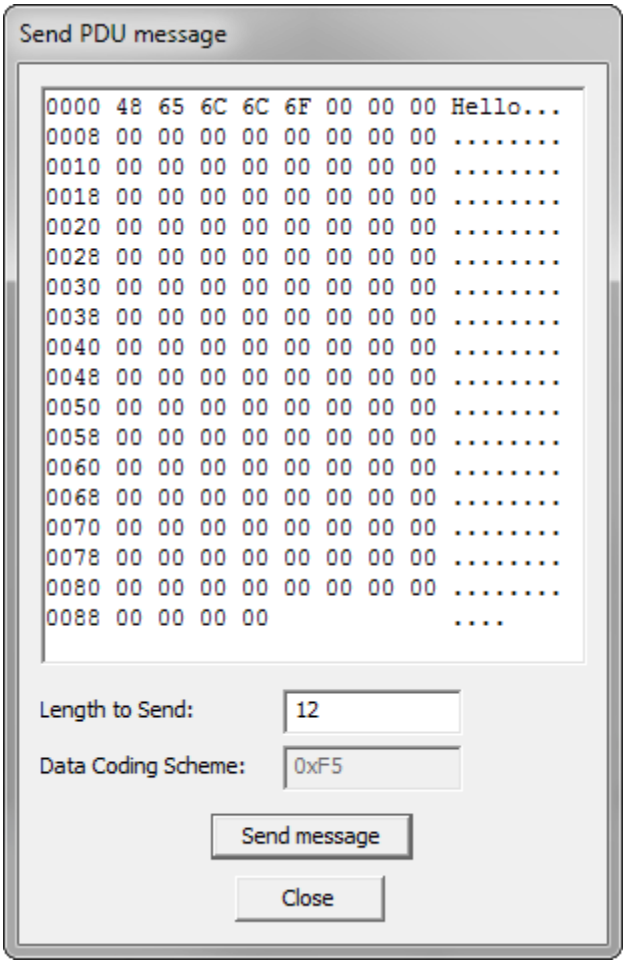
2.3.9.11 Device: SMS Messages



Through this dialog, it is possible to send and receive SMS messages to/from a connected RTCU device. All messages to/from this window will be using the phone number "9999". If an SMS message is sent from within a VPL program using the gsmSendSMS() or the gsmSendPDU() functions, it will be received in the lower window called "Incoming SMS". Messages sent from the upper window will be received in the VPL program by either the function blocks gsmIncomingSMS or gsmIncomingPDU - depending on if "Send" or "Send PDU" is pressed.

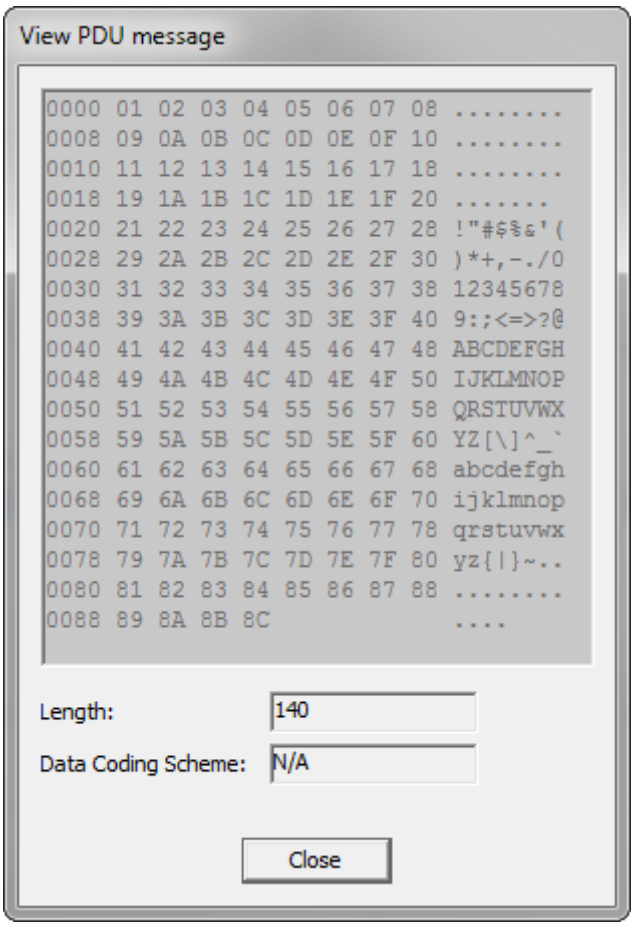
When the "Autoshow" option is enabled, the window will automatically be shown when an SMS from a connected device is received.

If the "Send PDU" button is pressed, the following dialog appears:



Through this dialog, it is possible to send binary messages to a connected RTCU.

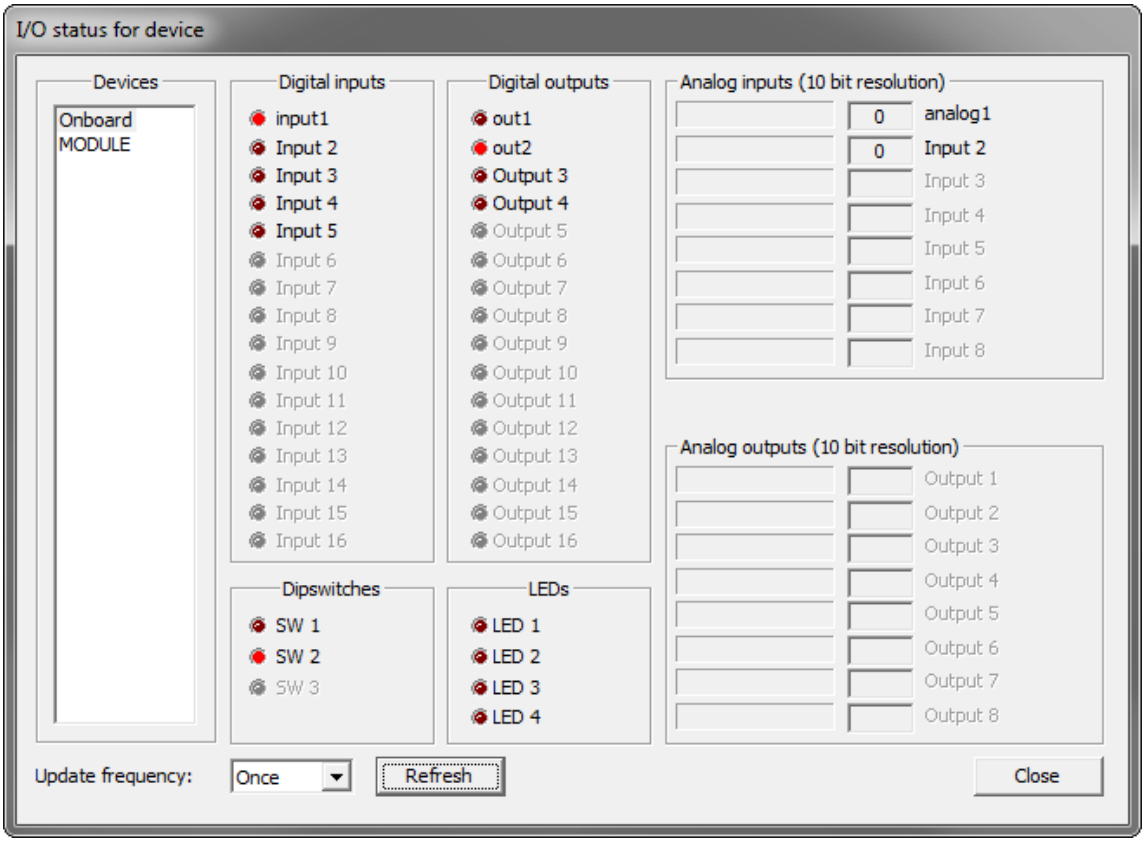
When a PDU message is shown, it is possible to double-click it, and this should invoke the following dialog:



2.3.9.12 Device: I/O Monitor

The "I/O Monitor" function allows for examining the physical I/O's of any connected RTCU device. Using the drop-down box makes it possible to select the update method. Should "once" be selected, the I/O status will be update every time the "Refresh" button is pressed.
If the mouse hovers over a label for an I/O signal, the comment for that VAR_INPUT/VAR_OUTPUT variable will be shown in a bubble.

Please note that the I/O monitoring takes some seconds to complete - especially if the connection to the RTCU is through a remote (modem) connection.



The "Devices" group lists the available devices with I/O.
The Onboard device is the I/O present on the RTCU device; the others, if present, are I/O Extension devices.

When the RTCU IDE connects with the RTCU device, the I/O Monitor will query the RTCU device for the number of I/O available.
If the number of I/O signals from the I/O Extension devices is
* a match with the current project:
The I/O monitor will use the I/O Extension devices from the project to determine the number of devices, and the number of I/O on each device.

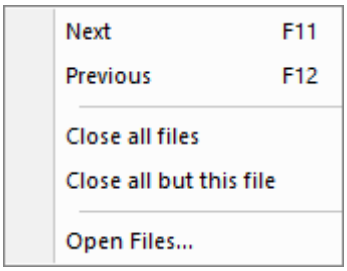
* not a match with the current project:
The I/O monitor will use a 'super' device, to determine the number of devices, and the number of I/O on each device.

The super device are defined to have 16 Digital inputs, 16 Digital outputs, 8 Analog inputs and 8 Analog outputs.

2.3.10 Menu item: Window

2.3.10.1 Menu item: Window

By using the "Window" menu items, it is possible to manipulate the various windows found on one's screen.



The individual items:

- Next
- Previous
- Close all files
- Close all but this file
- Open Files...

2.3.10.2 Window - Next

This will activate and show the next open file.

2.3.10.3 Window - Previous

This will activate and show the previous open file.

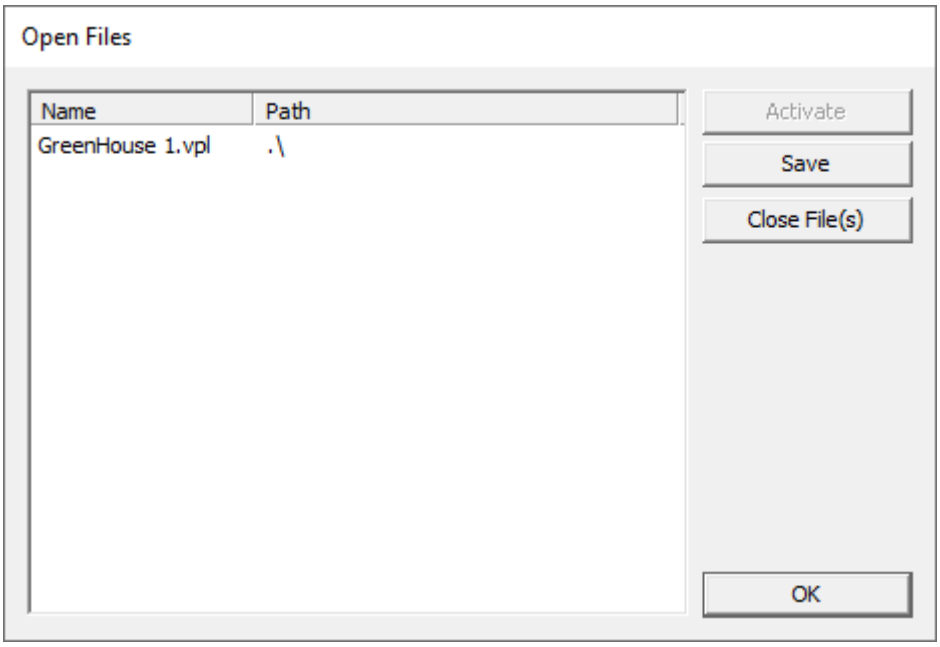
2.3.10.4 Window - Close all files

This will close all open files.

2.3.10.5 Window - Close all but this file

This will close all open files, except the active file.

2.3.10.6 Window - Open Files



The Open Files dialog shows a list of all the files that are open in the editor.

The following actions can be performed on the list:

Activate
This will activate and show the selected file.
Only the first file is activated, if multiple files are selected

Save
This will save the contents of the selected files

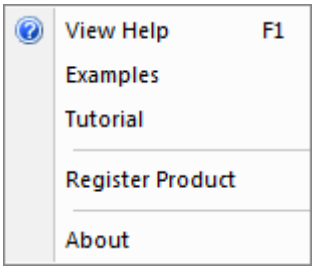
Close File(s)
This will close the selected files.
A pop-up message will ask to save the contents of dirty files.

2.3.11 Menu item: Help

2.3.11.1 Menu item: Help

By using the "Help" menu, it is possible to receive help on specific items, register the RTCU IDE program, and to access the Online Examples and Tutorial.

When one has an active file, such as a VPL program file, it is possible to position the cursor on a word in the program and then press Alt-F1. This will bring up the online help for that specific word.



The individual items:

- View Help
- Examples
- Tutorial
- Register Product
- About

2.3.11.2 Help - View Help

This command will start the Windows Help system. One will be presented with the contents of the RTCU IDE online help manual.

2.3.11.3 Help - Examples

This will give access to the Examples section of the online help.

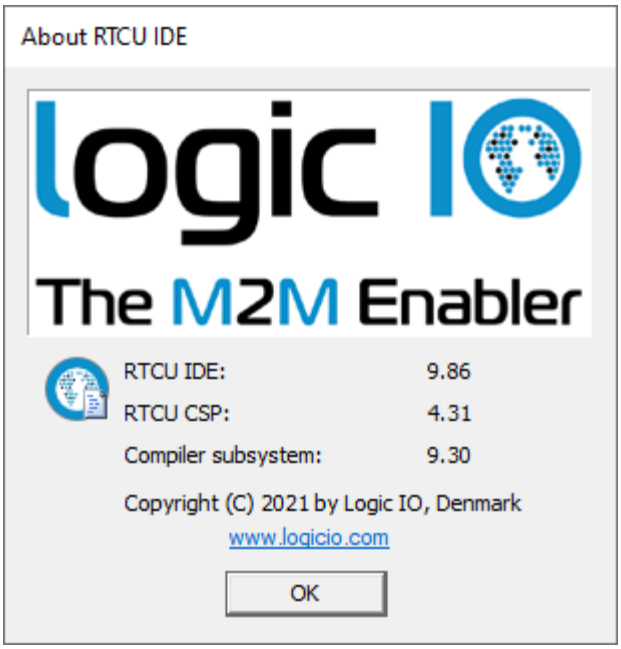
2.3.11.4 Help - Tutorial

This will give access to the Tutorial section of the online help.

2.3.11.5 Help - Register Product

This command allows one to register the RTCU IDE product. This will enable us to send email notifications when new versions, features etc. become available for the product.

2.3.11.6 Help - About

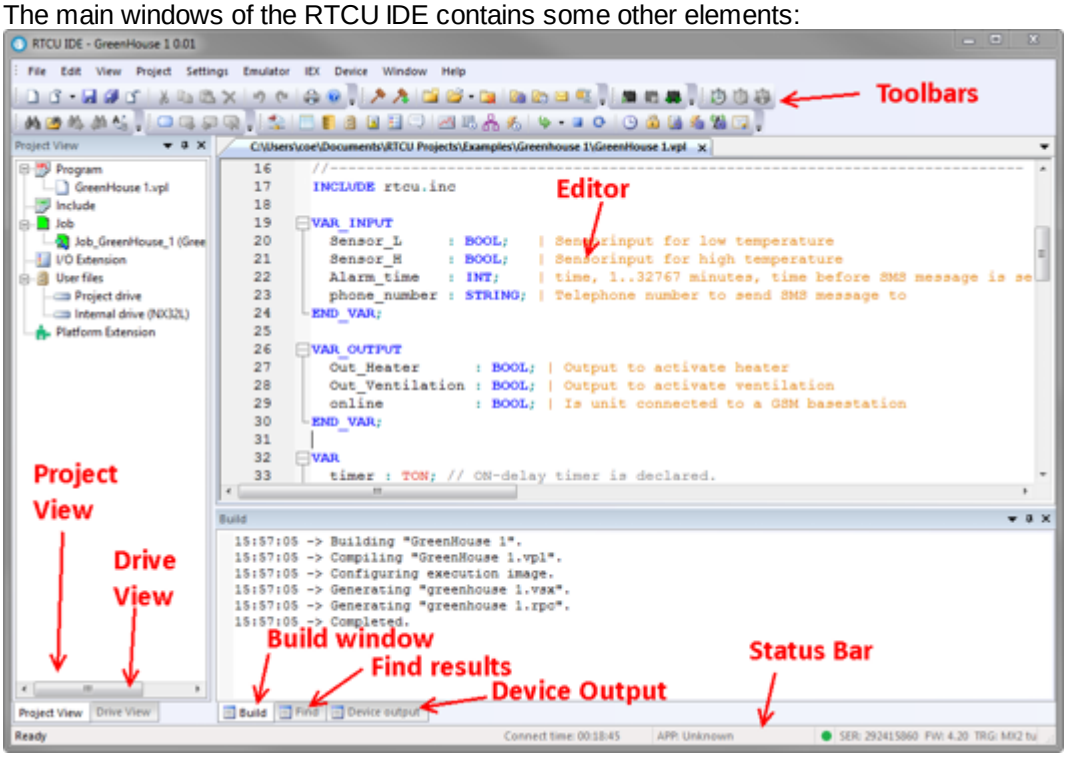


This command shows the current version number of the RTCU IDE program and a copyright notice.

2.4 Windows

2.4.1 Windows

The main windows of the RTCU IDE contains some other elements:

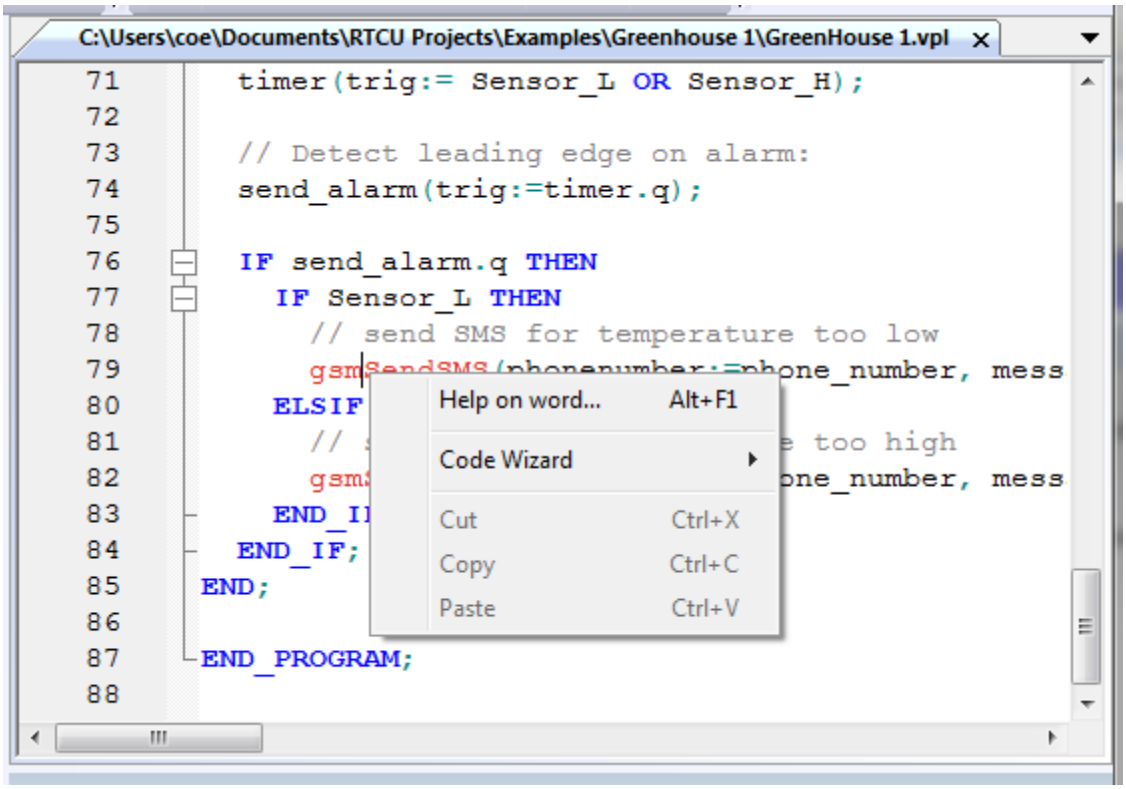


The elements(except the status bar) can be dragged around to other locations if wanted.

- Toolbars
- Status bar

- Editor Window
- Project View
- Drive View
- Build Window
- Find Results
- Device Output

2.4.2 Editor Window

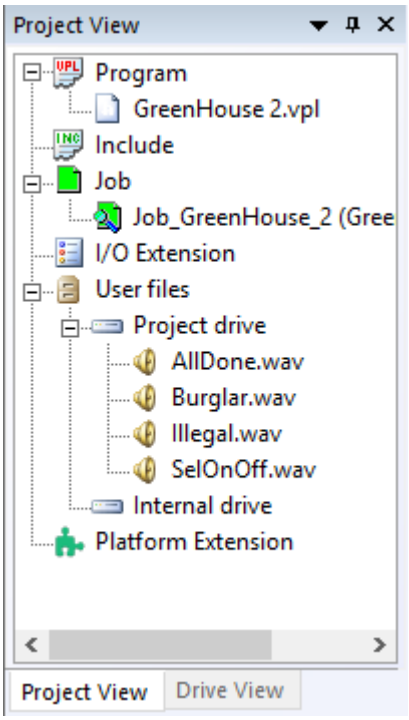


The editor provides syntax highlighting for VPL code and has a right click menu with the following options:

- **Help on word...**
  - Opens the Online Help and attempts to find the documentation for the word at the location of the cursor.
- **Code wizard**
  - Provides small code snipplets for all the available functions.
- Cut
- Copy
- Paste

2.4.3 Project View

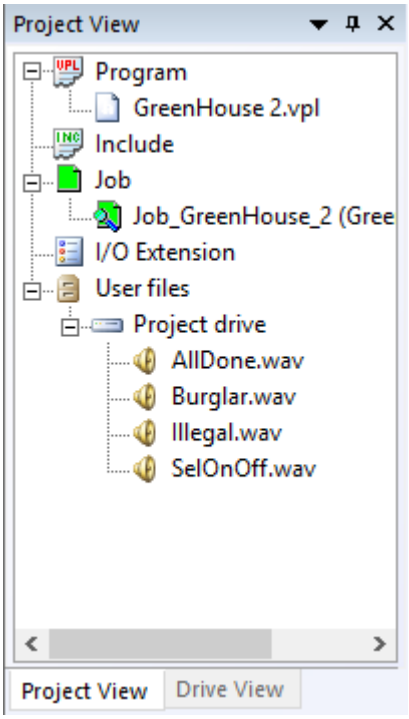
2.4.3.1 Project View



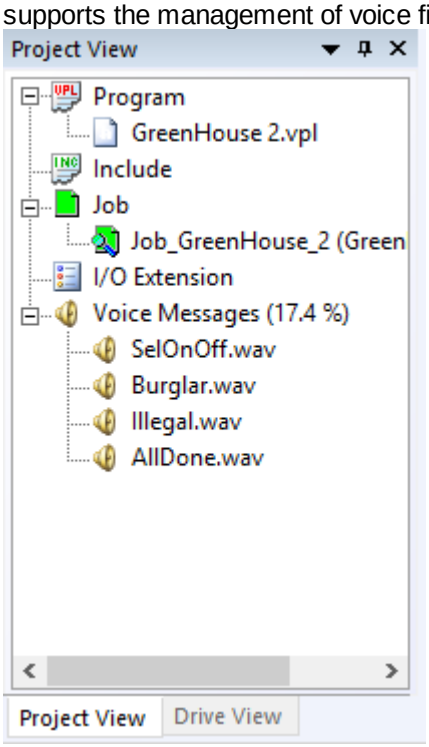
The Project View contains the project tree. Using this, it is possible to manage Program- and Include files, as well as Jobs, I/O extensions, Platform Extensions and User files, including Voice messages.
When the mouse pointer is directed to one of the items in the project tree and subsequently right-clicked, a drop-down menu will appear with a number of possible commands that can be activated for the specified item. Please look on the following pages for an explanation of the individual commands:

- Program
- Include
- Job
- I/O Extension
- User Files
  - o Project drive
  - o Internal drive
- Platform Extension

When a NX32 project is loaded, the Internal drive and Platform extension is removed.



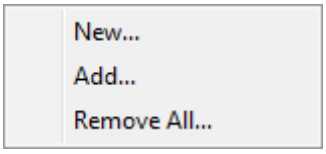
When an X32 project is loaded, the User files item is changed to Voice Messages and only supports the management of voice files:



2.4.3.2 Project View: Program

2.4.3.2.1 Project View: Program

Right-clicking on the "Program" text in the project tree will make the following drop-down menu appear:

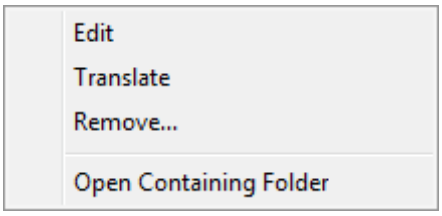


The individual items:

- New
- Add

"Remove All" will remove all Program files from the project. Please note that while the files are not actually deleted from the disk, they are no longer referenced and included in the current project.

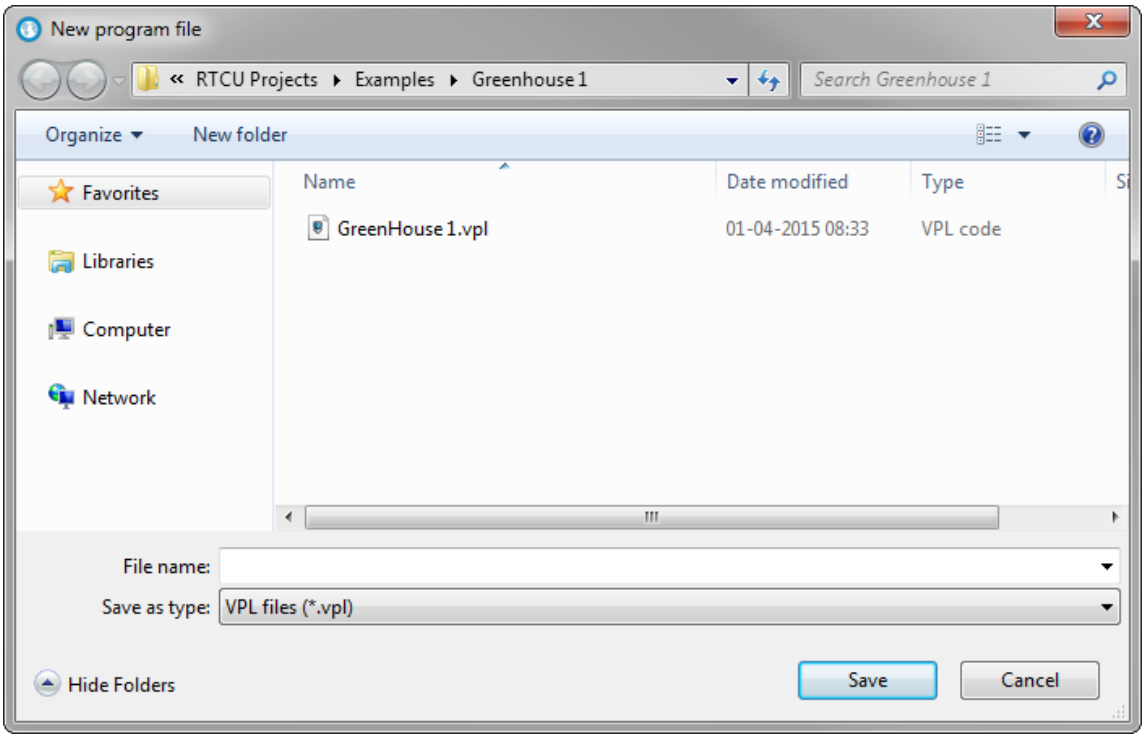
Right-clicking on a specific file in the "Program" section of the project tree will make the following drop-down menu appear:



The "Edit" command will open the file in the editor, so that you are able to make modifications to it, save it, print it, and so on.
"Translate" will translate the Program file and report any errors that are found.
"Remove" will remove the file from the project (the file is not deleted - just removed from the project).
"Open containing folder" will open the folder containing the file in Windows Explorer.

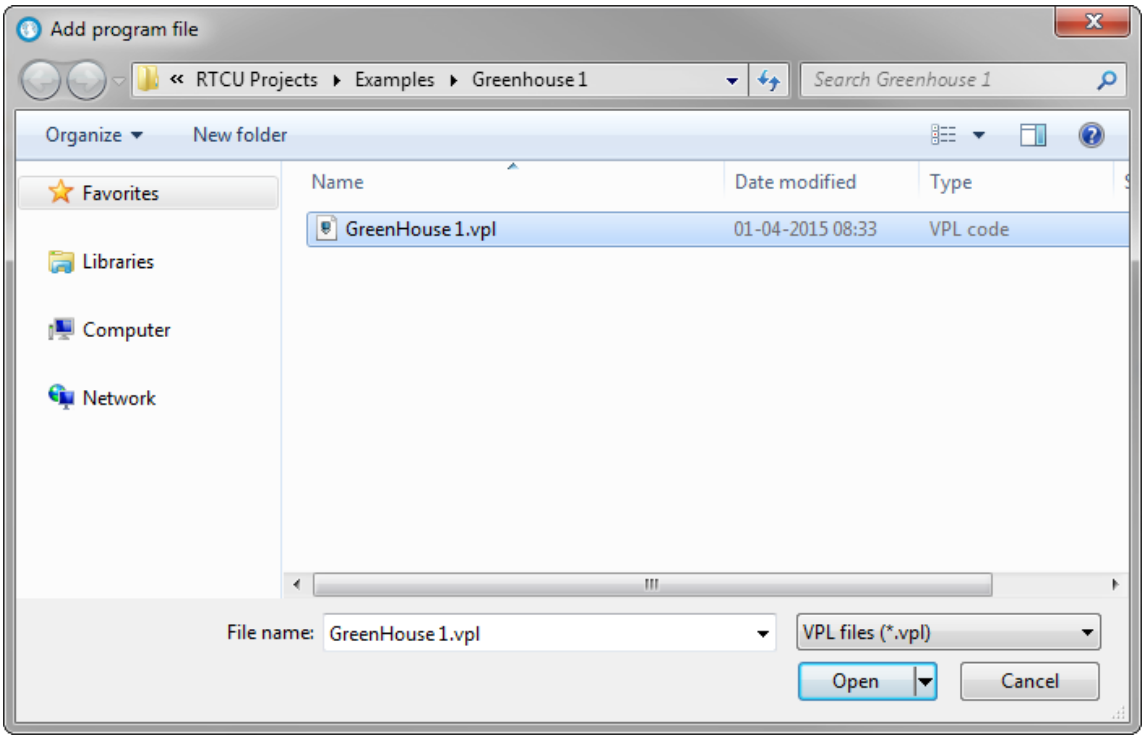
Note that it is not possible to use "Translate" on INC (Include) files.

2.4.3.2.2 Project View: New Program



This allows one to name the new VPL (Program) file.
A skeleton VPL program will be inserted in the file automatically upon creation.

2.4.3.2.3 Project View: Add Program



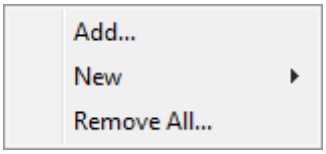
This will add a VPL (Program) to the current project.

2.4.3.3 Project View: Include

2.4.3.3.1 Project View: Include

It is possible to add Include (.INC) files to a project in order for them to be managed from within the project itself. Include files will not be separately compiled by the RTCU IDE but must be included by a Program file (.VPL) file by using the INCLUDE command.

Right-clicking on the "Include" text in the project tree will make the following drop-down menu appear:

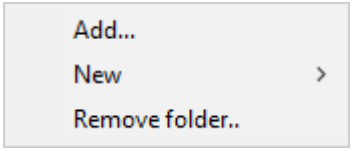


The individual items:

- Add
- New:
  - File
  - Folder

"Remove All" will remove all Include files from the project. Please note that while the files are not actually deleted from the disk, they are no longer referenced and included in the current project.

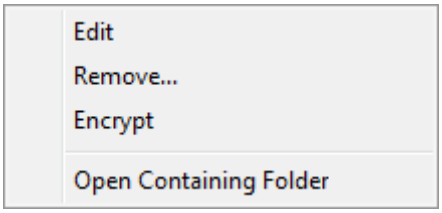
Right-clicking on a folder in the "Include" section of the project tree will make the following drop-down menu appear:



The individual items:

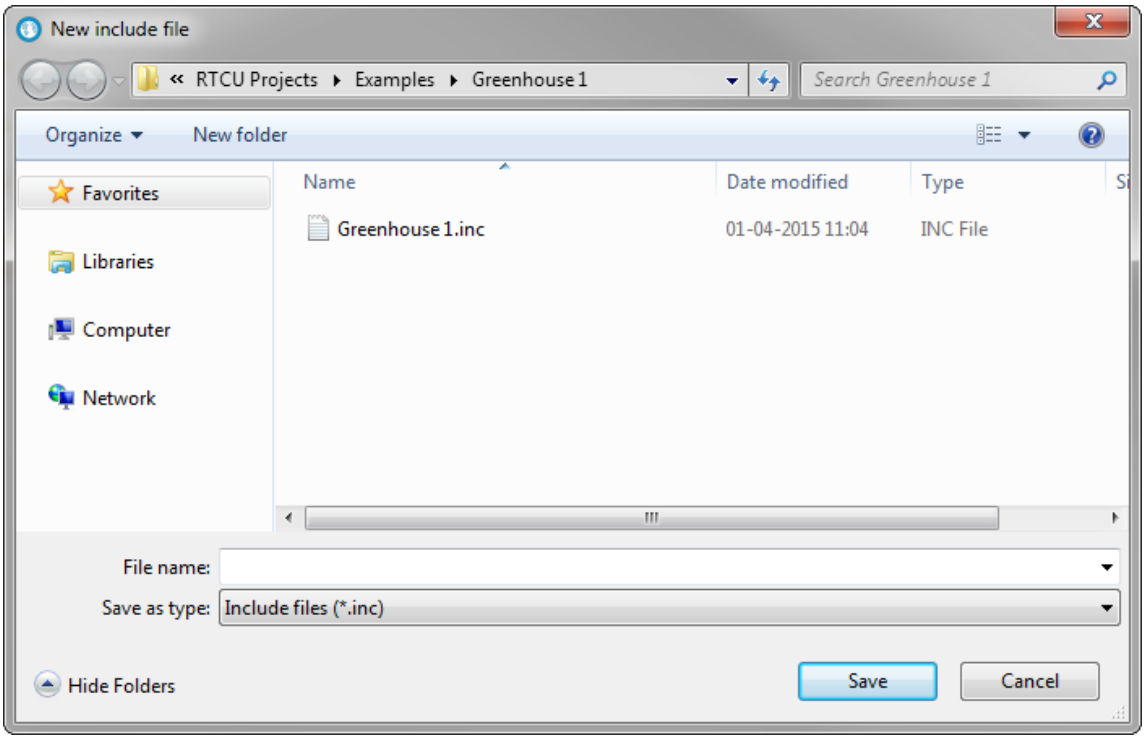
- Add
- New:
  - File
  - Folder
- Remove folder

Right-clicking on a specific file in the "Include" section of the project tree will make the following drop-down menu appear:



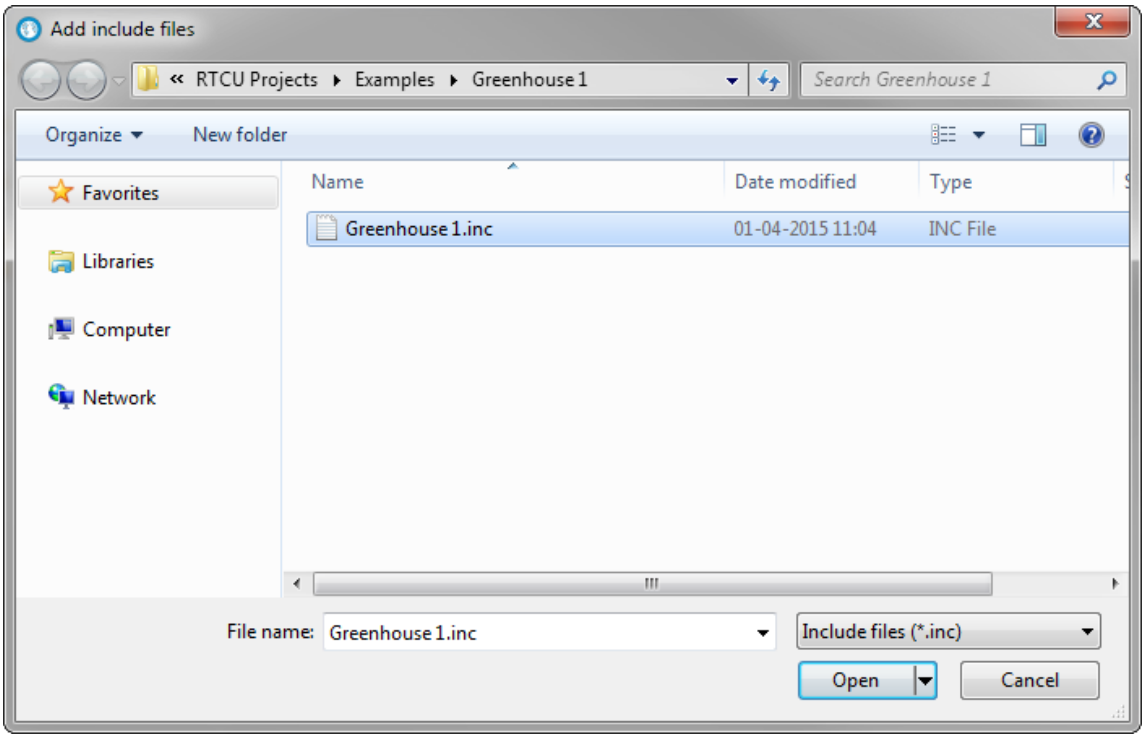
The "Edit" command will open the file in the editor, so that you are able to make modifications to it, save it, print it, and so on.
"Remove" will remove the file from the project (the file is not deleted - just removed from the project).
The "Encrypt" command will generate an encrypted version of the file. The encrypted file has the same name as the source file with the extension "ENC" instead of "INC" and is located in the same folder. For more information see Encrypted include file.
"Open containing folder" will open the folder containing the file in Windows Explorer.

2.4.3.3.2 Project View: New Include



This allows you to name the new INC (Include) file.
Please note that Include files are not automatically compiled but must be included in a VPL program by using the INCLUDE statement.

2.4.3.3.3 Project View: Add Include

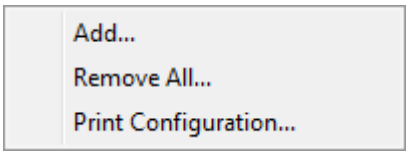


This will add an INC (Include) to the current project.
Please note that Include files are not automatically compiled but must be included in a VPL program by using the INCLUDE statement.

2.4.3.4 Project View: Job

2.4.3.4.1 Project View: Job

Right-clicking on the "Job" text in the project tree will make the following drop-down menu appear:

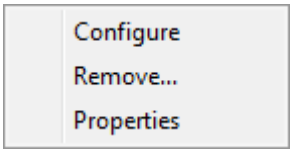


The individual items:

- Add
- Print configuration

"Remove All" will remove all Job files from the project. Please note that while the files are not actually deleted from the disk, they are no longer referenced and included in the current project. Please also note that the project is limited to 16 Jobs.

If you right-click on a specific job in the "Job" section of the project tree, the following drop-down menu will appear:

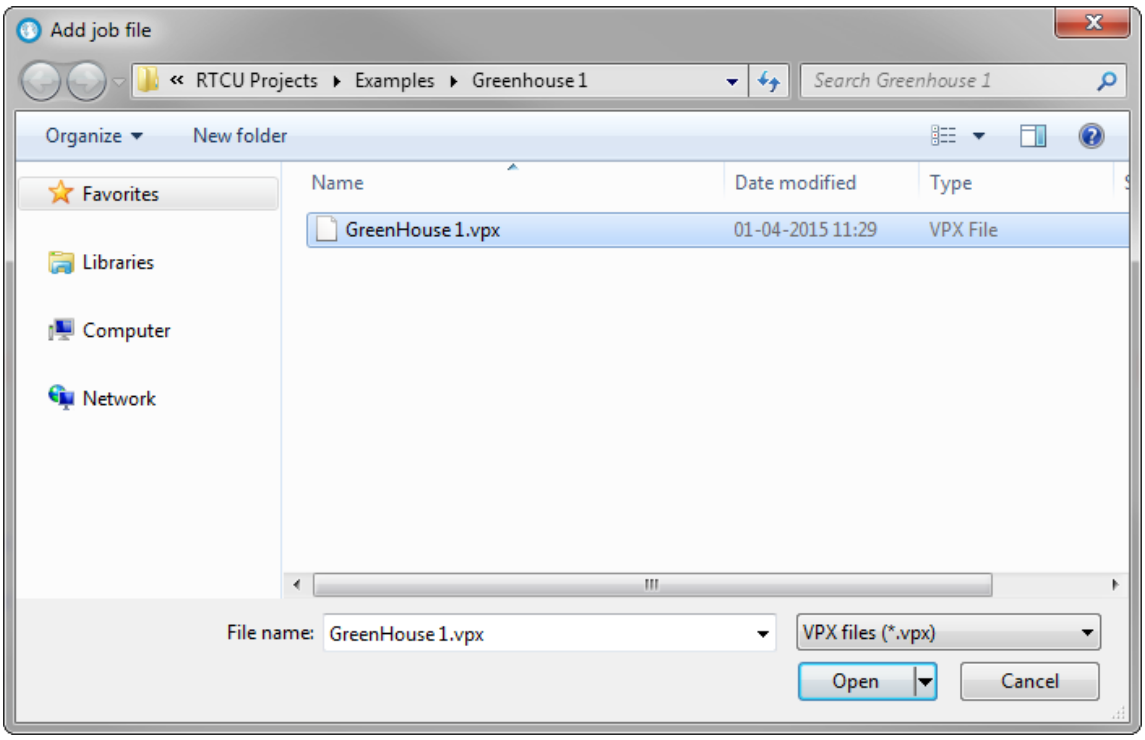


The individual items:

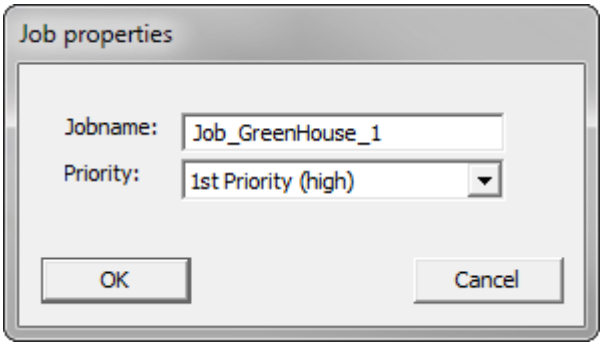
- Configure
- Properties

"Remove" will remove the file from the project (the file is not deleted - just removed from the project).

2.4.3.4.2 **Project View: Add Job**



This enables one to add a job to the current project. The following dialog will present itself:



In this dialog, you can name the job as well as select the priority of the job (high/low).

Please note that all tasks running at the same priority will be run sequentially - switching between each job every cycle.

Assuming for example that three jobs, A, B, and C, are running at the same priority, A would run uninterrupted until END is reached, then B would run until END is reached, then C, then A again, etc. In most cases, it is recommended that one uses true multithreading rather than creating multiple jobs. Please see the section on Multithreading.

2.4.3.4.3

Project View: Print configuration

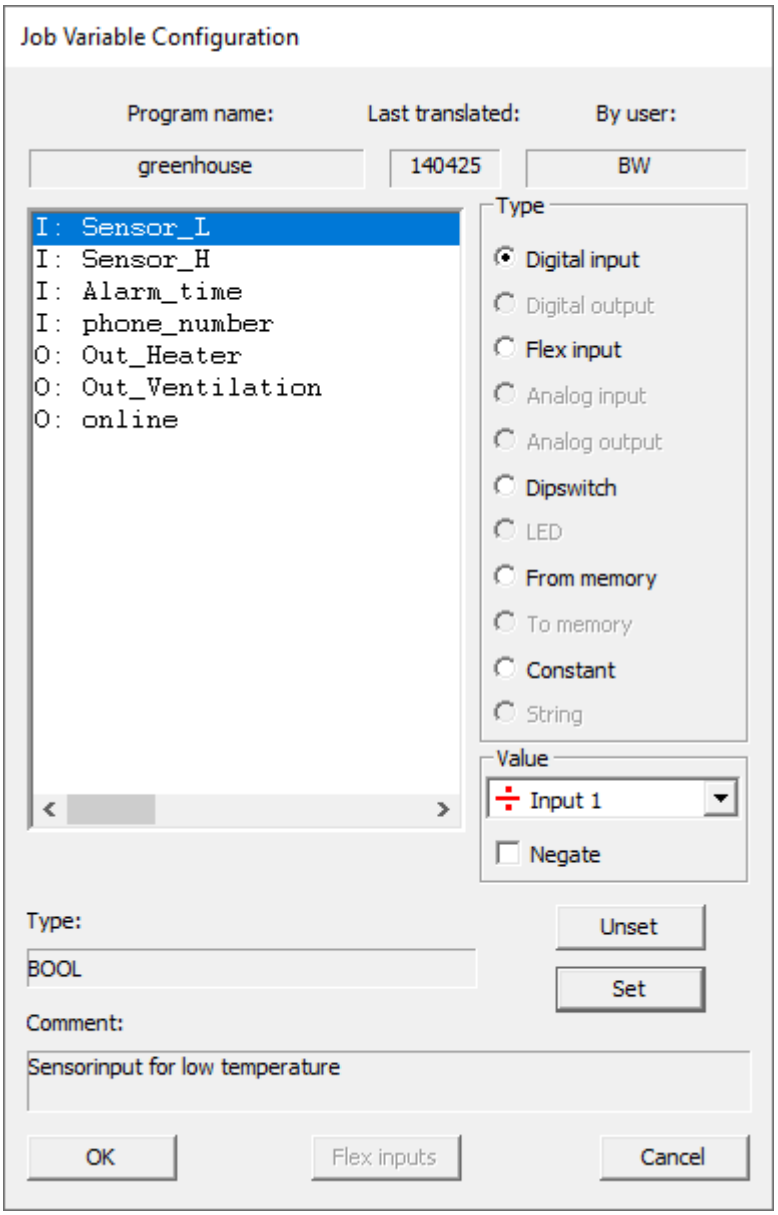
The "Print Configuration" command will make a printout of the current I/O configuration. This will show you which physical in- and outputs are used in your program (shown for each Job). This makes it easier to create the actual connections to an RTCU device. It can also function as documentation for the project.

A configuration printout can look like this:

```
------------------------------------------------------------------------
---------
 Configuration of all I/O signals for project "Greenhouse-1.prj"
 Printed 2001-01-02 11:04
------------------------------------------------------------------------
---------

Job: Job_Greenhouse_1 (Greenhouse_1.vpx)
  ------------------------------------------------------------------------
---------
  I: Sensor_L        Digital Input 0
  Sensorinput for low temperature
  ------------------------------------------------------------------------
---------
  I: Sensor_H        Digital Input 1
  Sensorinput for high temperature
  ------------------------------------------------------------------------
---------
  O: Out_Heater      Digital Output 0
  Output to activate heater
  ------------------------------------------------------------------------
---------
  O: Out_Ventilation Digital Output 1
  Output to activate ventilation
  ------------------------------------------------------------------------
---------
```

2.4.3.4.4 Project View: Configure Job

2.4.3.4.4.1 Project View: Configure Job



This dialog allows one to configure all the configurable parameters of the selected Job - that is to say all variables defined in the VAR_INPUT and VAR_OUTPUT sections.

Using the VAR_INPUT and VAR_OUTPUT sections, it is possible to assign in- and outputs, as well as numerical constants, strings etc., to a Job. This means that even without access to the the source file (.VPL file) for a given Job, it is still possible to assign values to the various in- and outputs of the program. In this way, it is possible just to distribute the .VPX file (Job file) and not the source code (.VPL file) for a particular module.

The list in the left part of the dialog shows all the variables defined in the VAR_INPUT and VAR_OUTPUT sections. The variables with an "I:" in front of them are inputs, and the variables with an "O:" in front of them are outputs. When one of the variables is selected, the two fields at the bottom of the dialog show the type and the comment field of the variable (that is to say the text written with orange in the editor when the program is first entered). In the right part of the

dialog, one can see a list with a variety of different types that you are able to assign to variables. Some of these types are grayed out (not active for this particular type of variable).

NOTE: "Ignition" can be selected as an input when building an X32-, NX32- or NX32L-based project (see Project: Settings).
Ignition refers automatically to the dedicated ignition input available on all devices. The actual digital input depends on the device in question. For example for the RTCU MX2i series, this is input 5, and for the RTCU CX1 series, this is input 1.

Variables of type BOOL can be negated by using the "Negate" checkbox. This is useful if you (or someone else) designed a program for use with a particular sensor, for example a "normally open" type, and the sensor you have in the actual application is a "normally closed" type of sensor. By using the "negate" checkbox it is possible to negate a given in- or output.

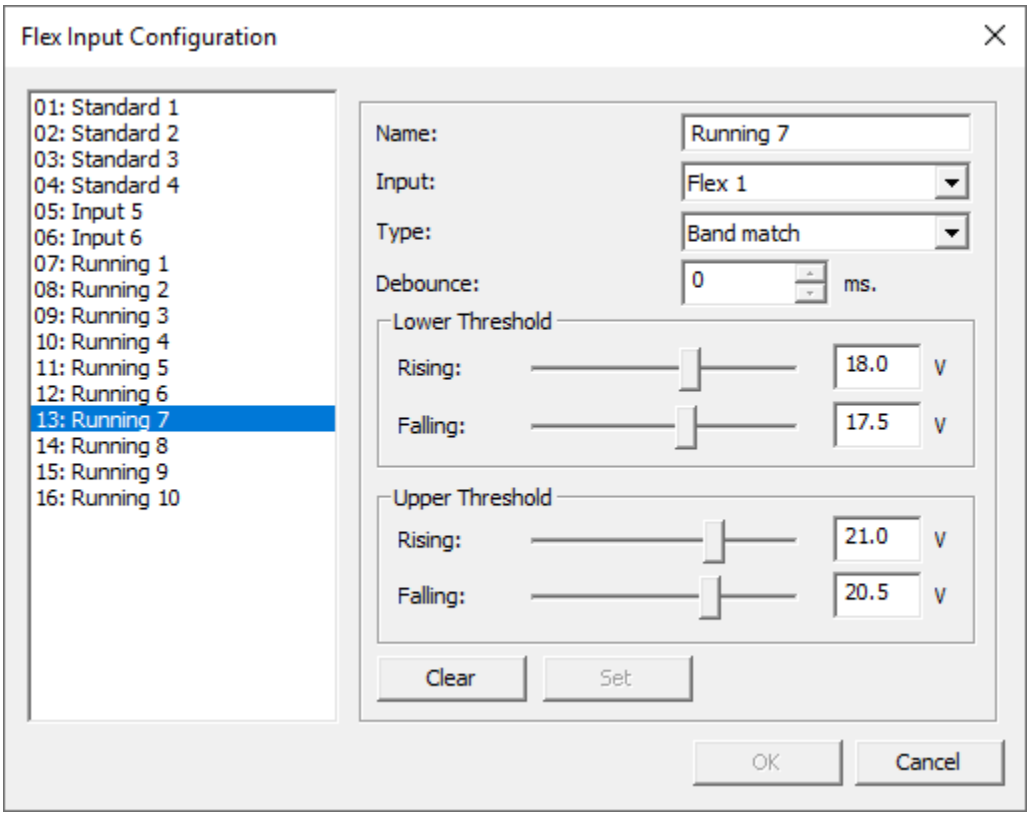
When variables of an integral type, such as INT/DINT, are assigned to an analog in-/output, it must always be taken into consideration what the actual range of the I/O is. Using a 10-bit analog input for example will yield a numeric range of 0..1023.

Using the memory I/O type is it possible to write to or read from a memory location as described in the memIORead and memIOWrite functions. Although there is 4096 memory locations available only the first 256 can be configured from the Job configuration dialog.

At the bottom of this dialog, the selected variables type and comment will be shown.

Clicking the Flex Inputs button will show the Flex Input configuration dialog, for configuring flex inputs as digital inputs.

2.4.3.4.4.2 Project View: Configure Flex Input



This dialog allows one to configure the Digital Flex Inputs.
Please see the technical manual for supported devices for the electrical details.

Name

The name which is displayed in the list of inputs in the Job configuration dialog.

Input
The physical flex input to use.

Type
The behavior of the input:

Standard	- The input acts like any other on-board digital input.
Threshold	- The input switches from OFF to ON, using the configured threshold.
Band match	- The input switches from OFF to ON to OFF, using the lower and upper thresholds.

Debounce
The number of milliseconds the value of the Flex input must be steady before the state is considered valid.
The resolution is currently 50ms.

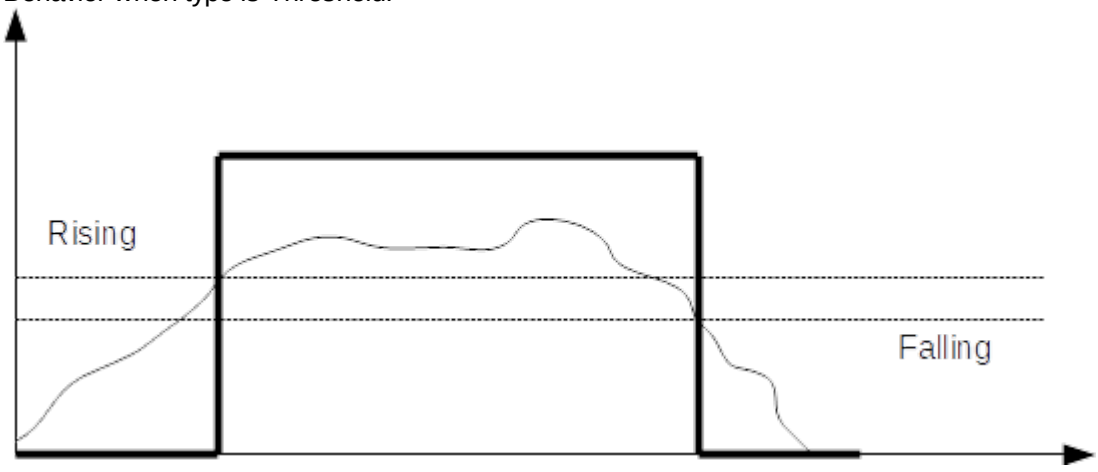
Lower Threshold / Threshold
This is the threshold where the input switches from OFF to ON when the input value rises.

The rising parameter is the threshold for the Flex input where the value changes from OFF to ON when the input value is rising.
The falling parameter is the threshold for the Flex input where the value changes from ON to OFF when the input value is falling.

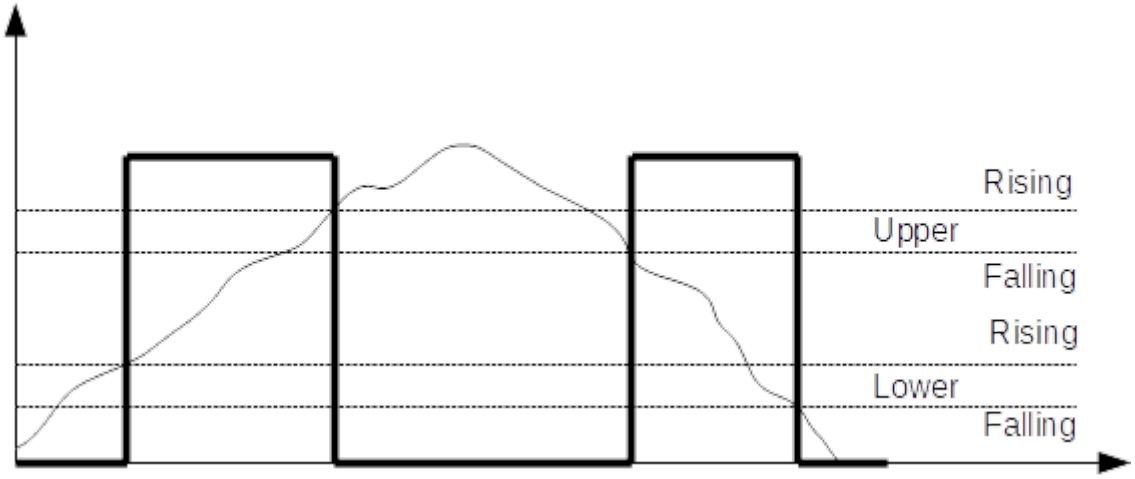
Upper Threshold
This is the threshold where the input switches from ON to OFF while the input value rises.

The rising parameter is the threshold for the Flex input where the value changes from ON to OFF when the input value is rising.
The falling parameter is the threshold for the Flex input where the value changes from OFF to ON when the input value is falling.

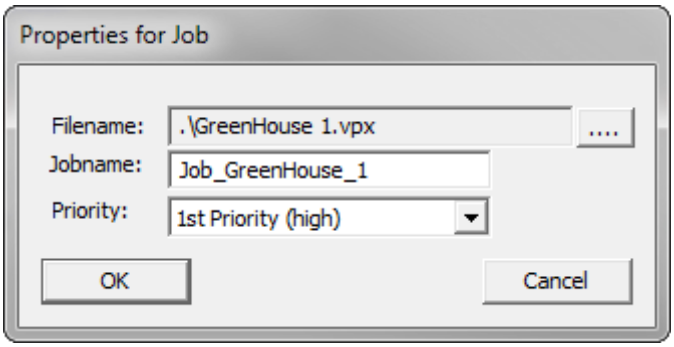
Behavior when type is Threshold:



Behavior when type is Band match:



2.4.3.4.5 Project View: Jobproperties



This dialog allows one to change the properties of a job in the project. The file used for the job can be changed - as can the name and priority of the job.
The priority of a job determines how much processing time each job will receive. A low-priority job will make a pause of 10 milliseconds between each scan (BEGIN/END cycle). A high-priority job will not have this delay.

2.4.3.5 Project View: I/O Extension

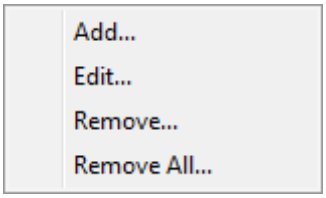
2.4.3.5.1 Project View: I/O Extension

Right-clicking on the "I/O Extension" text of the project tree will make the following drop-down menu appear:

The individual items:

- Add
- Remove All - this will remove all nets related to the project.

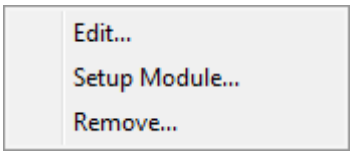
Right-clicking on a specific net in the "I/O Extension" section of the project tree will make the following drop-down menu appear:



The individual items:

- Add
- Edit
- Remove - this will remove the net from the project (the devices of the net are also removed).
- Remove All - this will remove all devices from the net.

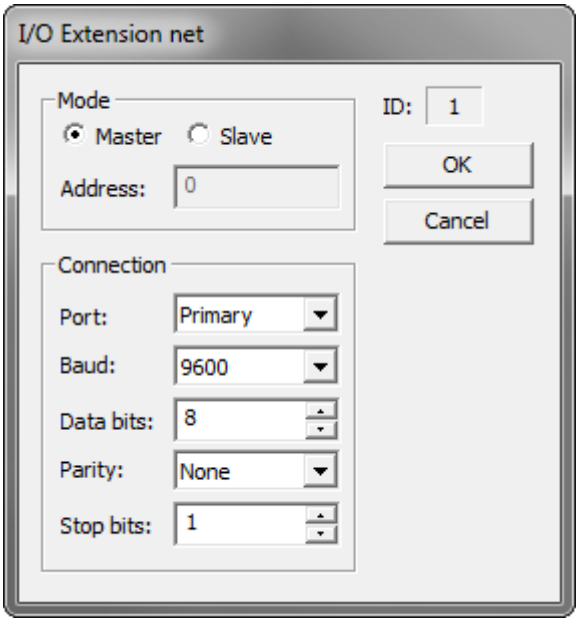
Right-clicking on a specific device in the "I/O Extension" section of the project tree will make the following drop-down menu appear:



The individual items:

- Edit
- Setup module
- Remove - this will remove the device from the net.

2.4.3.5.2 Project View: I/O Extension net



This dialog allows one to configure an I/O Extension net.

Mode:
This is the role that the RTCU device will take on the net.

- Master — In this mode, the RTCU device will monitor the Inputs and Outputs of the remote devices that are present on the net.
- Slave — In this mode, a remote device on the net can monitor the available Inputs and Outputs of the RTCU device.

Address:
This is the address of the RTCU device.
The address is only used when Slave mode is selected.

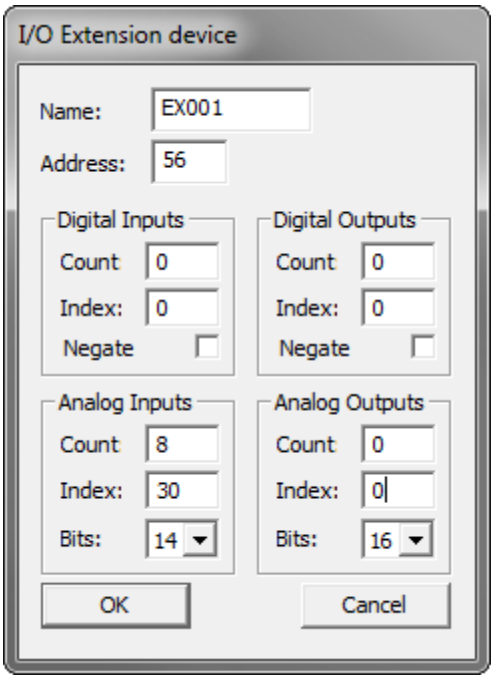
Connection:
This is the MODBUS configuration used to communicate with remote devices on the net.
The configuration will be set to MODBUS default values when a new net is added.
Note: When these values, apart from the port number, are changed, the new setting must be deployed to the device with Setup device.
The port is enumerated as follows:

Port 0	Serial port 0
Port 1	Serial port 1
Port 2	Serial port 2
Primary	Primary RS485 port
Secondary	Secondary RS485 port
	(If not present the primary RS485 port will be used)

(See ser: Serial port for more information)

Note: Not all RTCU devices have three serial ports. Please consult the technical manual for the relevant model.
Note: The RTCU device will always use RS485 if the selected serial port supports this.

2.4.3.5.3 Project View: I/O Extension device



This dialog allows one to configure an I/O extension device.

Please consult the Technical Manual for the specific I/O Extension device for information about what parameters to use.
In all MODBUS I/O Extension devices delivered by Logic IO, there is detailed information on how to set up the module and what to enter in the set-up dialog.

Name:
This the name of the Device.
The name is used to identify the device in the Job configuration dialog, in the I/O monitor window, and in the Emulator.
The name is limited to 6 characters.

Address:
This is the address of the device. Each device on the net must have a unique address - all devices are delivered with address 1 by default.
Note: When changing the address, the new setting must be deployed to the device with Setup device.

Digital Inputs:
This is where the Digital Inputs are configured.
If there are no Digital Inputs in the device, set the parameters to 0 (zero).

Count:
This is the number of digital inputs on the device.

Index:
This is the index of the register used to read the state of the inputs. Only use this if the device does not support the "Read digital inputs" command.

Negate:
This option will negate the state of the digital inputs on the device.

Digital Outputs:
This is where the Digital Outputs are configured.
If there are no Digital Outputs in the device, set the parameters to 0 (zero).

Count:
This is the number of digital outputs on the device.

Index:
This is the index of the holding register used to set the state of the outputs. Only use this if the device does not support the "Write coils" command.

Negate:
This option will negate the state of the digital outputs on the device.

Analogue Inputs:
This is where the Analog Inputs are configured.
If there are no Analog Inputs in the device, set the parameters to 0 (zero).

Count:
This is the number of analog inputs on the device.

Index:
This is the index of the register used to read the state of the inputs.

Bits:
This is the number of bits used in the conversion.
This is used to give the correct range of values in the I/O monitor and the Emulator.

Analog Outputs:
This is where the Analog Outputs are configured.
If there are no Analog Outputs in the device, set the parameters to 0 (zero).

Count:

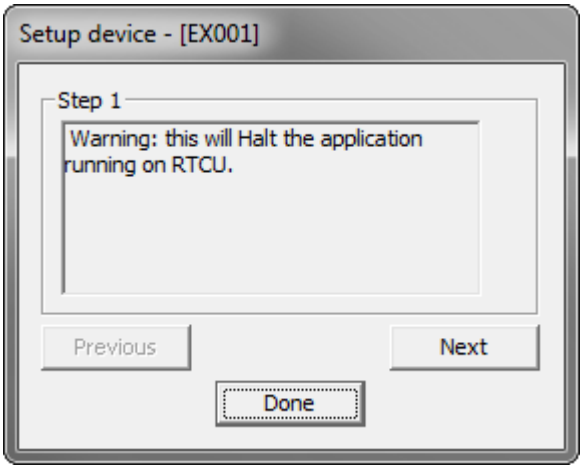
This is the number of analog outputs on the device.

Index:
This is the index of the holding register used to set the state of the outputs.

Bits:
This is the number of bits used in the conversion.
This is used to give the correct range of values in the I/O monitor and the Emulator.

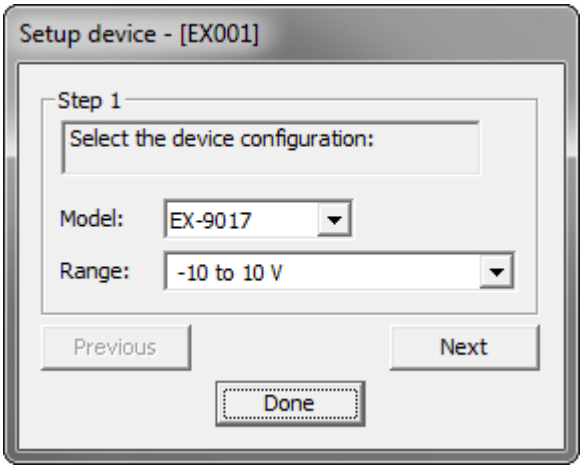
Note:
It is possible to access holding registers by treating them as Analog inputs (to read) or Analog outputs (to write).

2.4.3.5.4 Project View: Setup device



This dialog allows one to configure a MODBUS device delivered by Logic IO without having to do so manually.
The step-by-step guide will show you how to connect the device to an RTCU device before setup.

If the device has analog in- or outputs, the first step changes to:



Model:
Select the model of the Device.
The drop-down list contains the models that are currently supported.

Range:
The range of the Input or Output.

2.4.3.6 Project View: User files

2.4.3.6.1 User files: Project drive

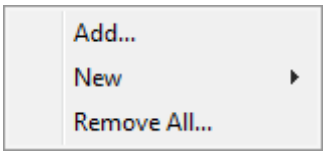
The Intellisync Project Drive supports additional user files, in addition to the application itself. These user files can be Voice Messages or resources that the application needs.

It is possible to drag files and folders to move the item to a different folder. This includes moving to or from the internal drive.

Note: When using X32 architecture projects, this part of the project tree is replaced by Voice Messages

---

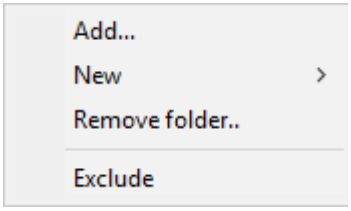
Right-clicking on the "Project drive" text of the project tree will make the following drop-down menu appear:



The individual items:

- Add
- New:
  - File
  - Folder (Only available when using NX32L architecture projects)
  - Voice Message
- Remove all

Right-clicking on a folder in the "Project drive" section of the User files will make the following drop-down menu appear:

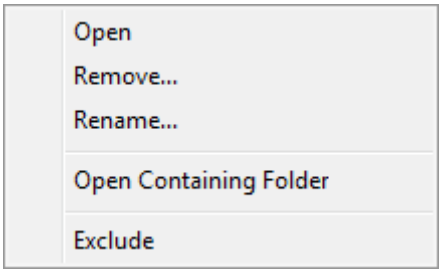


The individual items:

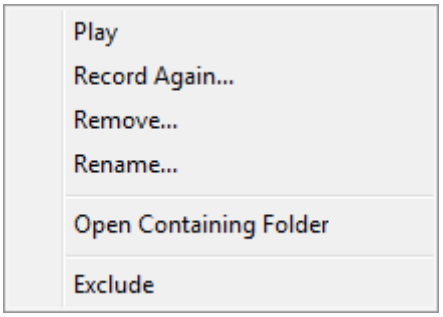
- Add
- New:
  - File
  - Folder
  - Voice Message
- Remove folder
- Exclude can be used to temporarily prevent a folder, including the files and folders contained within, from being included in the RTCU Project Container.

n.b. Folders and files can be dragged and dropped.

Right-clicking on a normal file in the "user files" section of the project tree will make the following drop-down menu appear:



If the file is a .wav file, the following drop-down menu appears instead:



The individual items:

"Open..." will open the file with the system editor, if it is not a known source file.

"Play" will play the selected voice message through the loudspeakers of your PC (This requires your PC to have an audio card).

- Record again
- Remove
- Rename
- Open containing folder
- Exclude can be used to temporarily prevent a file from being included in the RTCU Project Container.

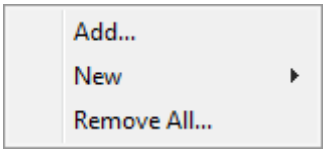
2.4.3.6.2 User files: Internal drive

The user files in this branch will be stored on the internal drive in the device.
These files will automatically be transferred to an NX32L device when the project is synchronized.

It is possible to drag files and folders to move the item to a different folder. This includes moving to and from the project drive.

Note: This can only be used to update and add files, it is not possible to use this to delete existing files.
Note: This part of the project tree is only present, when using NX32L architecture projects.

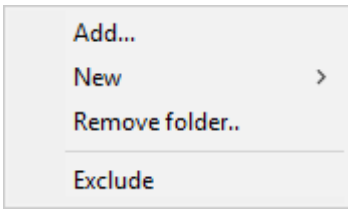
Right-clicking on the "Internal drive" text of the User files will make the following drop-down menu appear:



The individual items:

- Add
- New:
  - File
  - Folder
  - Voice Message
- Remove all

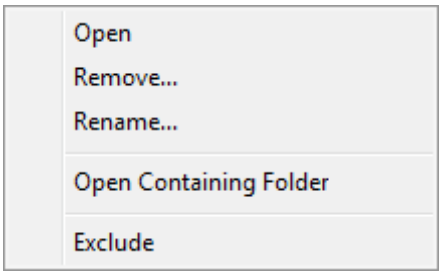
Right-clicking on a folder in the "Internal drive" section of the User files will make the following drop-down menu appear:



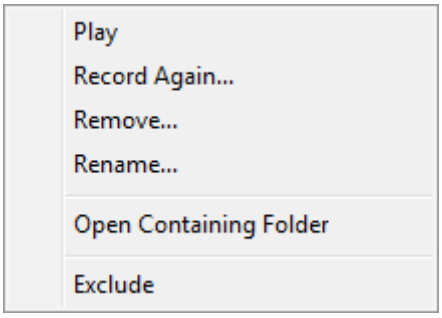
The individual items:

- Add
- New:
  - File
  - Folder
  - Voice Message
- Remove folder
- Exclude can be used to temporarily prevent a folder, including the files and folders contained within, from being included in the RTCU Project Container.

Right-clicking on a normal file in the "Internal drive" section of the User files will make the following drop-down menu appear:



If the file is a .wav file, the following drop-down menu appears instead:



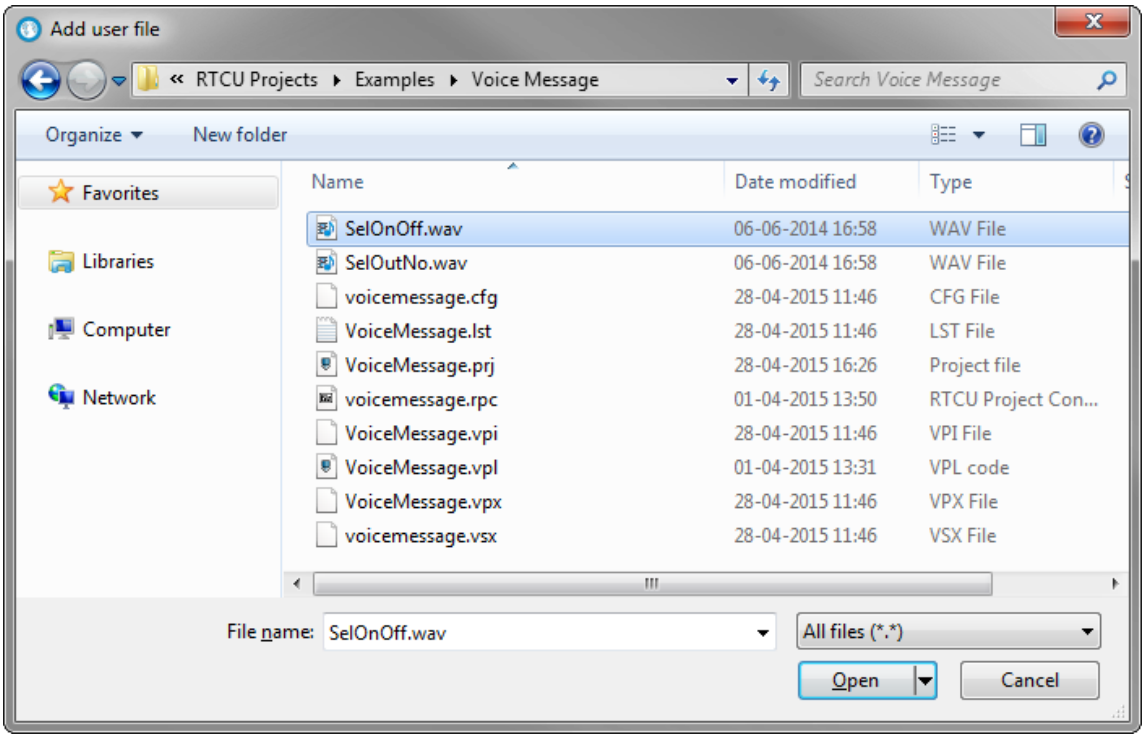
The individual items:

"Open..." will open the file with the system editor, if it is not a known source file.

"Play" will play the selected voice message through the loudspeakers of your PC (This requires your PC to have an audio card).

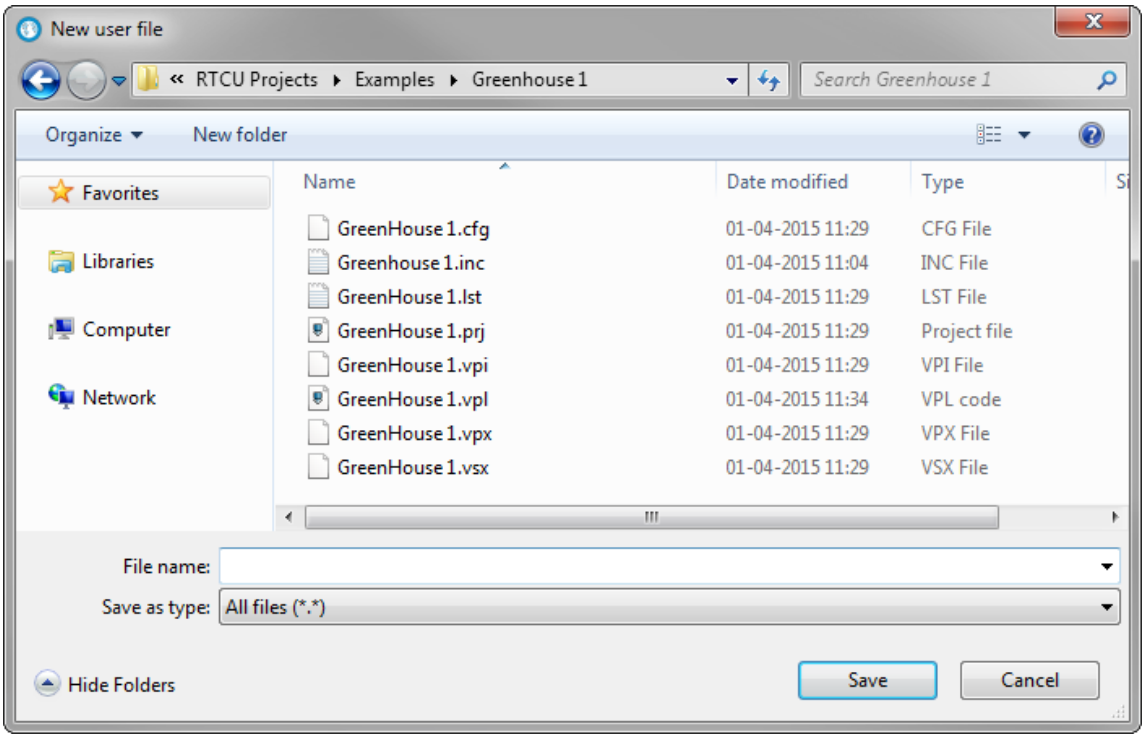
- Record again
- Remove
- Rename
- Open containing folder
- Exclude can be used to temporarily prevent a file from being included in the RTCU Project Container.

2.4.3.6.3 User files: Add File



This will add a file to the project drive.

2.4.3.6.4 User files: New File

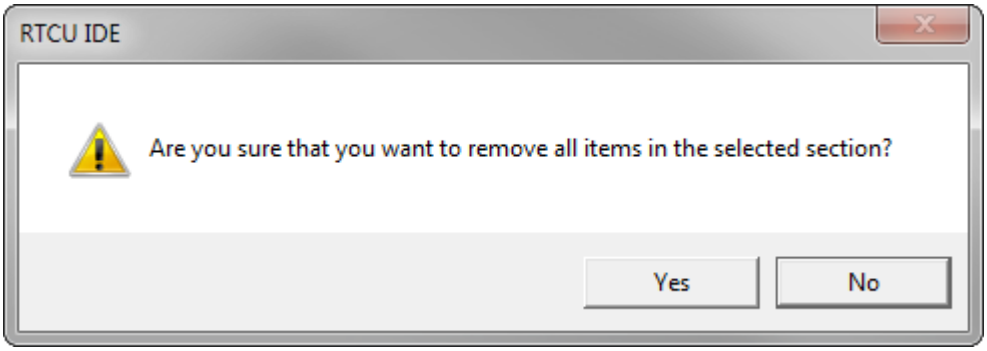


This command creates a new file and includes it in the current project.

2.4.3.6.5 User files: New folder

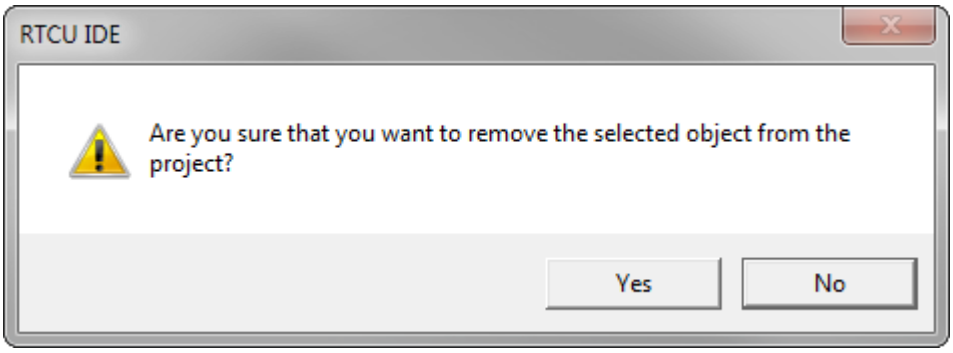
This command creates a new folder in the current project.
The folder will not be created on the disk.

2.4.3.6.6 User files: Remove All



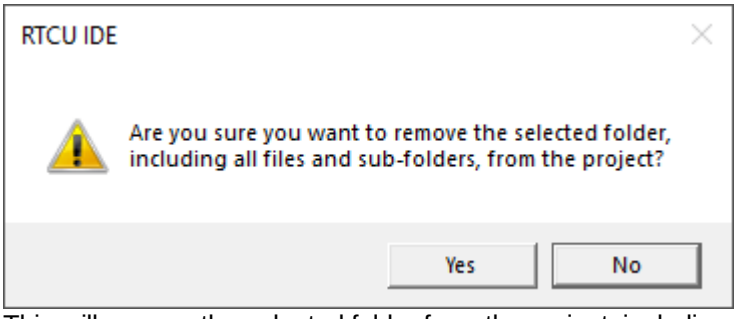
This will remove all the files in the section from the project.
The files will not be deleted, but remains on the disk.

2.4.3.6.7 User files: Remove



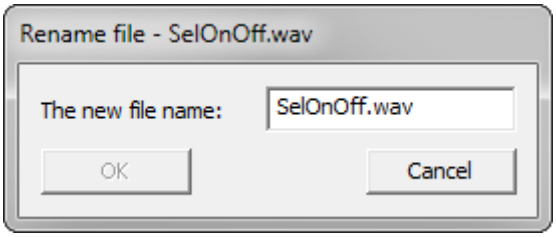
This will remove the selected file from the project. The file will remain on the disk.
It is also possible to delete files by using the "delete" key

2.4.3.6.8 User files: Remove folder



This will remove the selected folder from the project, including any files and folders it contains.
The files will not be deleted, but remains on the disk.

2.4.3.6.9 User files: Rename



This will change the name of the selected files. Note that this also renames the files on the disk.
The OK button remains disabled until a new valid name has been entered.

2.4.3.6.10 User files: Open Containing Folder

This will open the folder containing the file in Windows Explorer.

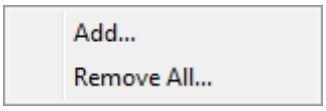
2.4.3.7 Project View: Platform Extension

2.4.3.7.1 Project View: Platform Extension

Platform extensions are supported under the NX32L execution architecture and allows the user to develop and install Programs and Modules.
Please refer to the Platform Extension functions API section for further details.

The Platform Extension part of the project tree, is where the programs and modules required by the project is added.
These platform extensions will be installed automatically when transferring the project to a device.

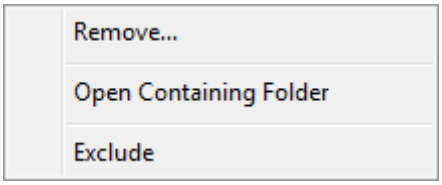
Right-clicking on the "Platform Extension" text of the project tree will make the following drop-down menu appear:



The individual items:

- Add
- Remove all

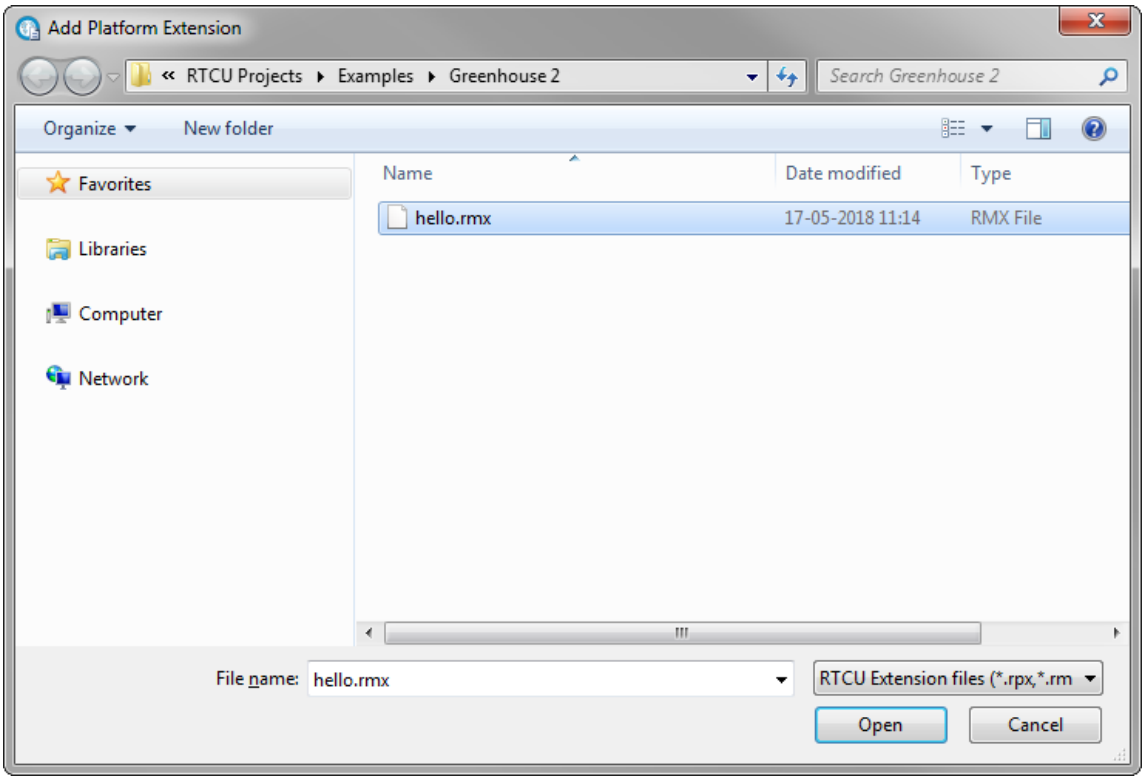
Right-clicking on an extension in the "Platform Extension" section of the project tree will make the following drop-down menu appear:



The individual items:

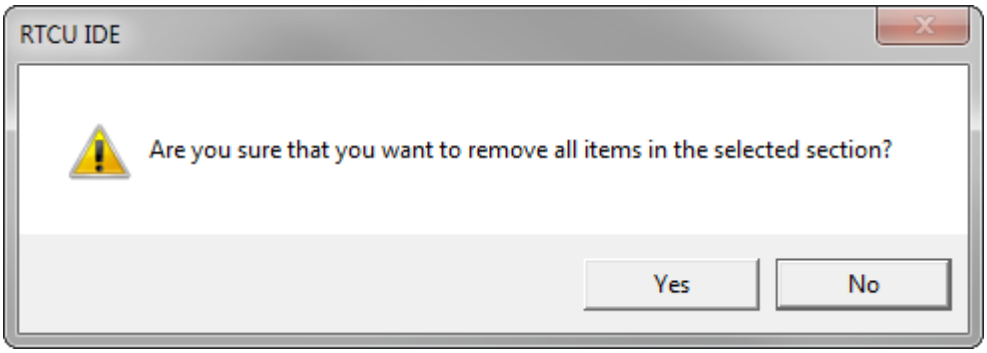
- Remove
- Open containing folder
- Exclude can be used to temporarily prevent an extension from being included in the RTCU Project Container.

2.4.3.7.2 Platform Extension: Add



This will add a Platform Extension to the project..

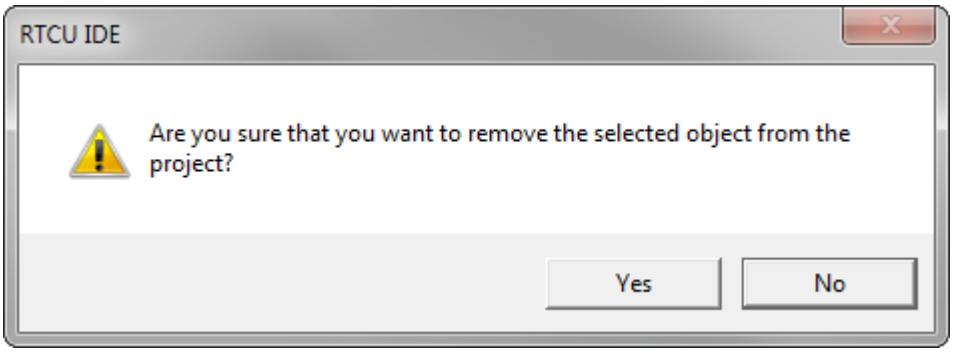
2.4.3.7.3 Platform Extension: Remove All



This will remove all the platform extensions from the project.
The files will not be deleted, but remains on the local file system.

Please note, that removing platform extensions from the project, will not remove the extensions from the device.
To remove the platform extension from the device, use the Platform Extension dialog in the IDE, or the VPL application can delete the file.

2.4.3.7.4 Platform Extension: Remove



This will remove the selected platform extension from the project. The file will remain on the local file system.
It is also possible to delete files by using the "delete" key

Please note, that removing a platform extension from the project, will not remove it from the device.
To remove the platform extension from the device, use the Platform Extension dialog in the IDE, or the VPL application can delete the file.

2.4.3.7.5 Platform Extension: Open Containing Folder

This will open the folder containing the platform extension file in Windows Explorer.

2.4.3.8 Project View: Voice Messages

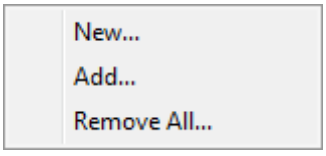
2.4.3.8.1 Project View: Voice Messages

Voice messages can be used on the RTCU platform to deliver spoken messages to, for example, a GSM Module. By using the Voice functions available on some of the RTCU platforms, it is possible to make advanced Voice-Response systems - that is to say alarm systems that can deliver alarm messages using "natural" voice messages. The voice messages are managed through the project tree with the item "Voice messages".

Note: When using NX32 architecture projects, voice messages are to be found under User files

---

Right-clicking on the "Voice messages" text of the project tree will make the following drop-down menu appear:

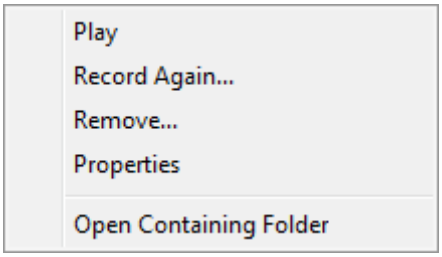


The individual items:

- New
- Add

"Remove All" will remove all voice message files from the project. Please note that while the files are not actually deleted from the disk, they are no longer referenced and included in the current project.

Right-clicking on a specific voice message in the "Voice messages" section of the project tree will make the following drop-down menu appear:

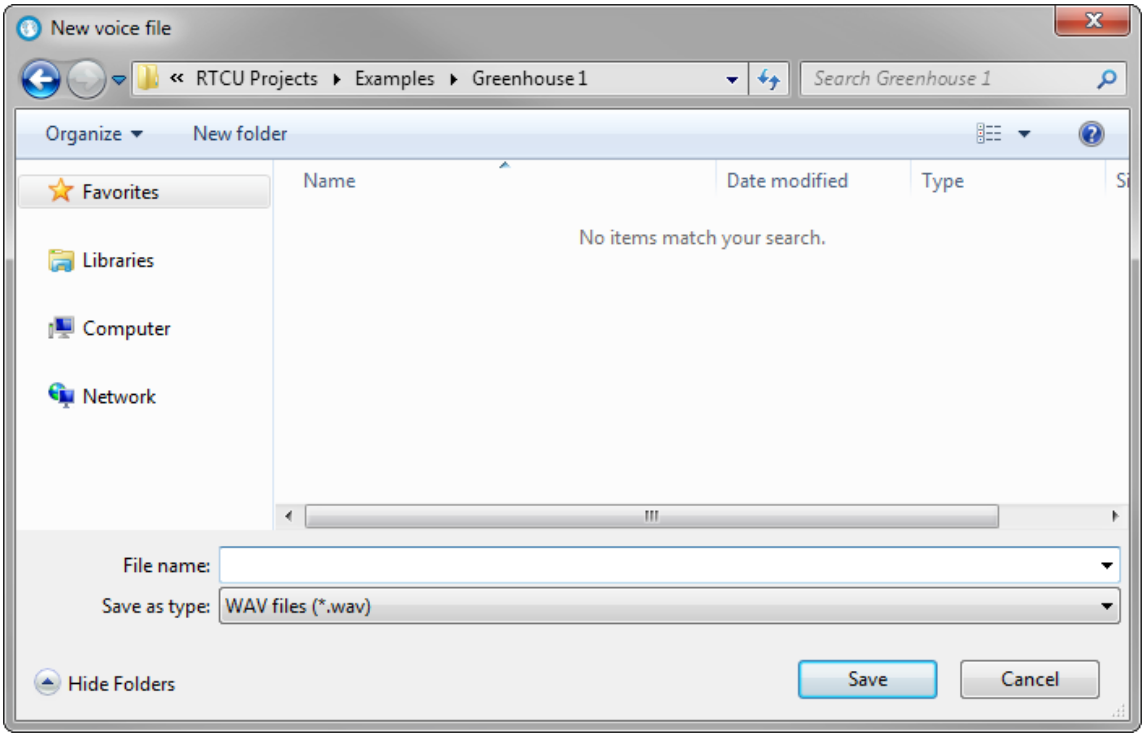


The individual items:

"Play" will play the selected voice message through the loudspeakers of your PC (This requires your PC to have an audio card).

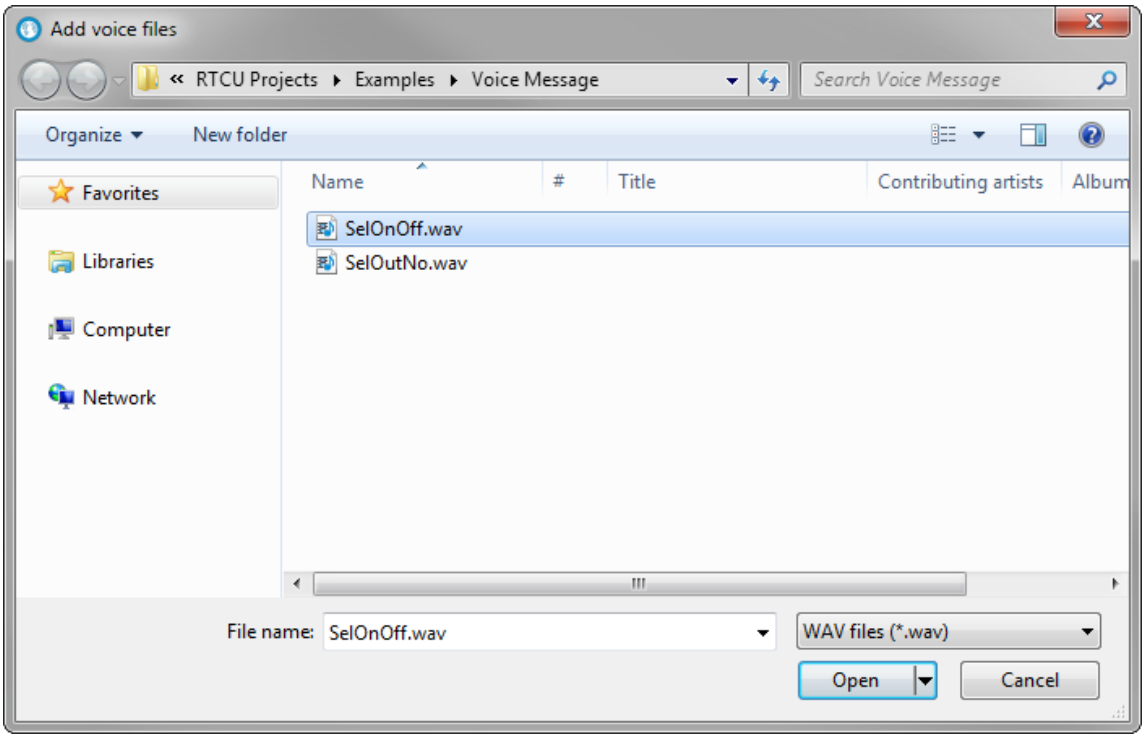
- Record again
- Remove
- Properties
- Open containing folder

2.4.3.8.2 Voice Messages: New Voice File



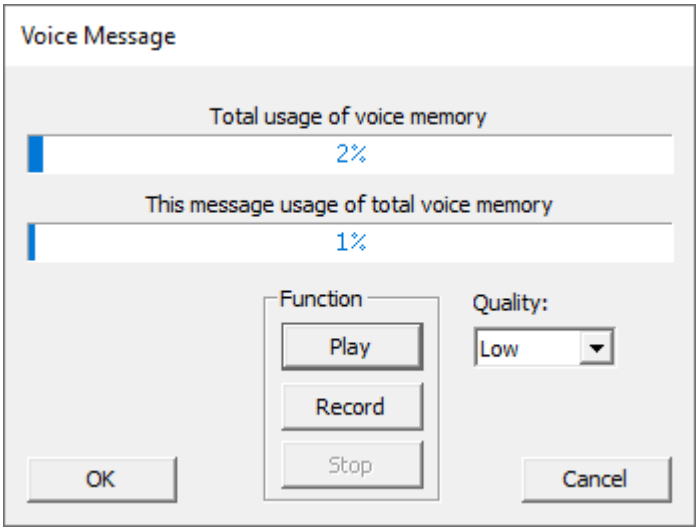
This command creates a new voice file, includes it in the current project and opens the Voice Recorder to record the contents of it.

2.4.3.8.3 Voice Messages: Add Voice File



This command adds an existing file to the current project.

2.4.3.8.4 Voice Messages: Voice Recorder



This is the built-in recorder for voice messages. The dialog shows the total amount of used storage for voice messages, as well as how much of the total memory this specific message is occupying. These two numbers are updated dynamically while a message is being recorded.
Pressing the "Play" button will playback the message on the PC.
Pressing the "Record" button will start recording a message, overwriting the current message.

The quality drop down selects the quality of the recorded speech, which can be either low, standard or high.

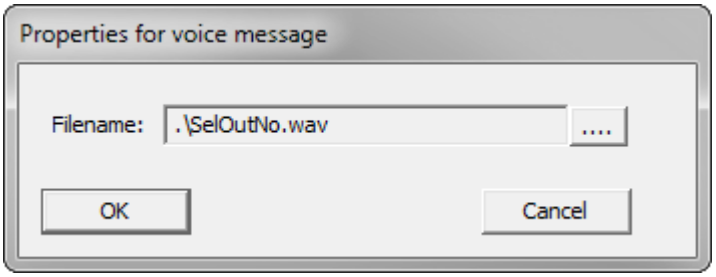
The voice messages are stored in the WAV audio format.

The table shows the details for the quality settings:

Quality	Low	Standard	High*
Sample rate	4 kHz	8 kHz	16 kHz
Data bits	8	8	16
Channels	Mono	Mono	Mono
Bit rate	32 kbit/s	64 kbit/s	256 kbit/s

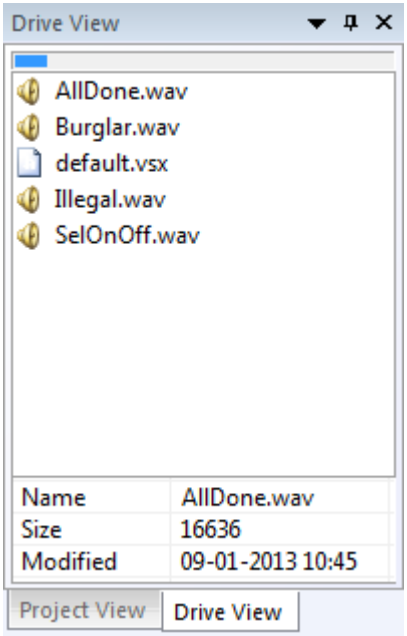
*only available for NX32L projects.

2.4.3.8.5 Voice Messages: Properties



This allows one to change the file name of an included voice message.

2.4.4 Drive View

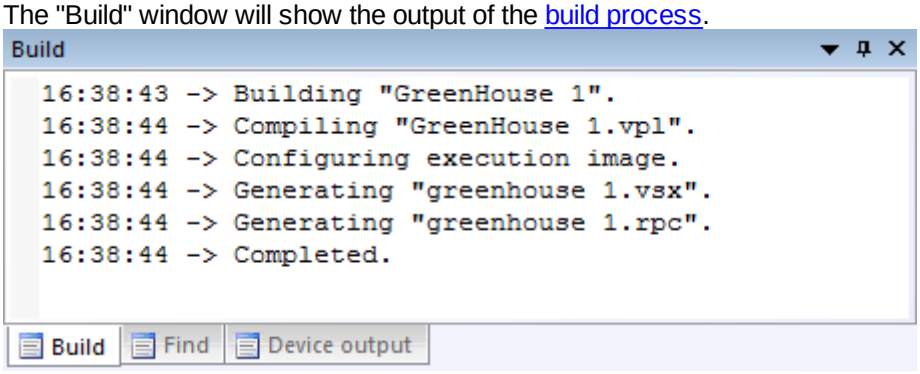


The Drive View is used when building in the NX32/NX32L compilation mode and is used to show the contents of the Intellisync Project Drive.

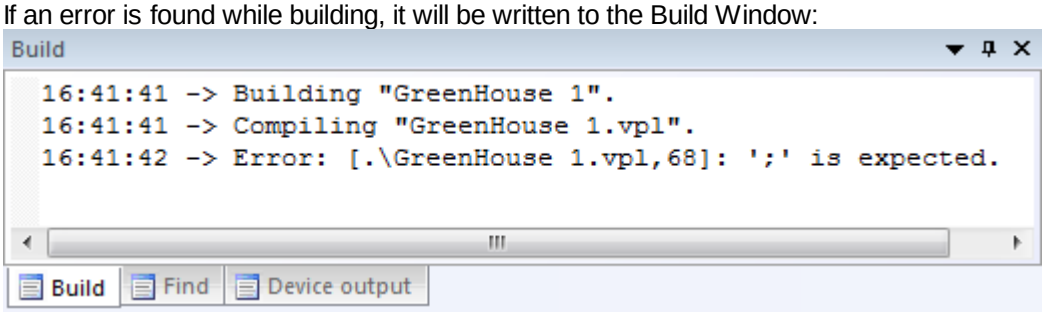
The bar at the top shows how much of the available space on the drive that is in use.
At the bottom, the properties of the selected file are listed.

2.4.5 Build Window

The "Build" window will show the output of the build process.

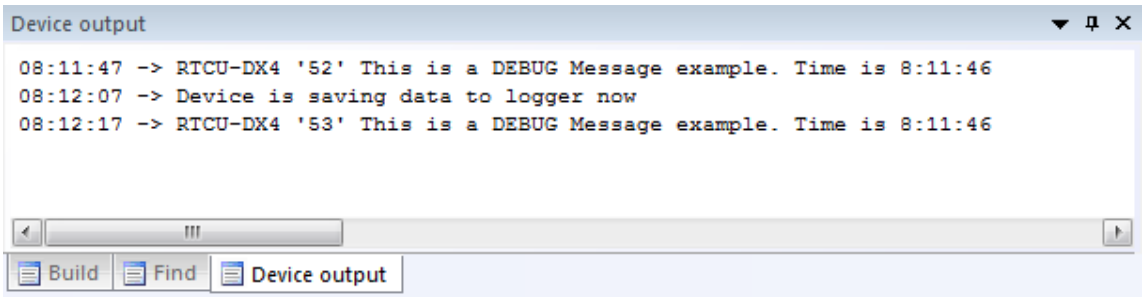


If an error is found while building, it will be written to the Build Window:



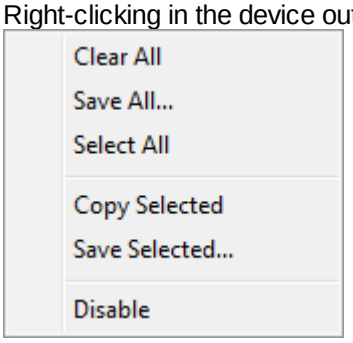
Double-click on an error to open the file and select the line with the error.

2.4.6 Device output



The "Device output" window will show the output of any DebugMsg() or DebugFmt() commands from a connected RTCU device.

Right-clicking in the device output window shows this menu:



Press "Clear All" to clear the contents of the device output window.
Press "Save All" to save the contents of the device output window as a text file.
Press "Select All" to select all the lines in the device output window.

Press "Copy Selected" to copy the selected lines in the device output window to the clipboard.
Press "Save Selected" to save the selected lines in the device output window as a text file.

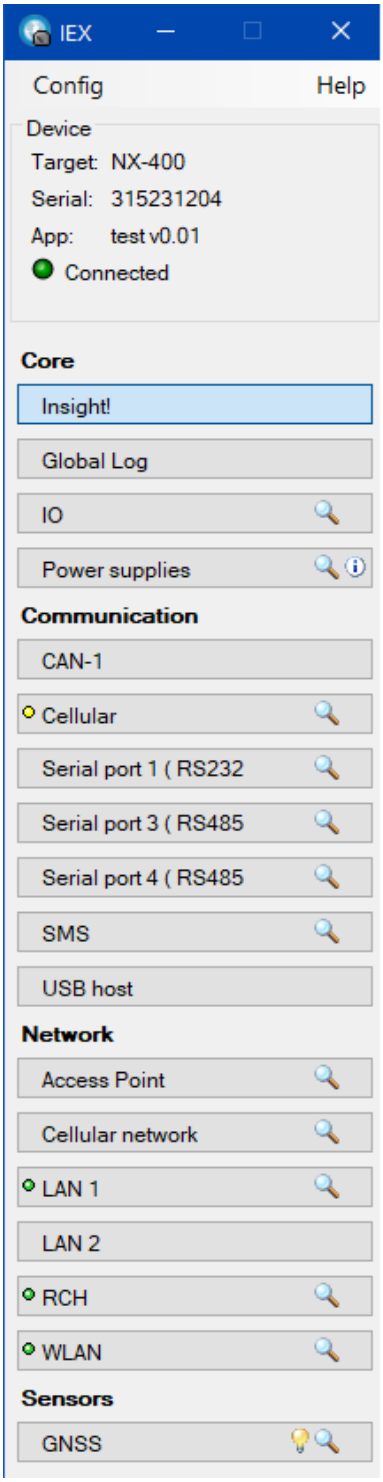
The "Disable/Enable" menu point to control whether the debug messages should be shown or not. After connection has been established to an RTCU device, either by cable or wireless, the menu will show which state the debug messages are in currently.

If a message is longer than 239 characters, it will be split across multiple lines in the RTCU IDE and the split lines will be terminated with "...".

2.5 RTCU Instrumented Execution (IEX)

2.5.1 RTCU Instrumented Execution (IEX)

The **RTCU Instrumented Execution (IEX)** is a supplementary product to the **RTCU IDE** that significantly enhances the platform's test and debugging capabilities.



Using **IEX**, it is possible to monitor and instrument a running application on any NX32L-capable device. Hardware virtualization supports a swift switch between physical and virtual hardware so that, for example, only the I/O is virtual, whereas the remaining hardware is physical. With **Insight!**, a new and powerful window into application execution opens up, as all variables and threads can be monitored and changed live. **RTCU IEX** is a versatile tool that can be used in a development scenario as well as in a production environment where a hard-to-find bug or hardware issue needs to be analyzed.

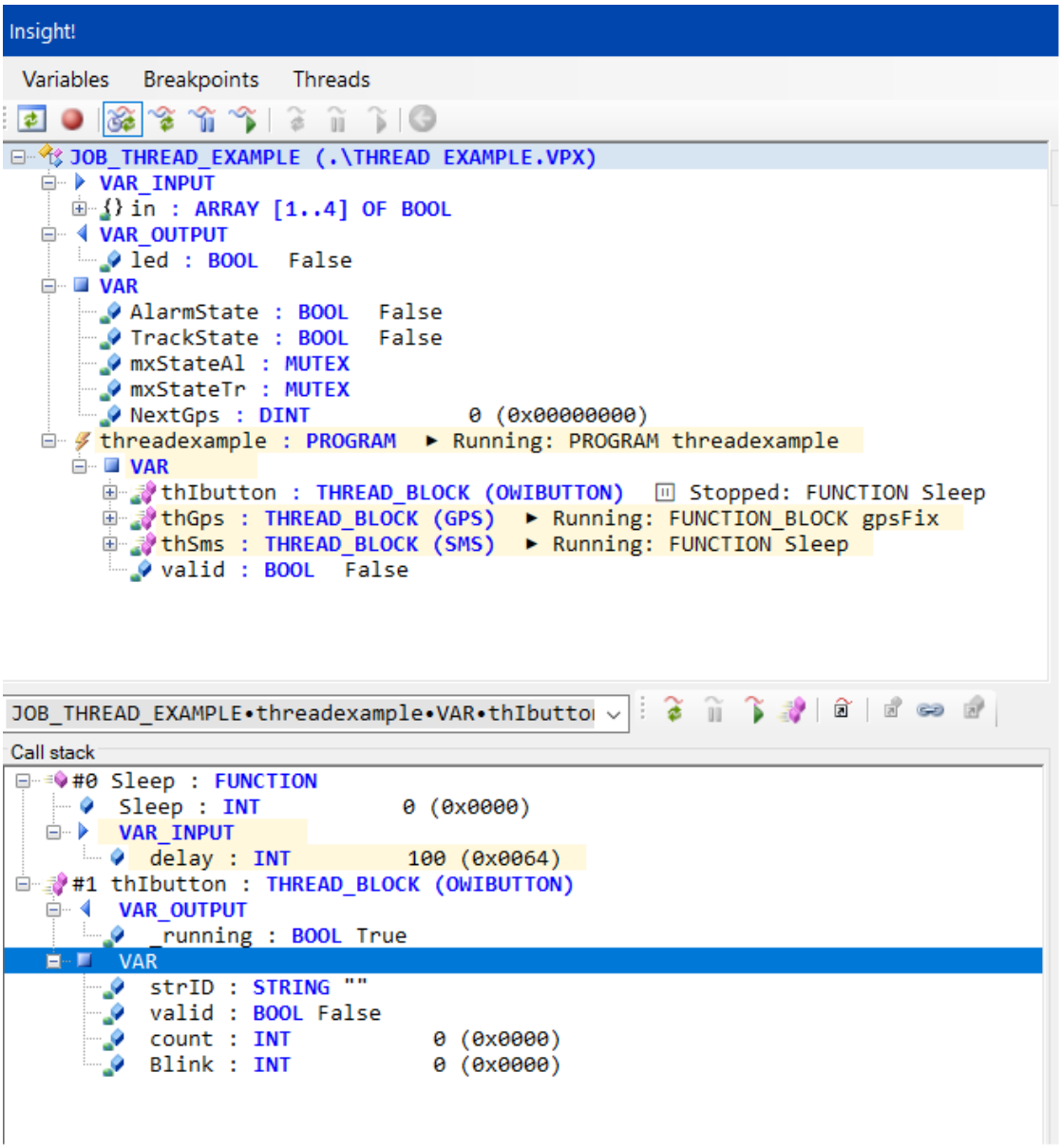
The **RTCU IEX** is a powerful successor to the RTCU Emulator. As it runs on a physical device, it gives more exact information on an application's behavior. It uses virtual hardware similar to the RTCU Emulator but with shadowed physical hardware available at an instance. This allows for a gradual transition from full virtualization to entirely physical hardware as development progresses.

RTCU IEX - key features:

- Allows emulator-like execution in a physical device.
- Automatically integrates with the **RTCU IDE.**
- Supported by all NX32L capable devices.
- Comprehensive logging functionality.
- Local cable connection. Remote access is under development.
- **Insight!**
  - Live view/editor for all global variables.
  - Threads can be stopped and resumed at any time.
  - A coded breakpoint can stop a thread.
  - The call stack of a stopped thread can be inspected/modified.
  - Automatic reference to PTR, ACCESS, and global variables.
  - Search for variables.
- **Virtual hardware:**
  - Digital and Analog input/outputs.
  - Power and battery.
  - CAN buses.
  - GNSS / GPS.
  - RS232 ports.
  - RS485 ports.
  - SMS / PDU.
  - Network interface state.
- **Monitoring**:
  - Detailed cellular information.
  - Detailed WLAN information.
  - RCH information
  - Access Point information.

- USB host port information.

The **RTCU IEX** can be downloaded for free from www.logicio.com



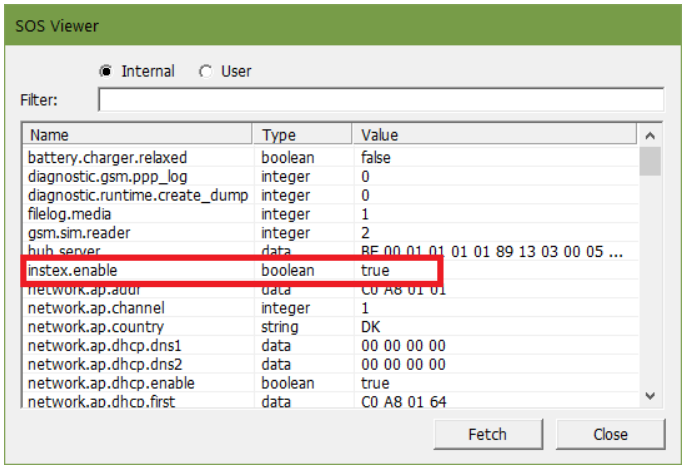
2.5.2 Getting started with IEX

This is a short guide how to get started using the **RTCU IEX**.

The following steps assumes that the **RTCU IDE** with **IEX** support is already installed:

1. Download the **RTCU IEX** installer.
2. Run the installer and wait completion.
3. After the installation a new toolbar is added to the RTCU IDE (may require a restart):

4. With the toolbar buttons it is possible to: Launch, Close and Connect to the **RTCU IEX**.
5. Download a suitable firmware with IEX support from www.logicio.com.
6. Transfer the firmware to a connected RTCU device and await installation.
7. Enable IEX in the device. Go to [Device]-[Data]-[SOS Viewer] and locate the "instex.enable" variable. Set it to TRUE:



8. Restart the device.
9. In the project settings:
   a. "Architecture" must be set to NX32L.
   b. "Debug" must be enabled.
   c. "Insight!" must be enabled.
10. Build and transfer the project to the device.
11. Press the toolbar Launch button and await the the **RTCU IEX** to start with the status "Waiting for device".
12. Press the toolbar Connect button and wait until the **RTCU IEX** reports "Connected" and the control panel unfolds.
13. **The RTCU IEX is now ready for action!**
14. Please consult the **RTCU IEX** documentation for further details and usage.

2.5.3 BREAKPOINT

The **BREAKPOINT** keyword will enter an execution breakpoint that will stop the thread in question when enabled in the **RTCU IEX**.
When breakpoints has not been enabled (default state) they will have no effect-.

Please refer to the **RTCU IEX** documentation for details and usage.

Example:

```
INCLUDE rtcu.inc

VAR
   a   : INT;
   str : STRING;
END_VAR;

PROGRAM test;

BEGIN
   a := 0;
   WHILE a < 1000 DO
      a := a + 1;
      IF a = 500 THEN
          //When breakpoints are enabled the thread will stop here:
         BREAKPOINT;
      END_IF;
   END_WHILE;
END;

END_PROGRAM;
```

2.6 RTCU Emulator

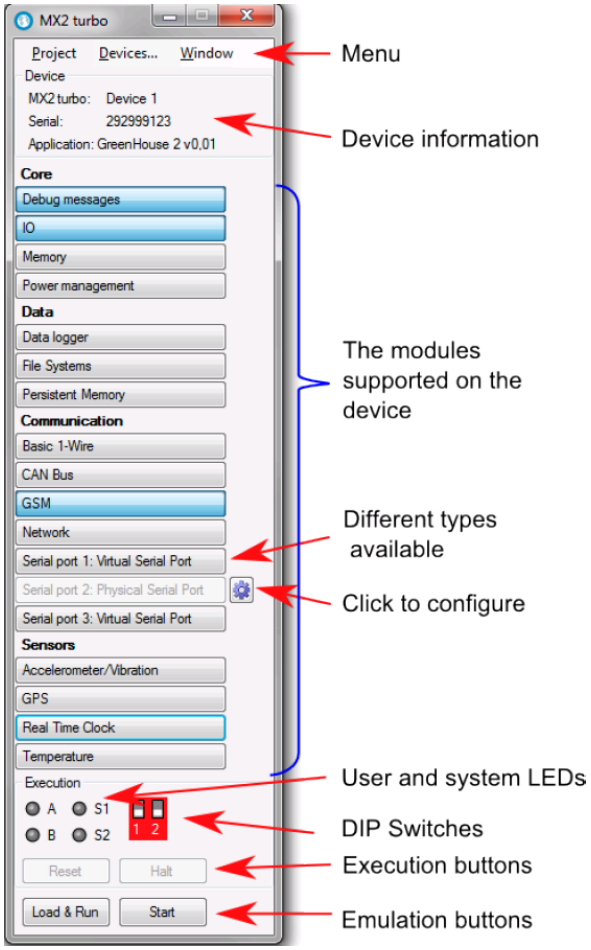
2.6.1 RTCU Emulator

The **RTCU Emulator** is a separately installable virtual execution environment that precisely emulates X32/NX32 devices.

The **RTCU Emulator** can be used as a stand-alone tool or it can be integrated swiftly into the RTCU IDE in order to make coding and test tightly connected. After installation the **RTCU Emulator** can be invoked and controlled directly from within the RTCU IDE Emulator menu item.

The **RTCU Emulator** is based on the same source code as the actual RTCU device firmware. Almost all features and the behavior of each and every physical device are precisely emulated.
This ensures a precise emulation with only a few limitations as described in the RTCU Emulator documentation.

Using the **RTCU Emulator** gives the developer a very powerful tool allowing test and debugging of applications in the convenience of a Windows environment - without any need for a physical RTCU device.



RTCU Emulator advantages:

- Based on the RTCU M2M Platform.
- Precise RTCU device emulation.
- Device specific emulation.
- Protected Execution Environment.
- Multiple devices with separate instance state.
- Device file-system using VHD files.
- Virtual Programming cable connection.
- Remote access over the RTCU Communication Hub.
- RTCU Deployment Server support.
- Powerful RTCU IDE integration.
- Download FREE from www.logicio.com

VPL Programming Language

3 VPL Programming Language

3.1 VPL Programming Language

The VPL language is a high-level programming language similar to Basic or Pascal, but there is a big difference which makes the VPL language very suitable for M2M applications.

Please examine the following sections to be given a better understanding of the many possibilities of the VPL programming language:

3.2 VPL Introduction Manual

The VPL language is a high-level programming language similar to Basic or Pascal, but there is a big difference which makes the VPL language very suitable for typical M2M applications.

Look at the following simple VPL program:

```
1.   include rtcu.inc;
2.
3.   var_input
        iA : bool; | This is P122
        iB : bool; | This is P123
        qA : bool; | This is P124
     end_var;

4.   var
        t : bool;
     end_var;

5.   program test;

6.   begin
        if iA and iB then
           qA := not qA;
        end_if;
     end;
     end_program;
```

(The numbers to the left are not part of the actual program.)

The above is a very simple VPL program which just inverts a variable when two others are true.

Description of the above items

1. The file "rtcu.inc" always has to be included. In this file, all the standard functions/function blocks are defined. Naturally, the user can create his own files with his own functions/function blocks which can then can be included.

2. At this place, there would, in a more realistic VPL program, be defined functions and function blocks.

3. The definition of the variables which can be configured outside the program with the help of the Configurator. Typically, the candidates will be the variables which have to reflect the physical in- or outputs. Apart from that, you often can find instances of assorted function blocks as timers. The text after "|" is a special comment which is attached to the variable. This information will be present in the translated program so that the exact information about a variable always will be available. VPL also offers traditional comments where the text is ignored by the translator:
   // the rest of the line is ignored.
   /**/ the content between /* and */ is ignored.

4. The variables which are local for the program. These cannot be configured outside of the program.

5. This indicates that the actual program begins here.

6. Main program.

When a VPL program is written and translated, one or more jobs will need to be created. Each job is then configured before the actual download and execution can be done.
The configuration "tie" the logic variable-names to the physical I/O.

Declaration of variables / arrays

VPL offers, among others, these basic types:

- BOOL Can hold the values TRUE or FALSE.
- SINT Can hold the values from –128 til 127.
- INT Can hold the values from –32768 to 32767.
- DINT Can hold the values from –2147483648 to 2147483647.
- STRING Can hold a string.

When choosing a type, it is preferable to select the most suitable because a bigger range of values occupies more space in the working storage.
A variable-name (identifier) consists of letters and numbers. The first character has to be a letter.
An initial value can be assigned in the declaration. If no assignment in the declaration statement is done, the value of the variable will always be 0 (false, off).

An identifier (variable name) has a maximum length of 80 significant characters except for global VAR_INPUT / VAR_OUTPUT declarations where the number of allowable characters is 20.

The assignment of the initial value is done like this:

:= <constant> after the type of the variable.

For example:

```
BB : DINT := 100;
```

When defining constants, VPL can use decimal, binary, octal and hexadecimal notations. This is done by adding respectively: 2#, 8# and 16# before the constant in question.
For example:

```
BB : DINT := 16#ABCD;
CC : INT  := 2#1000_1111;
```

The last example indicates that it is allowed to add '_' (underscore) at an arbitrary place in a constant for enhancing the readability.

Arrays:

VPL allows declarations of one-dimensional arrays.

For example:

```
DD : ARRAY[10..20] OF BOOL;
```

The above line declares DD as an array of BOOL with an indexing range from 10…20 which means that DD contains 11 elements. The usage of such a variable could be something like this:

```
FOR j:=10 TO 20 DO
  DD[j]:=ON;
END_FOR;
```

The above line assigns all the elements in DD the value ON. If an illegal indexing is tried, it will trigger a run-time error.

Arrays can also be assigned initial-values, like:

```
DD : ARRAY[10..20] OF INT := 1234,5678,2234,5677;
```

In the above example, the elements with indexes 10,11,12,13 will be initialized to the above values while the remaining elements will automatically be set to 0 (zero).

Control structures

In this section, we will describe a number of control structures which the VPL language offers. These structures will be shown in some small template-like applications.

Selection: IF statement

```
IF <expression> THEN
   <statement> <statement>
END_IF;

IF <expression> THEN
   <statements>
ELSIF <expression> THEN
   <statements>
ELSIF <expression> THEN
   <statements>
ELSE
   <statements>
END_IF;
```

Selection: CASE statement

```
CASE <expression> OF
   <constant> : <statements>
   <constant> : <statements>
   <constant> : <statements>
ELSE
   <statements>
END_CASE;


CASE <expression> OF
<constant 1>,<constant 2>,<constant 3>:
   <statements>
<constant 1>..<constant 2>:
   <statements>
<constant 1>..<constant 2>,<constant 3>:
   <statements>
ELSE
   <statements>
END_CASE;
```

Iteration: WHILE statement

```
WHILE <expression> DO
   <statements>
END_WHILE;
```

Iteration: REPEAT statement

```
REPEAT
   <statements>
UNTIL <expression>
END_REPEAT;
```

Iteration: FOR statement

```
FOR j:=<expression> TO <expression> DO
   <statements>
END_FOR;

FOR j:=<expression> TO <expression> BY <expression> DO
   <statements>
END_FOR;
```

Iteration: EXIT

In all iteration structures (WHILE, REPEAT and FOR) the word EXIT can appear. EXIT will terminate the inner-most loop.

Ex:

```
WHILE a AND b DO
  FOR j:=1 TO 1000 DO
    IF c THEN EXIT; END_IF;
  END_FOR;
END_WHILE;
```

If the 'c' is true (c<>0), it will terminate the FOR loop but NOT out of the WHILE loop. EXIT outside iterations will create a syntax error.

RETURN

Finally, the key word 'RETURN' will be mentioned. This returns immediately from a function or a function block (please refer to the description of functions and function blocks).

Functions

A function is declared like this:

```
FUNCTION name : returntype;
   VAR_INPUT
     variable : type;
     ... ...
     variable : type;
   END_VAR;

   VAR
```

```
    variable : type;
    ... ...
    variable : type;
  END_VAR;

  <statements>
END_FUNCTION;
```

The return type has to be a simple type - such as BOOL, SINT, INT or DINT. VAR_INPUT contains the variables which have to be assigned a value outside the function. That means that these variables can't be modified inside the function but only be read. The VAR section is the private variables of the function. These can only be read/modified from within the function. The variables can be assigned an initial-value. The statements, which are in the function body, can only use the variables which are declared as input or private variables.

An example on a function which adds three numbers

```
FUNCTION Add : INT;
VAR_INPUT
   a : INT;
   b : INT;
   c : INT;
END_VAR;
   Add := a+b+c;
END_FUNCTION;
```

The assignment of the return value is done in the "Add := …." statement.
This function could be used like this:

```
j := Add(a:=3,b:=4,c:=10);
```

After this statement, the variable j will contain the value 17.

Please note how the variable of the function is explicitly assigned by writing the name of the variable. It is possible not to assign a value to the variable of a function; in this case its initial value will be used.

For example:

```
j := Add(c:=5,a:=2);
```

These call will assign the initial-value for b, that is 0, the value of j will after these call correctly be 7. Please note that the order of these assignments is arbitrary.

Parameter passing by value or by reference

In the above example, the parameters to the function were passed by *value.* It is also possible to pass the parameters to a function *by reference*.
When a parameter is passed by value, a *copy* of the parameter is made. Therefore, changes made to the formal parameter by the called function is not possible and will therefore have no effect on the corresponding actual parameter.

When a parameter is passed by reference, conceptually, the actual parameter itself is passed (and just given a new name - the name of the corresponding parameter). Therefore, any changes made to the formal parameter do affect the actual parameter. For example:

```
FUNCTION test;
VAR_INPUT
   a : ACCESS INT;
END_VAR;
   a := a + 1;
END_FUNCTION;

VAR
   x : INT := 10;
END_VAR;

test(a:=x);
```

In this example, the "test"'s parameter is passed by reference by using the ACCESS attribute. Therefore, the assignment to "a" in "test" is actually changing the variable "x" so that the value of x after the function call is 11.

For more information regarding the use of passing reference parameters, please refer to the ACCESS keyword.

Function blocks

The function block is an extremely useful structural facility when developing VPL programs. A function block that is executed can deliver one or more values to the surrounding world. By repeated execution, the delivered values will not necessarily be the same every time. This is because a function block has a state that is stored between each call. These have to be viewed in relation to a function which always generates the same value by repetition of the same call.

Please observe this very useful function block:

```
FUNCTION_BLOCK CTD;
VAR_INPUT
   cd : BOOL R_EDGE; | count down
   ld : BOOL;        | load preset value
   pv : INT;         | preset value
END_VAR;
VAR_OUTPUT
   q  : BOOL;        | count down occurred
   cv : INT;         | current value
END_VAR;

IF ld THEN
  cv := pv;
ELSIF cd
  cv := cv - 1;
END_IF;
q := cv <= 0;
END_FUNCTION_BLOCK;
```

By declaration of variable of the type BOOL in a VAR_INPUT section, it can be defined that a variable only has to be true if there is either detected a rising edge (R_EDGE) or a falling edge (F_EDGE).

The above function block is very useful for countdown applications. CTD (count down) will at each rising edge "cd" count down its current counter value. "ld" will load the preset value "pv" in "cv". Finally "q" states whether the counter has reached 0.

CTD is a brilliant example at the many advantages of using function blocks. All the logic for a CTD is contained in a single unit with a well-defined interface. A well-developed library of function blocks will make developing VPL programs much easier and it will also reduce the number of errors.

As it was with the functions, the variables in the section of VAR_INPUT can only be modified outside the function block, and the variables in the VAR_OUTPUT section can only be read outside the function and not modified. The statement in the VAR section is the private variable of the function block and can therefore only be used inside the function block.

The instantiation of a function block is done like this:

```
AAA : CTD;
```

To get access to AAA input/output variables, the following notation is used: AAA.<variable>
For example, the initial-value can be set to 1000 with the help of:

```
AAA.pv := 1000;
```

Alternatively, calling the function-block can do the assignment:

```
AAA(pv:=1000);
```

This has the same effect as making an assignment followed by a call to the function block.

An example of using the CTD:

```
VAR_INPUT
   i1  : BOOL; | input
   i2  : BOOL; | another input
END_VAR;

VAR_OUTPUT
   o   : BOOL; | output
END_VAR;

VAR
   a   : CTD;
   tmp : BOOL;
END_VAR;

PROGRAM CTD_test;

a.pv := 1000; // set preset value
BEGIN
   IF i1 THEN
   // load default value
      a(ld:=TRUE);
      a(ld:=FALSE);
   ELSE
      a(cd:=i2);
   END_IF;
   IF a.q THEN
      // count down:
      o := NOT o;
   END_IF;
```

```
END;
END_PROGRAM;
```

This was a short introduction to VPL. There are many features not described in this section, most importantly multithreading, that is also an integral part of the language.

Please also have a look at Reserved words.

3.3 Reserved words

Below is a list of all the reserved keywords in the VPL language. Some of the words are not documented but are reserved either because they are used internally or because they will be included in the VPL syntax at some later time. Please also note that there is a number of standard functions, function blocks and platform support functions that are normally included in a VPL program (when using the INCLUDE statement).

- ACCESS
- ADDR
- ALIGN
- AND
- ARRAY
- ASYNC
- AUTO
- BEGIN
- BOOL
- BREAKPOINT
- BY
- BYTE
- CALLBACK
- CASE
- CHANNEL
- CLASS
- CONFIG
- CONFIGURATION
- CONSTANT
- __DATE__
- __DEBUG__
- DINT
- DO
- DOUBLE
- (DYNAMIC)
- ELSE
- ELSIF
- END
- END_CASE
- END_FOR
- END_FUNCTION
- END_IF
- END_PROGRAM
- END_REPEAT
- END_STRUCT_BLOCK
- END_TASK
- END_THREAD_BLOCK
- END_VAR
- END_WHILE
- EXIT
- EXTCALL
- F_EDGE
- FALSE
- __FILE__
- FILE

- FLOAT
- FLOATB
- FOR
- FUNCTION
- FUNCTION_BLOCK
- HIDE
- IF
- IMAGE
- IMPLEMENTATION
- INCLUDE
- INT
- INTERFACE
- INTERVAL
- __LINE__
- __LSS__
- MANDATORY
- MOD
- __MODE_LARGE__
- __MODE_NX32__
- __MODE_X32__
- __MODE__NX32L__
- MODCALL
- MUTABLE
- MUTEX
- __NOP__
- NOT
- NOAUTOCALC
- OF
- OFF
- ON
- OR
- PRIORITY
- PROGRAM
- PTR
- R_EDGE
- REPEAT
- RETURN
- SCAN
- SEMAPHORE
- SINT
- SIZEOF
- STRING
- STRING_BEGIN
- STRING_END
- STRUCT_BLOCK
- SYSHANDLE
- TASK
- THEN
- THREAD_BLOCK
- __TIME__
- TO
- TRUE
- UINT
- UNTIL

- UPDATEIO
- UPDATEOUT
- USINT
- VAR
- VAR_INPUT
- VAR_OUTPUT
- VOICE
- WHILE
- XOR
- #DEFINE
- #UNDEFINE
- #IFDEF
- #END_IF
- #ELSE

3.4 Anatomy of a VPL program

3.4.1 Anatomy of a VPL program

Below is seen a typical VPL program. We will describe each of the sections of the program. Please note that the red numbers (**nn:**) to the left are not part of the code and are only included to make it easier to reference the individual lines in the description below.

```
1:  //-------------------------------------------------------------------------
2:  // Greenhouse_1.vpl, created 2000-12-31 14:45
3:  //
4:  //-------------------------------------------------------------------------
5:  INCLUDE rtcu.inc
6:
7:  //  Input variables that can be configured via the configuration dialog
8:  VAR_INPUT
9:
10: END_VAR;
11:
12: //  Output variables that can be configured via the configuration dialog
13: VAR_OUTPUT
14:
15: END_VAR;
16:
17: // The global variables of the program
18: VAR
19:
20: END_VAR;
21:
22: PROGRAM Greenhouse_1;
23:
24:
25: // The next code will only be executed once after the program starts
26:
27: BEGIN
28: // Code from this point until END will be executed repeatedly
29:
30:
31: END;
32:
33: END_PROGRAM;
```

Line 1..4
This is a comment. Comments can start with a "double slash" (//) and continue to the end of the line. It is also possible to use the /* to start and */ to finish the comment. This way, comments can extend over more than one line, and this form of comments are useful when excluding parts of the program code - for example during program testing.
Line 5
This is a INCLUDE statement. This is used to include other files in a program. It includes the definitions of all the built-in functions, function blocks, and Platform support functions.
Line 8
This starts the VAR_INPUT section. In this section, all variables that are inputs to the program, and that need to be able to be configured from the configuration dialog, are declared. Please note that all variables declared within this section are global and can be accessed from any section of the VPL code.
Line 10
This ends the VAR_INPUT section.
Line 13
This starts the VAR_OUTPUT section. In this section, all variables that are outputs to the program, and that need to be able to be configured from the configurationdialog are declared.

Please note that all variables declared within this section are global and can be accessed from any section of the VPL code.

Line 15

This ends the VAR_OUTPUT section.

Line 18

This starts the VAR section. In this section, all variables that are global for the program are declared.

Line 20

This ends the VAR section.

Line 22

This defines the name of the program.

Line 27

This starts the main program. The code from the BEGIN to the END statement are executed repeatedly.

Line 31

This ends the section that started with the BEGIN statement.

Line 33

This is the end of the program.

Next we will show a more complete program. This program is taken from the tutorial section of the online help manual. Following the code, an explanation of the various parts of the program are explained.

Please note again that the red numbers (**nn:**) to the left are not part of the code and are only included to make it easier to reference the individual lines in the description below.

```
1:  //-------------------------------------------------------------------------
2:  // Greenhouse_1.vpl, created 2000-12-31 14:45
3:  //
4:  //-------------------------------------------------------------------------
5:  INCLUDE rtcu.inc
6:
7:  //  Input variables that can be configured via the configuration dialog
8:  VAR_INPUT
9:     Sensor_L    : BOOL; | Sensor input for low temperature
10:    Sensor_H    : BOOL; | Sensor input for high temperature
11:    Alarm_time : INT;  | time, 1..546 minutes, time before SMS message is
sent
12:    phone_number : STRING; | The number the SMS message should be sent to
13: END_VAR;
14:
15: //  Output variables that can be configured via the configuration dialog
16: VAR_OUTPUT
17:    Out_Heater      : BOOL; | Output to activate heater
18:    Out_Ventilation : BOOL; | Output to activate ventilation
19: END_VAR;
20:
21: // The global variables of the program
22: VAR
23:    timer : TON; // ON-delay timer is declared.
24:                 // A On-delay timer goes active when 'trig' has
25:                 // been active for 'pt' seconds.
26:    send_alarm : R_TRIG; // detects leading edge on alarm
28:
29: END_VAR;
30:
31: PROGRAM Greenhouse_1;
32: // The next code will only be executed once after the program starts
33:
34: BEGIN
35: // Code from this point until END will be executed repeatedly
```

```
36:
37:     // If the temperature is to high then activate ventilation:
38:     IF Sensor_H THEN
39:        Out_Ventilation:= ON;
40:     ELSE
41:        Out_Ventilation:= OFF;
42:     END_IF;
43:
44:     // If the temperature is to low then activate the heater:
45:     IF Sensor_L THEN
46:        Out_Heater:= ON;
47:     ELSE
48:        Out_Heater:= OFF;
49:     END_IF;
50:
51:     // Start timer on the leading edge of the sensor-inputs:
52:     timer(trig:= Sensor_L OR Sensor_H);
53:
54:     // Detect leading edge on alarm:
55:     send_alarm(trig:=timer.q);
56:
57:     IF send_alarm.q THEN
58:        IF Sensor_L THEN
59:           // send sms for temperature to low
60:           gsmSendSMS(number:=phone_number, message:="Temperature to low");
61:        ELSIF Sensor_H THEN
62:           // send sms for temperature to high
63:           gsmSendSMS(number:=phone_number, message:="Temperature to high");
64:        END_IF;
65:     END_IF;
66:
67: END;
68:
69: END_PROGRAM;
```

Line 8..13
This starts the VAR_INPUT section. In this section, all variables that are inputs to the program, and that we need to be able to configure from the configurationdialog, are declared. We declare two variables of the type BOOL. They are variables that can only hold either "1" or "0" (you can use ON/OFF and TRUE/FALSE instead of 1/0). The variable "Alarm_time" is a variable of the type INT. The variable "phone_number" is a variable of the type STRING, and in this example it contains a telephone number.

Line 16..19
This starts the VAR_OUTPUT section. In this section, all variables that are outputs from the program, and that we need to be able to configure from the configurationdialog, are declared. We declare two variables of the type BOOL. They are variables that can only hold a "1" or "0" (you can use ON/OFF and TRUE/FALSE instead of 1/0).

Line 22..29
This starts the VAR section. In this section, all variables that are global in the program are declared. We declare "timer" as a variable of the type TON. TON is a function block. By writing the statement in line 23, we instantiate the function block TON and give this instantiation of the function block the name "timer". The variable "send_alarm" is an instantiation of the function block R_TRIG.

Line 38..41
This is an example of an IF statement. Line 38 is executed if the variable "Sensor_H" is active (1/ON or TRUE) otherwise line 40 is executed.

Line 52
This is a way of "connecting" the two inputs "Sensor_L" and "Sensor_H" to the trigger signal of the timer TON function block (instantiated by the "timer" variable). The "trig" input of the function block "timer" will be true in relation to either of the two variables (because of the OR operator).

Line 55
This is an example of reading a VAR_OUTPUT variable from a function block and using it as a signal for a VAR_INPUT variable of another function block.
Line 60
This is a call of a function (gsmSendSMS). When a function is called, the control is given to the function. The calling program will not gain control again before the function returns control to the caller. The arguments to a function (and a function block) can be listed in random order, and the order is not significant. When a parameter is not listed, the VAR_INPUT variable of the function / functionblock will keep its default value.

For further study, please have a look at:

- Comments
- Constants
- INCLUDE
- VAR
- VAR_INPUT
- VAR_OUTPUT
- END_VAR
- R_EDGE
- F_EDGE
- MODCALL

3.4.2 Comments

Comments are very useful when writing VPL programs, as well as when trying to read and understand VPL code written by yourself and others. Comments can occupy one line or multiple lines. When declaring variables, one of two methods can be used (see the example below). The "multiple line comment" is sometimes useful when certain parts of the program are to be disabled - possibly because one is running the program in the RTCU Emulator and does not want to test all aspects of the program in question. It can also be useful if one is looking to disable certain parts of the program that are not yet finished.

Another type of comments is comments written in the declaration of variables in the VAR_INPUT and VAR_OUTPUT sections. These comments will be shown for each variable in the configuration dialog (this is only true for the VAR_INPUT and VAR_OUTPUT sections in the main program, not for the individual function blocks !)

All comments in a VPL program are shown in grey color when viewed in the built-in syntax highlighting editor. Comments written as part of declaring variables in the VAR_INPUT and VAR_OUTPUT sections are written in orange.

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   variable_temp : INT;   | This is a comment that will be shown in the
configuration dialog
END_VAR;

PROGRAM test;

BEGIN
   ...
   // This is an example of a one-line comment that extends to the end of the
line
   ...
   /* And this is
      an example of a comment
      that extends
      more than one line
   */
   ...
END;

END_PROGRAM;
```

3.4.3 Constants

Constants can take the form of integer, floating-point of string values.
For integers it is possible to specify the value in decimal, binary, octal or hexadecimal notation. This is done by adding 2#, 8# or 16# before the actual value.

Floating-point constants are identified with a decimal '.' indicator, such as: **'55.00'** or just **'55.'** The differentiation is important as **'55'** will be treated as a integer value whether **55.0** will be treated as a floating-point value.

When defining integer or floating-point constants, it is possible to include '_' (underscore) at any position in the number to enhance readability.

Example:

```
INCLUDE rtcu.inc

VAR
   var  : INT;
   varf : FLOAT;
END_VAR;

PROGRAM test;

BEGIN
   var := 4711;  // This will assign the value 4711 to var.
   var := 47_11; // This will also assign the value 4711 to var.

   var := 2#10_0010; // This will assign the binary value 100010 to var.
(100010 equals 34 in decimal notation)
   var := 8#476;     // This will assign the octal value 476 to var.(476
equals 318 in decimal notation)
   var := 16#2F32;   // This will assign the hexadecimal value 2F32 to var.
(2F32 equals 12082 in decimal notation)
   varf := 4711.0;   // This will assign the value 4711.0 to varf.
   varf := 4_711.56; // This will assign the value 4711.56 to varf.

END;

END_PROGRAM;
```

3.4.4 INCLUDE

The INCLUDE statement is used to include another file within the current file. A good example is the rtcu.inc file which is included in all RTCU applications. The rtcu.inc file contains the declarations of all the standard functions and function blocks in the RTCU.

INCLUDE files are often used for common functionality that can be used in several projects. INCLUDE files can be managed in the project tree,but must still be manually included in the program by using the INCLUDE statement.

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
END;

END_PROGRAM;
```

3.4.5 Encrypted Include file

An encrypted include file (ENC file) is an include file (INC file) where the contents have been encrypted.
It can be used with INCLUDE in the same way that normal include files can be.
The encrypted file cannot be added to a project, loaded into the editor, or changed.

The encrypted files are used in situations where one needs to share the code with others but does not want the code to be changed.
For example, in the case one needs to share a communication protocol.

Generating an encrypted include file from an include file is done from the pop-up menu in the project control.

3.4.6 VAR

The VAR statement is used to initiate the section where a function, function block, or the main program variables can be declared.

Variables that are configurable are declared in the VAR_INPUT/VAR_OUTPUT sections.

It is allowed to have more than one VAR section within the same scope.

Example:

```
INCLUDE rtcu.inc

VAR
   variable_temp : INT;
   variable_test : DINT;
END_VAR;

PROGRAM Greenhouse_1;

BEGIN
   ...
END;

END_PROGRAM;
```

3.4.7 VAR_INPUT

The VAR_INPUT statement is used to initiate the section where a function, function block, or the main program input variables can be declared. If the VAR_INPUT section is used in the main program, variables declared within this section can be configured from the configuration dialog. Variables declared in the VAR_INPUT section can only be read from within the program and not written.

Please note that when declaring variables in this section, a | sign followed by a description is a special type of comment. Comments following this syntax will be shown as comments when the configuration dialog is invoked. This helps the person who does the configuration to remember the function of the variable being configured.

A variable in the VAR_INPUT section can have the attribute R_EDGE or F_EDGE. This enables detection of leading and falling edges on a digital input.

It is allowed to have more than one VAR_INPUT section within the same scope.

Example:

```
INCLUDE rtcuinc

VAR_INPUT
   variable_temp : INT;  | This is a variable that can be configured from the
configuration dialog
   variable_test : DINT; | This is another variable that can be configured
from the configuration dialog
END_VAR;

PROGRAM Greenhouse_1;

BEGIN
   ...
END;

END_PROGRAM;
```

3.4.8 VAR_OUTPUT

The VAR_OUTPUT statement is used to initiate the section where a function block, thread block, or the main program output variables can be declared. If the VAR_OUTPUT section is used in the main program, variables declared within this section can be configured from the configuration dialog. Variables declared in the VAR_OUTPUT section can only be written from within the program and not read.

It is allowed to have more than one VAR_OUTPUT section within the same scope.

Example:

```
INCLUDE rtcu.inc

VAR_OUTPUT
   variable_temp : INT;  | This is a variable that can be configured from the
configuration dialog
   variable_test : DINT; | This is another variable that can be configured
from the configuration dialog
END_VAR;

PROGRAM Greenhouse_1;

BEGIN
   ...
END;

END_PROGRAM;
```

3.4.9 END_VAR

The END_VAR statement ends a section that was initiated with the VAR or VAR_INPUT / VAR_OUTPUT statements.

For an example, look at the VAR or VAR_INPUT / VAR_OUTPUT statements.

3.4.10 R_EDGE (Attribute)

R_EDGE is an attribute that can be used on the BOOL variables that are declared in a VAR_INPUT section. By using this attribute, the variable it is used on will only be TRUE when there is a rising edge detected on the input value.

Please also have a look at F_EDGE for a falling edge attribute.

All variables with the R_EDGE attribute will only be updated when BEGIN or UPDATEIO() is executed.

Example:

For an example of the R_EDGE, please have a look at the CTD function block.

3.4.11 F_EDGE (Attribute)

F_EDGE is an attribute that can be used on the BOOL variables that are declared in a VAR_INPUT section. By using this attribute, the variable it is used on will only be TRUE when there is a falling edge detected on the input value.

Please also have a look at R_EDGE for a rising edge attribute.

All variables with the F_EDGE attribute will only be updated when BEGIN or UPDATEIO() is executed.

Example:

For an example of the R_EDGE (which is similar to F_EDGE), please have a look at the CTD function block.

3.4.12 MODCALL

MODCALL is used for calling external C functions that have been developed by the **RTCU Platform SDK**.
The parameters to MODCALL are as follows:

- The module name.
- The symbolic name of the C-function.
- Result code variable (INT)
  - 0 = Function was successfully invoked.
  - 1 = The symbolic function name was not found.
  - 2 = Invalid module and/or function name.
  - 3 = Not supported.

Before using an external module it must be loaded with the extModuleLoad function.
Also refer to the Platform Extension functions section.

The detailed semantics of MODCALL and the corresponding parameters are described in the **RTCU Platform SDK**.

Example:

```
FUNCTION isPrime : BOOL;
VAR
  x : DINT;
  rc : INT;
END_VAR;

isPrime := MODCALL("my_calc","is_prime_number",rc); // Call the
'is_prime_number' function in the 'my_calc' module.
                                                    // rc will contain 0 when
the function was successfully invoked.

END_FUNCTION;
```

3.5 Program Flow Control

3.5.1 Program Flow Control

The following pages describe the various elements of the VPL programming language that control the flow of a program. The elements are divided into two groups - that is selection and iteration elements.

VPL offers the following language elements for conditional execution of code:

- IF-THEN-ELSE
- CASE-OF

VPL offers the following language elements for iterative execution of code:

- WHILE-DO
- FOR-TO-DO
- REPEAT-UNTIL
- BEGIN-END

Keywords used in combination with the above:

- RETURN
- EXIT
- PROGRAM
- UPDATEIO
- UPDATEOUT

3.5.2 IF

IF statements are used for conditional execution of code in VPL programs.

```
IF <expr> THEN
  statement;
ELSIF <expr> THEN
  statement;
ELSE
  statement;
END_IF;
```

<expr> is an expression that evaluates to a BOOL, and it is therefore always either TRUE or FALSE. Based on the <expr>, the code after the THEN keyword is either executed or bypassed.

Example:

```
INCLUDE rtcu.inc

VAR
   a   : INT;
   b   : INT;
   str : STRING;
END_VAR;

PROGRAM test;

BEGIN

   IF a > b THEN
      str := "a is grater than b";
   ELSIF a = b THEN
      str := "a equals b";
   ELSE
      str := "a is lesser than b";
   END_IF;

END;

END_PROGRAM;
```

3.5.3 THEN

The word THEN is part of the IF-THEN statement.

It can also be used in #IFDEF conditional compilation directive.

3.5.4 ELSE

The word ELSE can be part of either

- IF-THEN statement
- CASE statement

It can also be used in #IFDEF conditional compilation directive.

3.5.5 ELSE, (IF Statement)

ELSE is part of the IF statement.

3.5.6 ELSIF

ELSIF is part of the IF statement.

3.5.7 END_IF

END_IF is part of the IF statement.

3.5.8 RETURN

By using RETURN, it is possible to return from a function or function block before the END statement of that function/function block has been reached.

3.5.9 BEGIN

BEGIN starts a section of code that will be executed repeatedly. The code between BEGIN and END will be executed until an EXIT statement is reached. If no EXIT statement is reached, the code will keep executing indefinitely. During each iteration, a call to UPDATEIO will be done to update all physical in- and outputs. Typically in a VPL program, one main loop will keep executing indefinitely and make the necessary calls to functions and function blocks. The BEGIN/END construction is only allowed in the program section.

Example:

```
INCLUDE rtcu.inc

VAR
   a   : INT;
   b   : INT;
   str : STRING;
END_VAR;

PROGRAM test;

// Code from this point to END are executed forever
BEGIN

   IF a > b THEN
      str := "a is grater than b";
   ELSIF a = b THEN
      str := "a equals b";
   ELSE
      str := "a is lesser than b";
   END_IF;

END;

END_PROGRAM;
```

3.5.10 END

END is used in conjunction with BEGIN.

3.5.11 CASE

CASE statements are used for conditional execution of code in VPL programs.

```
CASE <expression> OF
<constant 1>,<constant 2>,<constant 3>:
   <statements>
<constant 1>..<constant 2>:
   <statements>
<constant 1>..<constant 2>,<constant 3>:
   <statements>
ELSE
   <statements>
END_CASE;
```

<expr> is an expression that evaluates to a number. If one of the <num> values has the same value as <expr>, that section of code will be executed.
If none of the <num> values match the <expr> value, the statements after the ELSE word are executed, and if no ELSE statement is found, the code after the END_CASE word will be executed.
The <expression> is limited to variables of SINT and INT types.

Example:

```
INCLUDE rtcu.inc

VAR
   a   : INT;
   str : STRING;
END_VAR;

PROGRAM test;

BEGIN

   CASE a OF
      -1    : str := "a is minus one";
      1     : str := "a is one";
      2..6  : str := "a is between two and six";
      7,9   : str := "a is either seven or nine";
   ELSE
      str := "a is something else...";
   END_CASE;

END;

END_PROGRAM;
```

3.5.12 OF

OF is used in either:

- ARRAY
- CASE statement

3.5.13 ELSE, (CASE Statement)

ELSE is part of the CASE statement.

3.5.14 END_CASE

END_CASE is part of the CASE statement.

3.5.15 WHILE

WHILE statements are used for repetitive conditional execution of code in VPL programs.

```
WHILE <expr> DO
  statement;
END_WHILE;
```

<expr> is an expression that evaluates to a BOOL. As long as <expr> evaluates to TRUE, the code until END_WHILE will be executed repeatedly. The WHILE loop can be terminated by using the EXIT statement. Please note that unlike a REPEAT statement, the <expr> is evaluated before the code is executed, and if the <expr> is FALSE, the code between the WHILE and END_WHILE will not be executed.

Example:

```
INCLUDE rtcu.inc

VAR
   a   : INT;
   str : STRING;
END_VAR;

PROGRAM test;

BEGIN
   a := 0;
   WHILE a < 10 DO
      a := a + 1;
   END_WHILE;
   // At this point a is 10

END;

END_PROGRAM;
```

3.5.16 DO

DO is used in both:

- FOR statement
- WHILE statement

3.5.17 END_WHILE

END_WHILE is part of the WHILE statement.

3.5.18 EXIT

EXIT is used to terminate the execution of the following interative control structures:

- WHILE-DO
- FOR-TO-DO
- REPEAT-UNTIL

Calling EXIT will resume execution immediately after the corresponding END_WHILE, END_FOR, or END_REPEAT statements.

3.5.19 FOR

FOR statements are used for iteration of code in VPL programs. The code between the FOR and END_FOR will be executed a number of times.

```
FOR <variable>:=<n> TO <m> BY <k> DO
   statement;
END_FOR;
```

The <variable> above is assigned the value n, and the statement is then executed. <variable> will the be incremented by the value <k>, and if <variable> is lesser than <m>, the statement is executed again, and so on. If the BY <k> is omitted, the <variable> is incremented by 1 each time.

Example:

```
INCLUDE rtcu.inc

VAR
   a : INT;
END_VAR;

PROGRAM test;

BEGIN

   FOR a := 1 TO 100 BY 10 DO
      // a will get the values 10,20,30,40,50,60,70,80,90,100
      ...
   END_FOR;

END;

END_PROGRAM;
```

3.5.20 TO

TO is part of the FOR-TO-DO statement.

3.5.21 BY

BY is part of the FOR statement.

3.5.22 END_FOR

END_FOR is part of the FOR statement.

3.5.23 PROGRAM

PROGRAM names, as well as encapsulates, the VPL main program together with END_PROGRAM.

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
END;

END_PROGRAM;
```

3.5.24 END_PROGRAM

END_PROGRAM encapsulates the main VPL program together with PROGRAM,

3.5.25 UPDATEIO

UPDATEIO will force an update of all in- and outputs on the RTCU platform. Normally UPDATEIO is not needed as the BEGIN/END construction will handle the updating of the in- and outputs. But in some cases, it can be necessary to utilise the UPDATEIO manually in a program. For example in a REPEAT statement, it could be necessary to update all in- and outputs in each iteration. UPDATEIO will update first the inputs, then the outputs, in that order.

Example:

```
INCLUDE rtcu.inc

VAR_OUTPUT
   output : ARRAY[1..16] OF BOOL;| Our 16 digital outputs
END_VAR;

VAR
   a      : INT;
END_VAR;

PROGRAM test;

BEGIN
   a := 1;

   REPEAT
      output[a] := TRUE; // Set the output TRUE
      UPDATEIO; // Update all in- and outputs
      Sleep(delay := 100); // Wait 100 milliseconds

      output[a] := FALSE; // Set the output FALSE
      UPDATEIO; // Update all in- and outputs
      Sleep(delay := 100); // Wait 100 milliseconds

      a := a + 1; // Next output
   UNTIL a > 16
   END_REPEAT;
END;

END_PROGRAM;
```

This example will make a "running" light on the 16 outputs. Output 1 will be on for 100 ms, then output 2 will be on for 100 ms, and so on.

3.5.26 UPDATEOUT

UPDATEOUT will force an update of all outputs on the RTCU platform. Normally UPDATEOUT is not needed as the BEGIN/END construction will handle the updating of the outputs. But in some cases, it can be necessary to utilise the UPDATEOUT manually in a program. For example in a REPEAT statement, it could be necessary to update all outputs in each iteration.

Example:

```
INCLUDE rtcu.inc

VAR_OUTPUT
   output : ARRAY[1..16] OF BOOL;| Our 16 digital outputs
END_VAR;

VAR
   a      : INT;
END_VAR;

PROGRAM test;

BEGIN
   a := 1;

   REPEAT
      output[a] := TRUE; // Set the output TRUE
      UPDATEIO; // Update all in- and outputs
      Sleep(delay := 100); // Wait 100 milliseconds

      output[a] := FALSE; // Set the output FALSE
      UPDATEOUT; // Update all outputs
      Sleep(delay := 100); // Wait 100 milliseconds

      a := a + 1; // Next output
   UNTIL a > 16
   END_REPEAT;

END;

END_PROGRAM;
```

This example will make a "running" light on the 16 outputs. Output 1 will be on for 100 ms, then output 2 will be on for 100 ms, and so on.

3.5.27 REPEAT

REPEAT statements are used for repetitive conditional execution of code in VPL programs.

```
REPEAT
  statement;
UNTIL <expr>
END_REPEAT;
```

<expr> is an expression that evaluates to a BOOL. As long as <expr> evaluates to TRUE, the code until UNTIL will be executed repeatedly. The REPEAT loop can be terminated by using the EXIT statement. Please note that unlike a WHILE statement, the <expr> is evaluated after the code is executed, and if the <expr> is FALSE, the code between the REPEAT and UNTIL will at the very least be executed once.

Example:

```
INCLUDE rtcu.inc

VAR
   a   : INT;
END_VAR;

PROGRAM test;

BEGIN
   a := 0; // Even if we assign 200 to 'a' at this point, the statements
           // between REPEAT and UNTIL will be executed once as the test
           // is done AFTER the first execution of the statements !
   REPEAT
      a := a + 10;
   UNTIL a > 100
   END_REPEAT;
END;

END_PROGRAM;
```

3.5.28 UNTIL

UNTIL is part of the REPEAT statement.

3.5.29 END_REPEAT

END_REPEAT is part of the REPEAT statement.

3.6 Datatypes

3.6.1 Datatypes and variables

Variables

Variables in a VPL program are used as placeholders for values. A variable can hold some kind of value. The type of value it can hold depends on the type of the variable. In VPL a number of data types exists. Various types are used for storing integer numbers, floating-point numbers and a type exists for storing text strings. It is even possible to make arrays of the simple data types - more on that in the ARRAY section.

When a program has variables of different types (which is usually the case), it is possible to use typecast to convert between the different data types. Please also see the section **Typecast** below for more information how to case between the various data types.

Datatypes

The VPL programming language offers a number of different data types.
The different data types are able to hold different types of data - be that integer numbers, floating-point numbers or strings. In a program, it is always best to try to select the smallest possible data type that is able to handle the given values. This will help reduce the space needed for the program. If, for example, one needs a variable with a value between 1 and 10, it is sufficient to declare a variable as the type SINT. On the other hand if mathematical expressions are to be handled a FLOAT needs to be used.

The complete list of the native data types are:

Type	Description	Value range	Size
BOOL	Boolean for storing true/false	TRUE or FALSE	1 byte
BYTE	Small unsigned integer value	0..255	1 byte
USINT	Small unsigned integer value	0..255	1 byte
SINT	Small signed integer value	-128 .. 127	1 byte
UINT	Standard unsigned integer value	0..65535	2 bytes
INT	Standard signed integer value	-32768 .. 32767	2 bytes
DINT	Large integer value	-2147483648 ,, 2147483647	4 bytes
FLOAT	Single precision floating-point value	±1.18E-38 .. ±3.4E38	4 bytes
DOUBLE	Double precision floating-point value	±2.23E-308 .. ±1.8E308	8 bytes
STRING	String for storing characters	All characters, except 0.	2 bytes for the handle
PTR	Anonymous address	n/a	4 bytes
CHANNEL	Handle to a channel	n/a	2 bytes
SEMAPHORE	Handle to a semaphore	n/a	2 bytes
MUTEX	Handle to a mutex	n/a	2 bytes
FILE	Handle to a file	n/a	2 bytes
SYSHANDLE	Generic system handle	n/a	4 bytes
CALLBACK	Address of a callback function	n/a	4 bytes

For declaring arrays of data, please have a look at:

- ARRAY This declares an ARRAY of a specific data type.

Typecast

Variables of different types cannot be assigned to one another when the destination is of different data type than the source. The VPL language contains a strong type-checking mechanisms, preventing errors when developing programs and the programmer is notified by syntax errors when trying to assign expressions to variables that are not of the correct type. In some cases, however, the programmer needs to assign expressions of the "wrong" type to variables, and because of that, typecasts are part of the VPL language. Typecasts are used to convert one type of expression to another type - for example if one wishes to convert a number to a BOOL variable. The BOOL type can only contain the values TRUE or FALSE, and it is therefore not possible to try to assign the value 1 or 0 (or any other number) to a variable of the type BOOL. If there is a need for such things, it is possible to use typecasting. The typecast is performed by writing the following:

```
var := TYPE(expression);
```

Here "var" is the destination variable, "TYPE" is one of the built-in data types, and "expression" is the expression that needs to be converted.

Example:

```
INCLUDE rtcu.inc

VAR
   var_bool : BOOL;
   var_int  : INT;
   var_dint : DINT;
END_VAR;

PROGRAM test;

BEGIN
   var_int  := 1;
   var_bool := BOOL(var_int);   // This makes a typecats on the value 1 (an
integer) and assigns it to var (the same as TRUE/ON)
   var_int  := INT(var_dint);   // This makes a typecats on the var_dint to a
int type
   var_dint := var_int;         // This is allowed, as a INT is a "smaller"
data type than DINT is
END;

END_PROGRAM;
```

3.6.2 BYTE

BYTE is one of the basic data types. A BYTE can hold integer values in the range 0 to 255. A BYTE behaves exactly as an USINT and can be considered an alias for the same.

An BYTE data type occupies 1 byte of storage.

3.6.3 USINT

USINT is one of the basic data types. A USINT can hold integer values in the range 0 to 255.

An USINT data type occupies 1 byte of storage.

3.6.4 SINT

SINT is one of the basic data types. A SINT can hold integer values in the range -128 and 127.

A SINT data type occupies 1 byte of storage.

3.6.5 UINT

UINT is one of the basic data types. A UINT can hold integer values in the range 0 to 65535.

An UINT data type occupies 2 bytes of storage.

3.6.6 INT

INT is one of the basic data types. An INT can hold integer values in the range -32768 and 32767.

An INT data type occupies 2 bytes of storage.

3.6.7 DINT

DINT is one of the basic data types. A DINT can hold integer values in the range -2147483648 to 2147483647.

A DINT data type occupies 4 bytes of storage.

3.6.8 FLOAT

FLOAT is one of the basic data types supported by the VPL programming language from firmware version 3.10.

A FLOAT can hold single-precision floating-point values in the range ±1.18E-38 .. ±3.4E38 with a precision of approx. 7 decimal digits. Additionally, the denormalized numbers in the range ±1.4E-45 .. ±1.18E-38 can be held with reduced precision.
A FLOAT data type occupies 4 bytes of storage.

The FLOAT data type is based on the IEEE 754 standard, and can – in addition to the floating-point values – contain two kinds of error values, **±infinity** and **NaN** (Not a Number).
These error values will be returned from some math functions and calculations if the result is out of range, or if the calculation is invalid, e.g. due to division by zero.
If a sub-calculation returns an error value, the entire calculation will return an error value, which can be checked with IsInf and IsNaN.

It is possible to transparently mix FLOAT and DOUBLE without explicit type-conversion.

3.6.9 DOUBLE

DOUBLE is one of the basic data types supported by the VPL programming language from firmware version 5.00 / R1.30.00.

A DOUBLE can hold double-precision floating-point values in the range ±2.23E-308 .. ±1.8E308 with a precision of approx. 16 decimal digits. A DOUBLE data type occupies 8 bytes of storage.

The DOUBLE data type is based on the IEEE 754 standard, and can – in addition to the floating-point values – contain two kinds of error values, **±infinity** and **NaN** (Not a Number).
These error values will be returned from some math functions and calculations if the result is out of range, or if the calculation is invalid, e.g. due to division by zero.
If a sub-calculation returns an error value, the entire calculation will return an error value, which can be checked with IsInf and IsNaN.

It is possible to transparently mix DOUBLE and FLOAT without explicit type-conversion. Loss of precision will occur when assigning a DOUBLE to a FLOAT variable.

Any constant floating point value will be handled as a single precision FLOAT.

3.6.10 BOOL

BOOL is one of the basic data types. A BOOL can only take two values - either FALSE or TRUE. BOOL is often used as a data type for Digital I/O's because a Digital I/O can be represented by either FALSE (not active) or TRUE (active).

A BOOL data type occupies 1 byte of storage.

For assigning values to a BOOL type, please see:

- TRUE, ON
- FALSE, OFF

or

- Datatypes and Typecasting

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   I1 : BOOL; | Input signal
END_VAR;

VAR_OUTPUT
   O1 : BOOL; | Output signal
END_VAR;

PROGRAM test;

BEGIN
   IF I1 THEN
      O1 := TRUE;
   ELSE
      O1 := FALSE;
   END_IF;
END;

END_PROGRAM;
```

3.6.11 STRING

STRING is one of the basic data types. A variable of the type STRING is used for processing strings of text for example when sending messages over SMS or on a display.

A STRING data type occupies 2 bytes of storage because the actual string is handled separately and is subject to automatic garbage collection.

The maximum length of a string is 16384 characters when **Large String Support (LSS)** has been enabled in the project settings. Without LSS the maximum size is just 254 characters. Internally, strings are zero-terminated and therefore the string itself cannot contain the character 0 (zero).

In order to include special characters in strings, it is possible to use the following control character strings:

```
$$  --> $
$"  --> "
$L  --> linefeed (value 10)
$P  --> formfeed (value 12)
$R  --> carriage return (value 13)
$T  --> Tab (value 9)
$N  --> Newline (value 13, value 10)
$xx --> Value xx (xx in hex, 00..FF)
```

Another option is to define the strings using STRING_BEGIN and STRING_END, as special characters does not need to be escaped in raw strings.
When assigning a variable of the type STRING to another variable of the same type, the contents are not copied. The assignment only has the effect of making the two variables point to the same string.

Note: A newline in the string follows the Windows convention and is composed of an CR (value 13) followed by LF (value 10).

Dynamic strings:

A dynamic string is a string that must be created and stored in RAM at run time.

If you have declared:

```
A : STRING;
```

and you assign:

```
A:="Hello world";
```

this will not be a dynamic string but a static string and will not occupy any space in RAM.

If you assign:

```
A:=strConcat(str1:="Hello ",str2:="world");
```

it will be a dynamic string that will occupy the string-length + 1. In this case 12 bytes.

Please see the section on Operators for more information.

The following resource limitations apply:

- X32 Execution Architecture: Maximum 400 dynamic strings with a total size of 62 Kbytes.
- NX32 Execution Architecture: Maximum 600 dynamic strings with a total size of 125 Kbytes.
- NX32L Execution Architecture: Maximum 4096 dynamic strings with a total size of 512 Kbytes.

Important note regarding the use of strings with multithreading:
Assigning a dynamic string to a global (shared) STRING variable from multiple threads is not thread-safe. An example is assigning a global string to a dynamic string created, for example, by strConcat() or the '+' operator. A MUTEX can be used to protect the operation in this special case.
Failure to follow these guidelines can result in a fault code 6 (Illegal string-id referenced). Please see the section on fault codes.

Example:

```
INCLUDE rtcu.inc

VAR
   text_to_send : STRING := "This is the message";
   str          : STRING;
   str2         : STRING;
END_VAR;

PROGRAM test;

BEGIN
   ...
   gsmSendSMS(number := "+44 12 34 56 789", message := text_to_send);
   str  := "This is a test $NThis is placed on a newline";
   str2 := "Hello" + " " + "World!";  // Result: "Hello World!".
   ...
END;

END_PROGRAM;
```

3.6.12 STRING_BEGIN

STRING_BEGIN is used to create raw strings. Raw strings still use the STRING data type, but are defined by surrounding the text with STRING_BEGIN and STRING_END instead of double quotes. This makes it possible to use special characters without having to escape them.

A raw string will consist of the contents of every line after the line with STRING_BEGIN, until it finds a line that starts with STRING_END.
It does not include the final linebreak before STRING_END.

No further text is allowed after STRING_BEGIN.

Note: A newline in the string follows the Windows convention and is composed of an CR (value 13) followed by LF (value 10).

Example:

```
INCLUDE rtcu.inc

VAR
   str1         : STRING := STRING_BEGIN
This is the contents of the first string.
It can include both " and $ with no escaping.
To end the string, place STRING_END at the start of a new line as such:
STRING_END;
   str2         : STRING;
   str3         : STRING;
   str4         : STRING;
END_VAR;

PROGRAM test;

   str2 := STRING_BEGIN
Raw strings can be concatenated like any other strings:
"
STRING_END + str1 + "$"";

   str3 :=
STRING_BEGIN
STRING_BEGIN does not have to be on the same line as the assignment.
This can sometimes help with the readability.
To end the string, STRING_END *must* be at the start of the line.
 STRING_END does not work, as there is a space before it.
STRING_END;

   str4 :=
string_begin
STRING_BEGIN and STRING_END do not have to use upper case letters.
sTrInG_eNd;


BEGIN

END;

END_PROGRAM;
```

3.6.13 STRING_END

This ends the raw string started with STRING_BEGIN.

3.6.14 ARRAY

An array is used to "group" a number of data elements together so that it can be referenced by an index.

An array can also be initialized - the exception being arrays declared in VAR_INPUT / VAR_OUTPUT or in a FUNCTION.
In case an array is not fully initialized, the remaining elements in the array will simply be set to the default value 0 - FALSE by the compiler.

Only one-dimensional arrays are supported with a maximum number of elements defined by the execution architecture. Under X32/NX32 the maximum number of elements is 65535 and the maximum size of each element is also 65535.
Under the NX32L execution there is an unlimited number of elements with an unlimited size of each element.

The actual number and size of the defined arrays are only limited by the available memory.

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   Input : ARRAY[1..8] OF BOOL; | The 8 input signals
END_VAR;

VAR_OUTPUT
   Output : ARRAY[1..8] OF BOOL; | The 8 output signals
END_VAR;

VAR
   index  : SINT;
   work1  : ARRAY[2..5] OF INT := 123,678,345,678;
   work2  : ARRAY[1..5] OF STRING := "This", "Is", "a", "Big", "Test";
END_VAR;

PROGRAM test;

BEGIN

   // Copy each of the 8 inputs to the outputs
   FOR index:=1 TO 8 DO
      Output[index] := Input[index];
   END_FOR;
END;

END_PROGRAM;
```

3.6.15 STRUCT_BLOCK

A STRUCT_BLOCK is a user-defined composite type. Struct blocks are an extremely useful facility to group variables as in this example:

```
STRUCT_BLOCK sbBufferPDU
   length : INT;
   data   : ARRAY[1..140} OF SINT;
END_STRUCT_BLOCK;
```

The biggest advantage of the STRUCT_BLOCK is found when writing data of different types to the serial port, PDU message, or file system.
Instead of writing the data types individually, they can be grouped in a STRUCT_BLOCK, and the device sees it as one variable.

```
STRUCT_BLOCK sbGpsData
   mode         : SINT;
   linsec       : DINT;
   latitude     : DINT;
   longitude    : DINT;
   speed        : DINT;
   course       : DINT;
   height       : INT;
   PDOP         : INT;
   HDOP         : INT;
   VDOP         : INT;
   inview       : SINT;
   used         : SINT;
END_STRUCT_BLOCK;

VAR
   gpsData : sbGpsData;
END_VAR;
```

To access the members of a STRUCT_BLOCK, the same syntax as used with a FUNCTION_BLOCK is used.
For example:

```
gpsData.speed := 100;
```

To send the data through a serial channel:

```
serSendData(port:=1, data:=ADDR(gpsData), size:=SIZEOF(gpsData));
```

Note: Care must be taken when working with a STRING inside a STRUCT_BLOCK as the actual string is not contained in the variable (see string).

3.6.16 END_STRUCT_BLOCK

This ends a STRUCT_BLOCK.

3.6.17 SYSHANDLE

The SYSHANDLE datatype is an abstract context-specific unique identifier that is used to safely reference and access system resource such as components in the GUI interface.
The SYSHANDLE is an opaque and general handle type in contrast to context specific handles offered by the SEMAPHORE, CHANNEL, MUTEX and FILE datatypes.

A SYSHANDLE has the following attributes:

- Occupies 4 bytes of storage.
- Is only assignment compatible with itself.
- Comparison operators are limited to the equal(=) or not-equal(<>) operators.
- Type cast is only possible to BOOL where only a valid handle yields TRUE.

3.6.18 PTR

The pointer data type holds a pointer - i.e. an address.

A PTR data type occupies 4 bytes of storage.

3.6.19 CHANNEL

The CHANNEL data type is used for passing data between threads.
Please see the section about the Channel functions for more information.

A CHANNEL data type occupies 2 bytes of storage.

Please also see the section on Thread synchronization.

3.6.20 SEMAPHORE

The SEMAPHORE data type is used for various thread-synchronization purposes. Please see the section about the Semaphore functions for more information.

A SEMAPHORE data type occupies 2 byte of storage.

Please also see the section on Thread synchronization.

3.6.21 MUTEX

The MUTEX data type is used for various thread-synchronization purposes.
Please see the section about the Mutex functions for more information.

A MUTEX data type occupies 2 bytes of storage.

Please also see the section on Thread synchronization.

3.6.22 FILE

The FILE data type is used to represent open files in the file system. When a file is created or opened, an ID (FILE) is returned, and this ID is used in all file operations (Read, Write etc.)

Please see the section about the file system for more information.

A FILE data type occupies 2 bytes of storage.

3.6.23 CALLBACK

The CALLBACK data type is used when working with callback functions.
Please refer to the function attribute CALLBACK for more details.

3.7 Predefined constants

3.7.1 Predefined constants

A number of constants are defined in VPL:

• TRUE, ON	This can be assigned to a variable of the type BOOL.
• FALSE, OFF	This can be assigned to a variable of the type BOOL.
• __LINE__	Returns the current line number as an integer of the current source file.
• __FILE__	Returns the file-name as a STRING of the current file.
• __TIME__	Returns the compilation time as a STRING.
• __DATE__	Returns the compilation date as a STRING.
• __DEBUG__	Defined when compiled with the debug option enabled.
• __LSS__	Defined when compiled with the LSS option enabled.
• __MODE_LARGE__	Defined when compiled as LARGE architecture.
• __MODE_X32__	Defined when compiled as X32 architecture.
• __MODE_NX32__	Defined when compiled as NX32 architecture.
• __MODE_NX32L__	Defined when compiled as NX32L architecture.

3.7.2 TRUE, ON

TRUE and ON are both synonymous for the same thing. They are used when assigning a constant to a variable of the type BOOL.

Example:

```
INCLUDE rtcu.inc

VAR
   var : BOOL;
END_VAR;

PROGRAM test;

BEGIN
   var := TRUE; // This sets the variable var to TRUE (the same as setting it
to ON)
   var := ON;   // This sets the variable var to ON (the same as setting it to
TRUE)

   var := BOOL(1); // This makes a typecast on the value 1 (an integer) and
assigns it to var (the same as TRUE/ON)
END;

END_PROGRAM;
```

3.7.3 FALSE, OFF

FALSE and OFF are both synonymous for the same thing. They are used when assigning a constant to a variable of the type BOOL.

Example:

```
INCLUDE rtcu.inc

VAR
   var : BOOL;
END_VAR;

PROGRAM test;

BEGIN
   var := FALSE; // This sets the variable var to FALSE (the same as setting
it to OFF)
   var := OFF;   // This sets the variable var to OFF (the same as setting it
to FALSE)

   var := BOOL(0); // This makes a typecast on the value 0 (an integer) and
assigns it to var (the same as FALSE/OFF)
END;

END_PROGRAM;
```

3.7.4 __LINE__

__LINE__ returns the current line number in the current source file as an integer. This is especially useful for debugging purposes.
Also see __FILE__.

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Show which line is being executed (this will show that line 7 is the
current line)
debugFmt(message := "Current line is \1", v1:=__LINE__);

END_PROGRAM;
```

3.7.5 __FILE__

__FILE__ returns a STRING with the name of the current source file. This is especially useful for debugging purposes.
Also see __LINE__.

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Show the name of the file that is being executed:
DebugMsg(message := "Current file-name is "+__FILE__);

END_PROGRAM;
```

3.7.6 __TIME__

__TIME__ returns a STRING with the system time at the time of compilation. This is especially useful for debugging purposes.
Also see __DATE__.

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Show the compilation time/date:
DebugMsg(message := "The compilation time is "+__TIME__);
DebugMsg(message := "The compilation date is "+__DATE__);


END_PROGRAM;
```

3.7.7 __DATE__

__DATE__ returns a STRING with the system date at the time of compilation. This is especially useful for debugging purposes.
Also see __TIME__.

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Show the compilation time/date:
DebugMsg(message := "The compilation time is "+__TIME__);
DebugMsg(message := "The compilation date is "+__DATE__);


END_PROGRAM;
```

3.7.8 __DEBUG__

__DEBUG__ is defined when debug is enabled for the project. See Project: Settings.

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

#IFDEF __DEBUG__ THEN
   DebugMsg(message := "Debug is enabled.");
#ELSE
   DebugMsg(message := "Debug is not enabled.");
#END_IF

END_PROGRAM;
```

3.7.9 __LSS__

__LSS__ is defined when Large String Support is enabled for the project. See Project: Settings.

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

#IFDEF __LSS__ THEN
   DebugMsg(message := "Large String Support is enabled.");
#ELSE
   DebugMsg(message := "Large String Support is not enabled.");
#END_IF

END_PROGRAM;
```

3.7.10 __MODE_LARGE__

__MODE_LARGE__ is defined when the project is compiled for the LARGE architecture. See Project: Settings.

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

#IFDEF __MODE_LARGE__ THEN
   DebugMsg(message := "Compilation architecture is Large");
#END_IF

END_PROGRAM;
```

3.7.11 __MODE_X32__

__MODE_X32__ is defined when the project is compiled for the X32 architecture. See Project: Settings.

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

#IFDEF __MODE_X32__ THEN
   DebugMsg(message := "Compilation architecture is X32");
#END_IF

END_PROGRAM;
```

3.7.12 __MODE_NX32__

__MODE_NX32__ is defined when the project is compiled for the NX32 architecture. See Project: Settings.

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

#IFDEF __MODE_NX32__ THEN
   DebugMsg(message := "Compilation architecture is NX32");
#END_IF

END_PROGRAM;
```

3.7.13 __MODE_NX32L___

__MODE_NX32L__ is defined when the project is compiled for the NX32L architecture. See Project: Settings.

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

#IFDEF __MODE_NX32L__ THEN
   DebugMsg(message := "Compilation architecture is NX32L");
#END_IF

END_PROGRAM;
```

3.8 Functions and Functionblocks

3.8.1 Functions and Functionblocks

Functions and function blocks are two very important ingredients in a typical VPL program.

Functions and function blocks makes it possible to structure a program in a way that makes it much more reader-friendly and comprehensible. They also enable one very important thing - namely reusability. By using functions and function blocks, it is possible to create general functions that can be reused in many different programs. It is always a good idea to try to make general and well-documented functions as this will greatly improve the usability.

On the RTCU platform, whenever one accesses the cellular module, uses a timer, or prints on the display, the tool that makes this possible to do in a simple way is either a function or a function block.

For details on functions and function blocks, please have a look at the following:

- FUNCTION
- END_FUNCTION
- ACCESS
- MANDATORY
- FUNCTION_BLOCK
- END_FUNCTION_BLOCK
- INTERFACE
- IMPLEMENTATION
- ASYNC
- ALIGN
- MUTABLE
- CALLBACK

3.8.2 FUNCTION

Functions are one of the important aspects of VPL programming. By using functions (and function blocks), it is possible to isolate often used pieces of code in small enclosed units. Reuse of previously developed functions enables one to develop programs faster and with far less errors.

A function can be declared like this:

```
FUNCTION name : returntype;

 VAR_INPUT
    variable : type;
    ...
    variable : type;
 END_VAR;

 VAR
    variable : type;
    ...
    variable : type;
 END_VAR;
 ...
 END_FUNCTION;
```

The return type must be one of the built-in data types of VPL. The VAR_INPUT section declares all the variables that can be assigned a value outside of the function (accomplished by the calling program). These variables cannot be modified from within the function but only read. The VAR section contains private variables of the function. The variables that are declared within the VAR section can only be seen from within the function itself. The variables can be assigned an initial value using the ':=' notation.

Below is an example of a simple function that can add 3 numbers together and return the result:

```
FUNCTION Add : INT;

 VAR_INPUT
     a : INT;
     b : INT;
     c : INT;
 END_VAR;

     Add := a+b+c; // Make the our function return the sum of the 3 numbers

 END_FUNCTION;
```

The return value of the function will be the value that is assigned to the name of the function as shown above.

The above function can then be used in a program in the following way:

```
j := Add(a:=3, b:=4, c:=10);
```

After this call, j will have the value 17.

Please note how the variables of the function is assigned values by stating their names. If you omit one or more of the variables when calling the function, the initial value of the variables (assigned in the functions VAR_INPUT section with the := notation) will be used instead. If no

initial value is given for the function, the value of 0 will be used instead (for a STRING an empty string will be used instead).

An example:

```
j := Add(c:=5,a:=2);
```

This call will use the initial value for b (in this case 0). Therefore, the return value of the call (which will be assigned to j) will be 7.

Please also note that it does not matter where in the list each of the variables is mentioned. This is one of the benefits of using this notation. It is possible to assign the variables in any desired order.

The standard parameter passing method is by value. When a parameter is passed by value, a copy of the parameter is made. Therefore, changes made to the formal parameter by the called function have an effect on the corresponding actual parameters.
It is also possible to pass parameters to a function by reference by using the ACCESS parameter attribute. When a parameter is passed by reference, conceptually, the actual parameter itself is passed (and just assigned a new name - the name of the corresponding parameter). Therefore, any changes made to the formal parameter do affect the actual parameter. Please see the information on the ACCESS keyword for additional information.

It is possible to separate the interface and implementation part of a function using the INTERFACE / IMPLEMENTATION functionality.

3.8.3 END_FUNCTION

This ends a FUNCTION.

3.8.4 ACCESS

The ACCESS keyword is used in combination with input parameters to a function to support parameter passing by reference. The standard parameter passing method to a function is by value.

This is a simple example of using reference parameter passing:

```
FUNCTION TEST;
VAR_INPUT
   v1 : ACCESS INT;
   v2 : ACCESS STRING;
   v3 : INT;
END_VAR;
   v1 := v1 * 5;
   v2 := "Hello world";
    v3 := 51;    <--- NOT ALLOWED
END_FUNCTION;

VAR
   x1 : INT := 1;
   x2 : INT := 1;
   s : STRING := "Nice";
END_VAR;
```

Calling the function:

```
TEST( v1 := x1, v2 := s,  v3 := x2 );
```

After the call to "test", the contents of the parameters passed to the function are as follows: "x1" is 5, "x2" is 1, and "s" is "Hello world".
Please note that "x2" did not change its value as it was passed by value. "x1" and "s" were both passed by reference and therefore the changes for the actual parameters to "TEST" changed accordingly.
Actually it is not possible for "TEST" to update the "v3" parameter as changes to VAR_INPUT by value parameters in a function are not allowed. Using the ACCESS attribute on a VAR_INPUT variable will, however, change this rule and allow updating of the referenced parameter.

By using the ACCESS attribute, it is possible to pass a STRUCT_BLOCK to a function:

```
STRUCT_BLOCK server_pack;
   mode         : SINT;
   linsec       : DINT;
   latitude     : DINT;
   longitude    : DINT;
END_STRUCT_BLOCK;


FUNCTION SendData : BOOL;
VAR_INPUT
   v   : ACCESS server_pack;
   res : ACCESS INT;
END_VAR;

   ... Sends the data and returns the status in 'res'.
   res := 0;
   SendData := TRUE;

END_FUNCTION;
```

```
VAR
   pack : server_pack;
   rc   : INT;
END_VAR;
```

Calling the function to send the "pack" of data:

```
... Fill in the data to send in 'pack', and send.
SendData( v := pack, res := rc );
```

Note that any possible changes to the "v" formal parameter in the SendData function will affect the content of the actual parameter "pack".

Using ACCESS parameters to a function has many advantages:

- In addition to the single return value from a function, an arbitrary number of additional return values can be realized.
- It can often be used easier and with a memory saving if used instead of a FUNCTION_BLOCK.
- ACCESS parameter passing with a STRUCT_BLOCK is possible, which allows a composite type to be passed and returned.

.. and only a few rules apply:

- All ACCESS parameters must be assigned by the caller.
- The variable types of the formal parameter and the actual parameter must be identical.

Also see the section on functions in the VPL Introduction Manual.

3.8.5 MANDATORY

The MANDATORY keyword is used in combination with input parameters to a function to support forced parameter.
By using the MANDATORY keyword it is enforced by the VPL compiler that the specified parameter must be assigned when the function is called.

This functionality can be used to ensure that a critical parameter is always assigned by the function caller.
Using the MANDATORY keyword excludes the possibility to assign a default value to the given parameter.

This is a simple example of using the MANDATORY keyword:

```
FUNCTION TEST;
VAR_INPUT
   v1 : MANDATORY INT;
   v2 : MANDATORY STRING;
   v3 : INT := 17;
END_VAR;
END_FUNCTION;
```

The TEST function declares that v1 and v2 must always be assigned by the caller and v3 is optional and if not assigned it will contain the default value of 17.

3.8.6 FUNCTION_BLOCK

Function blocks are an extremely important facility when developing VPL programs. A function block can accept and deliver different values to the calling program. As all function blocks are instantiated once and thereafter keep their own variables etc. in memory, they survive between calls to them, and can therefore deliver different results for each call. This is opposite to a function which has no knowledge of the previous times it was called (the variables of a function will not survive between multiple calls).

Have a look at the following function block, which in fact is a function block taken from the RTCU environment and therefore is available "right out of the box". You can find some more details about the CTD function block here.

```
FUNCTION_BLOCK CTD;

 VAR_INPUT
    cd : BOOL R_EDGE; | count down (R_EDGE will detect a leading edge on the
signal)
    ld : BOOL;        | load preset value
    pv : INT;         | preset value
 END_VAR;

 VAR_OUTPUT
    q  : BOOL; | count down occurred
    cv : INT;  | current value
 END_VAR;

 IF ld THEN
    cv := pv;
 ELSIF cd
    cv := cv-1;
 END_IF;
 q := cv <= 0;
END_FUNCTION_BLOCK;
```

When declaring a variable of the type BOOL in the VAR_INPUT section, it is possible to define that the variable only become TRUE if either a leading edge (R_EDGE) is detected or if a falling edge (F_EDGE) is detected.

The CTD function block above is very usable in cases where one needs to count down on some event. The CTD function block will decrement its counter value by 1 on each leading edge on "cd". Setting "ld" to TRUE will load the preset value "pv" into the counter "cv". The output "q" will be TRUE when the counter "cv" reaches 0.

CTD is a wonderful example of the many benefits of using function blocks. The complete logic of the CTD "function" is encapsulated in a unit with a very well-defined interface to the outside world. When one has a large library of predefined and developed function blocks (and functions), development of new programs will both be easier and also less time consuming. The number of errors introduced to a new program will also be reduced as many of the function blocks that one uses already have been used many times before and are therefore thoroughly tested.

As it was the case with functions, variables declared in the VAR_INPUT section can only be modified outside of the function block as variables in the VAR_OUTPUT section can only be read outside the function block (but not modified). Declarations in the VAR section are the private variables of the function block and can therefore only be used from inside the function block.

The instantiation of a function block is done in the following way:

```
 AAA : CTD;
```

To get access to the inputs and outputs of AAA, the following notation is used: AAA.<variable>

To set the initial value of the CTD (pv, preset value), one can use the following code:

```
AAA.pv := 1000;
```

This will set the default value (preset value) to 1000.

Another way to accomplish this is initializing the "pv" to 1000 and at the same time call the function block so it will be executed:

```
AAA(pv:=1000);
```

This has exactly the same effect as first assigning the value 1000 to "pv" and then calling the function block:

```
AAA.pv := 10000;
AAA();
```

Example on the use of CTD:

```
// Simple example showing the use of CTD

INCLUDE RTCU.INC

VAR_INPUT
   i1 : BOOL; | An input
   i2 : BOOL; | Another input
   a  : CTD;
END_VAR;

VAR_OUTPUT
   o  : BOOL; | output
END_VAR;

PROGRAM CTD_test;

 a.pv := 1000; // set preset value

BEGIN
   IF i1 THEN
      // load default value
      a(ld := TRUE);
      a(ld := FALSE);
   ELSE
      a(cd := i2);
   END_IF;
   IF a.q THEN
      // count down:
      o := NOT o;
   END_IF;
END;

END_PROGRAM;
```

It is possible to separate the interface and implementation part of a function block using the INTERFACE / IMPLEMENTATION functionality.

3.8.7 END_FUNCTION_BLOCK

This ends a function block.

3.8.8 INTERFACE

Using the INTERFACE / IMPLEMENTATION functionality it is possible to split a FUNCTION and FUNCTION_BLOCK into two sections. One section that caters for the interface and another section that caters for the actual implementation.

Using this separation it is possible to expose the interface in a separate INC file and then the implementation can be in another INC file that may be encrypted.

Another advantage with this separation is to break dependencies in large program complexes, such that as soon as the INTERFACE has been declared, then other parts of the code can reference the function / function block.

In the interface section of a function / function block, it is not allowed to specify local variables or any code statements. Similarly, in the implementation section, no return value or input/output variables can be specified.

Example of usage:

```
FUNCTION INTERFACE funcA:INT
VAR_INPUT
   a : INT;
   b : DOUBLE;
END_VAR;
// Notice that no local variables are allowed in the interface section.
// Code is also not allowed.
END_FUNCTION;


FUNCTION testA;
VAR
   p : INT;
END_VAR;

//This function references funcA, where only the interface is defined.
p := funcA(a := 10,b := 27.14);

END_FUNCTION;


FUNCTION IMPLEMENTATION funcA; // return value cannot be specified here.
//No VAR_INPUT is allowed here.
VAR
   aaa : INT; //Local variables are allowed here.
END_VAR;

END_FUNCTION;
```

3.8.9 IMPLEMENTATION

IMPLEMENTATION can be used in conjunction with FUNCTION and FUNCTION_BLOCK to implement the function or function block as defined by a prior INTERFACE definition.

Please see the section in INTERFACE for detailed usage information.

3.8.10 ASYNC

The ASYNC option can be used in the declaration of a function block.

When a program calls a function block without the ASYNC option, the execution of the code in the function block is said to be synchronous. That is, the calling program will first gain control again when the called function block is "finished" executing. If the option ASYNC is used in the declaration of a function block, the execution is said to be asynchronous.

When the calling program calls the function block, the execution of the function block is initiated as a separate thread. The calling program immediately regains the control and continues to execute its code. While it does so, the function block will also continue to execute its own code. If the calling program needs to know when the function block is finished executing, the function block needs a way of informing the calling program of this. This could be accomplished, for example, by using a variable in the function blocks VAR_OUTPUT section.

The execution of the asynchronous function blocks is done sequentially in the order of execution. This means that while one asynchronous function block is executing, a call to another asynchronous function block will be queued for execution until the currently executing function block terminates.

Instead of using an ASYNC function block, it is highly recommended to use true multithreading.

The ASYNC option is used like this:

```
FUNCTION_BLOCK ASYNC test;
...
END_FUNCTION_BLOCK;
```

3.8.11 ALIGN

The ALIGN option can be used in the declaration of a function and a function block. Normally, when you declare a function / function block without the ALIGN option, the VPL translator will pack all the data in the VAR_INPUT , VAR_OUTPUT, and VAR sections on a 1 byte boundary, but when using the ALIGN option, all variables that occupy 2 or 4 bytes will be aligned to the 16-bit architecture of the target platform.

The ALIGN option is only used in combination with the MODCALL to ensure that the data layout in the function block corresponds with the layout in the externally called C function.

```
FUNCTION_BLOCK ALIGN test;
...
END_FUNCTION_BLOCK;
```

3.8.12 MUTABLE

MUTABLE is an attribute that can be used on variables that are declared in a VAR_INPUT section of a function block or a thread block.

By using this attribute it is legal to assign a value to this variable from inside the function block / thread block.
The MUTABLE attribute can only be used on variables of simple data types.

This is a simple example of using mutable variables:

```
FUNCTION_BLOCK CB_TEST;
VAR_INPUT
   v1 : MUTABLE INT;
   v2 : MUTABLE STRING;
   v3 : INT;
END_VAR;
   v1 := v1 * 5;
   v2 := "Hello world";
   v3 := 51;    <--- NOT ALLOWED
END_FUNCTION_BLOCK;

VAR
   test : CB_TEST;
END_VAR;
```

Calling the function block:

```
test( v1 := 1, v2 := "Nice", v3 := 1 );
```

After the call to "test", the contents of the input variables are as follows: "v1" is 5, "v2" is "Hello world", and "v3" is 1.
"v1" and "v2" are both mutable and their values are therefore changed, while "v3" is not mutable and its contents remain unchanged.

3.8.13 CALLBACK

CALLBACK is used in relation to the declaration and usage of callback functions.

A callback function is a mechanism where a VPL function is called externally from either the firmware or from an RTCU M2M Platform SDK extension module. The canitpOpen function is an example of the use of a callback function to signal data reception.
Using a program architecture utilizing the callback functionality supports an event-driven design that can prove to take fewer resources, be more responsive, and yield a better code structure.

The CALLBACK keyword can be used both as an attribute to a function, but also as a variable type for holding a reference to a callback function.
The address-of operator @ is used to get the address of a callback function.

Below is a simple example of using the callback functionality:

```
INCLUDE rtcu.inc

FUNCTION CALLBACK myTimer;
VAR_INPUT
   no  : INT;
   arg : DINT;
END_VAR;
DebugFmt(message:="Timer#\1,arg=\4",v1:=no,v4:=arg);
END_FUNCTION;


PROGRAM test;

clockStart(no:=0,freq:=2,func:=@myTimer,arg:=2);
clockStart(no:=1,freq:=5,func:=@myTimer,arg:=5);
clockStart(no:=2,freq:=10,func:=@myTimer,arg:=10);

WHILE TRUE DO
   Sleep(delay:=1000);
END_WHILE;

END_PROGRAM;
```

The above function uses an inbuilt timer functionality to install three timers that each calls the callback at a given frequency. The first timer is called every 2 seconds, the second timer every 5 seconds, and the last timer every 10 seconds.
The output when running the program are as follows:

```
10:42:43 -> Timer#0,arg=2
10:42:45 -> Timer#0,arg=2
10:42:47 -> Timer#0,arg=2
10:42:47 -> Timer#1,arg=5
10:42:49 -> Timer#0,arg=2
10:42:51 -> Timer#0,arg=2
10:42:52 -> Timer#1,arg=5
10:42:52 -> Timer#2,arg=10
10:42:53 -> Timer#0,arg=2
10:42:55 -> Timer#0,arg=2
10:42:57 -> Timer#0,arg=2
10:42:57 -> Timer#1,arg=5
10:42:59 -> Timer#0,arg=2
10:43:01 -> Timer#0,arg=2
10:43:02 -> Timer#1,arg=5
10:43:02 -> Timer#2,arg=10
10:43:03 -> Timer#0,arg=2
```

```
10:43:05 -> Timer#0,arg=2
10:43:07 -> Timer#0,arg=2
10:43:07 -> Timer#1,arg=5
10:43:09 -> Timer#0,arg=2
10:43:11 -> Timer#0,arg=2
10:43:12 -> Timer#1,arg=5
10:43:12 -> Timer#2,arg=10
```

The clockStart function used is declared as follows:

```
FUNCTION ALIGN clockStart : INT;
VAR_INPUT
   no   : INT;  //clock number from 0..9
   freq : INT;  //in seconds. Set to 0 to stop clock.
   func : CALLBACK;  //callback function with the following parameters:
no:int, arg:dint
   arg  : DINT; //argument
END_VAR;
END_FUNCTION;
```

The clockStart function is only supported on NX32L architecture devices.

For more detailed information on the implementation of callback functions, please refer to the **RTCU M2M Platform SDK**.

3.9 Multithreading

3.9.1 Multithreading

Introduction

Support for multithreading is available on all devices with differences in the available resources depending on the RTCU architecture in use.

Please continue reading the section What is multithreading?

3.9.1.1 What is multithreading?

Multithreading is an environment where more than one "thread" of execution can take place at the same time. Without support for multithreading, or more than one "thread" of execution, the programmer will be forced to handle everything in one execution context.
Windows is a multithreaded environment in that many simultanous things can take place at the same time: the email client program can receive an email at the same time as the user is using the spreadsheet program and maybe copying a file all at the same time.

In a VPL program without multithreading, only one thread of execution is automatically started by the RTCU system software:

```
PROGRAM test;
                          <--- This is where the main thread automatically
starts execution.
DebugMsg(message:="Hello from VPL Program");
END_PROGRAM;
```

With multithreading, however, the VPL program can start additional threads of execution by declaring thread-blocks:

```
THREAD_BLOCK Piper;
                          <--- This is where the thread "Piper" will start
execution.
DebugMsg(message:="Hello from Piper!");
END_THREAD_BLOCK;


PROGRAM test;
VAR
   th : Piper;            <--- This declares an instance of the thread
"Piper".
END_VAR;

th();                     <--- This is where the thread "Piper" is started.
DebugMsg(message:="Hello from main!");
END_PROGRAM;
```

The output of the the multithreaded program above will therefore be:

```
Hello from main!
Hello from piper!
```

**** OR:

```
Hello from piper!
Hello from main!
```

The order of the execution cannot be predicted as the two threads execute concurrently.

Please continue reading the section Why use multithreading?

3.9.1.2 Why use multithreading?

As has already been pointed out, multithreading offers several "threads" of execution. This feature allows much easier program development as the problem can be broken down in several "sub-tasks" - threads as they are called. Multithreading will also make it easier to implement programs where different "sub-tasks" have different priorities and timing. Consider a high-speed I/O thread that takes care of the I/O handling by generating alarm messages to an SMS thread that is responsible for sending out SMS messages. It is clear that the I/O thread it not allowed to be halted because it takes time for the SMS message to be sent out. If the I/O thread blocks for a certain amount of time, it will most likely result in several input changes becoming lost with potentially severe consequences.

Multithreading your code can have many benefits. These include:

- Improved application responsiveness - any program in which many activities are not dependent upon each other can be redesigned so that each activity is defined as a thread.
- Improved program structure - many programs are more efficiently structured as multiple independent or semi-independent units of execution instead of as a single, monolithic thread. Multithreaded programs can be more adaptive to variations in demands than single threaded programs.
- Uses fewer resources compared to multiple program tasks.

Please continue reading the section Using multithreading.

3.9.1.3 Using multithreading

As a short example of how to use multithreading, we will look into a VPL program that monitors two digital inputs that will generate an SMS message on each rising edge. That could provide the basis for a simple alarm system with two door switches.

First we take a look at how the somewhat simplified program will look without multithreading:

```
INCLUDE rtcu.inc

VAR_INPUT
   al_1 : BOOL R_EDGE;
   al_2 : BOOL R_EDGE;
END_VAR;

PROGRAM test;
gsmPower(power := TRUE);
BEGIN
   IF al_1 THEN
      //Alarm!
      gsmSendSMS(phonenumber := "22448899", message := "Alarm. Door 1");
   END_IF;
   IF al_2 THEN
      //Alarm!
      gsmSendSMS(phonenumber := "22448899", message := "Alarm. Door 2");
```

```
   END_IF;
END;

END_PROGRAM;
```

This is a very simple program with one serious problem. If the "al_1" input goes high, it will send out an SMS message. That will take maybe 5..6 seconds before the call to gsmSendSMS() returns. If during this period the "al_2" (or the "al_1" once more) input goes high and low, it will never be detected by the program. This is serious as another thief could just have broken in.

Here is how we can improve these shortcomings with multithreading:

```
INCLUDE rtcu.inc


VAR_INPUT
   al_1 : BOOL R_EDGE;
   al_2 : BOOL R_EDGE;
END_VAR;

VAR
   chalarm : CHANNEL;
END_VAR;

//SMS thread
THREAD_BLOCK smsalarm;
VAR
   alarmno : SINT;
END_VAR;

WHILE TRUE DO
   chRead(ch:=chalarm,msg:=ADDR(alarmno),lenmax:=SIZEOF(alarmno));
   gsmSendSMS(phonenumber := "22448899",message := strFormat(format:="Alarm.
Door \1", v1:=alarmno));
END_WHILE;

END_THREAD_BLOCK;


PROGRAM test;
VAR
   smshandler : smsalarm;
   alarm : sint;
END_VAR;

//main thread
chalarm := chInit(msgMax:=10);
smshandler();
gsmPower(power := TRUE);
BEGIN
   IF al_1 THEN
      alarm:=1;
      chWrite(ch:=chalarm,msg:=ADDR(alarm),len:=SIZEOF(alarm));
   END_IF;
   IF al_2 THEN
      alarm:=2;
      chWrite(ch:=chalarm,msg:=ADDR(alarm),len:=SIZEOF(alarm));
   END_IF;
END;

END_PROGRAM;
```

This program is a bit longer than the single threaded version, but it will address all the shortcomings previously mentioned.
We realize that both the responsiveness and correctness of the application have been improved. The program structure can also be said to have improved, but this will be more visible in larger and more complex programs.

The program uses the Channel mechanism for passing the alarm as a number to the SMS thread. The chWrite() function call is non-blocking which means that the main thread will very quickly send the message and continue its operation. The SMS thread will automatically be blocked in the chRead() function call as it waits for messages from the main thread. When a message arrives, it will send the actual SMS message by using the gsmSendSMS() function. The speed of sending the SMS message is not important, because when new alarms do occur, they will simply be queued in the Channel mechanism and processed in the order they were inserted into the channel.

Please continue reading the section Thread syncronization.

3.9.1.4 Thread synchronization

When developing multithreaded applications, one important issue to understand is the need for thread synchronization. In the previous example with the SMS alarm application, we actually did see the Channel thread synchronization primitive used. Another need for thread synchronization occurs when more than one thread accesses shared data - like global variables used by several threads.

The MUTEX (Mutual Exclusion) data type is the most important mechanism that is used for implementing critical sections in a multithreading VPL program. Critical sections are used in the cases where more than one thread is accessing the same shared resource - like a global variable/data structure accessed from several threads.
By using a Mutex, it is ensured that only one thread will be executing in the critical section at one time. Other threads that also want to enter the critical section will be blocked until the thread executing in the critical section leaves it.

Use of critical sections is important to avoid the so-called "lost update problem" that occurs because each piece of code assumes that between the point where it takes a copy of the data and the point where it writes it back, no other tasks will change the data.
The VPL line:

```
Count:=Count + 1;
```

Will actually be executed as 3 machine code instructions as seen below:

```
LD      A,[Count]           ; Load the Count value into
                            register A
ADD     A,1                 ; Add 1 to the value of register A
ST      [COUNT],A           ; Save register a to Count
```

If we assume that two threads are executing these instructions to increment the Count variable:

```
        Thread A                    Thread B
A1:     LD      A,[Count]   B1:     LD      B,[Count]
A2:     ADD     A,1         B2:     ADD     B,5
A3:     ST      [COUNT],A   B3:     ST      [COUNT],B
```

Consider the following sequence of execution:

A1, A2, task_switch, B1, B2, B3, task_switch, A3 …

What problem occurs here?

Instruction	Thread A	Memory	Thread B
	A	**counter**	**B**
	?	10	?
A1	10	10	?
A2	11	10	?
B1	11	10	10
B2	11	10	15
B3	11	15	15
A3	11	11	15

Thread A will increment Count with 1 and Thread B will increment Count with 5. As Count is 10 initially, the correct answer after execution will be 16, but because of the lost update problem, the result is 11 instead.

By placing the instructions to increment the Count variable in a critical section, it will be assured that the load, increment, and store are executed in one atomic operation which will eliminate the lost update problem:

```
      Thread A
A0:   mxLock();
A1:   LD      A,[Count]
A2:   ADD     A,1
A3:   ST      [COUNT],A
A4:   mxUnlock();
```

```
      Thread B
B0:   mxLock();
B1:   LD      B,[Count]
B2:   ADD     B,5
B3:   ST      [COUNT],B
B4:   mxUnlock();
```

The sequence of execution is now:

A0, A1, A2, task_switch, B0, task_switch, A3, A4,task_switch, B1, B2, B3, B4 …

It can now be seen that by using a critical section, we have eliminated the lost update problem.

The short example below demonstrates how to protect the "Count" variable when accessed from 3 threads - the main thread and 2 threads dynamically created.
Without the use of critical sections implemented by a Mutex, the lost update problem will appear because the increment of the "Count" is composed of several machine code instructions that may be interrupted at any time.

```
INCLUDE rtcu.inc

VAR
   mxCnt : MUTEX;
   Count : DINT := 0;
END_VAR;

THREAD_BLOCK Thread_A;
```

```
WHILE TRUE DO
   mxLock(mx:=mxCnt);
   Count := Count + 1;
   mxUnlock(mx:=mxCnt);
END_WHILE;
END_THREAD_BLOCK;


THREAD_BLOCK Thread_B;
WHILE TRUE DO
   mxLock(mx:=mxCnt);
   Count := Count + 5;
   mxUnlock(mx:=mxCnt);
END_WHILE;
END_THREAD_BLOCK;


PROGRAM test;
VAR
   TA    : Thread_A;
   TB    : Thread_B;
   i     : INT;
END_VAR;

mxCnt := mxInit();

TA();
TB();

BEGIN
   Sleep(delay:=1000);
   DebugFmt(message:="Count: \4",v4:=Count);
END;

END_PROGRAM;
```

Another synchronization mechanism is the Semaphore.

Semaphores are a programming construct designed by E. W. Dijkstra in the late 1960s. Dijkstra's model was the operation of railroads. Consider, for example, a stretch of railroad in which there is a single track over which only one train at a time is allowed. Guarding this track is a semaphore. A train must wait before entering the single track until the semaphore is in a state that permits travel. When the train enters the track, the semaphore changes state to prevent other trains from entering the track. A train that is leaving this section of track must again change the state of the semaphore to allow another train to enter.

One classical example of the use of Semaphores is the so-called "Producer/Consumer problem" where a producer thread produces data to a consumer thread, and access to the shared buffer in between must be synchronized. In the RTCU environment, a CHANNEL could have been used to solve this problem but a more low-level implementation using three Semaphores could look like this:

```
INCLUDE rtcu.inc

VAR
   buffer   : ARRAY[0..4] OF INT;
   buf_in   : INT;
   buf_out  : INT;
   crit     : SEMAPHORE;
   notempty : SEMAPHORE;
```

```
   full     : SEMAPHORE;
END_VAR;

THREAD_BLOCK Producer;
VAR
   elem : INT;
END_VAR;
WHILE TRUE DO
   Sleep(delay:=1000);
   elem:=elem+1;
   semWait(sem:=notempty);
   semWait(sem:=crit);
   buffer[buf_in]:=elem;
   buf_in:=(buf_in+1) MOD (SIZEOF(buffer)/SIZEOF(buffer[0]));
   DebugFmt(message:="Produced=\1",v1:=elem);
   semSignal(sem:=crit);
   semSignal(sem:=full);
END_WHILE;
END_THREAD_BLOCK;

THREAD_BLOCK Consumer;
VAR
   elem : INT;
END_VAR;
WHILE TRUE DO
   Sleep(delay:=1000);
   semWait(sem:=full);
   semWait(sem:=crit);
   elem:=buffer[buf_out];
   buf_out:=(buf_out+1) MOD (SIZEOF(buffer)/SIZEOF(buffer[0]));
   DebugFmt(message:="Consumed=\1",v1:=elem);
   semSignal(sem:=crit);
   semSignal(sem:=notempty);
END_WHILE;
END_THREAD_BLOCK;

PROGRAM test;
VAR
   P : Producer;
   C : Consumer;
END_VAR;

notempty:=semInit(initval:=SIZEOF(buffer)/SIZEOF(buffer[0]));
full:=semInit(initval:=0);
crit:=semInit(initval:=1);
P();
C();

BEGIN
END;
END_PROGRAM;
```

The *notempty* and *full* Semaphores are used for synchronizing the insertion and removal to the buffer, so even as the Producer is producing faster than the Consumer, they will be fully synchronized without any data loss.
The *crit* semaphore is used exactly with the same purpose as a *Mutex*, and, as it has been demonstrated, a MUTEX can also be implemented by using a Semaphore with the initial value of 1 - also called a binary semaphore
Semaphores are rarely used for implementing critical sections because using the Mutex is a safer and more elegant way to accomplish this.

Please read more about the different synchronization mechanism in their respective sections - MUTEX, SEMAPHORE and CHANNEL.
Also have a look at the RTCU Multithreading data sheet for the limitations present in the system.

3.9.1.5 RTCU Multithreading data sheet

X32 architecture:

Maximum number of dynamically created threads in system:	**14**
Maximum number of SEMAPHORE in system:	**32**
Maximum number of MUTEX in system:	**32**
Maximum number of CHANNEL in system:	**16**
CHANNEL message pool:	**20000 bytes.**
Thread scheduling policy:	**Preemptive-FIFO**
Thread time slice policy:	**Fixed instruction slice / blocking event**

NX32 architecture:

Maximum number of dynamically created threads in system:	**20**
Maximum number of SEMAPHORE in system:	**64**
Maximum number of MUTEX in system:	**64**
Maximum number of CHANNEL in system:	**32**
CHANNEL message pool:	**40000 bytes.**
Thread scheduling policy:	**Preemptive-FIFO**
Thread time slice policy:	**Fixed instruction slice / blocking event**

NX32L architecture:

Maximum number of dynamically created threads in system:	**64**
Maximum number of SEMAPHORE in system:	**256**
Maximum number of MUTEX in system:	**256**
Maximum number of CHANNEL in system:	**128**
CHANNEL message pool:	**262144 bytes.**
Thread scheduling policy:	**Preemptive-FIFO**
Thread time slice policy:	**Fixed instruction slice / blocking event**

3.9.2 THREAD_BLOCK

This defines the beginning of a thread-block as described in more detail in the section on multithreading
The general structure of a thread-block is:

```
THREAD_BLOCK <name>;
VAR_INPUT
     ...
END_VAR;
VAR_OUTPUT
     ...
END_VAR;
     ...
END_THREAD_BLOCK;
```

The syntax is therefore the same as a function-block but with the word THREAD_BLOCK used instead of FUNCTION_BLOCK.
Also the ALIGN and ASYNC attributes are not allowed on a thread-block.

Advise:
When using threads, it is advised to ensure that all threads will periodically block waiting for some event like a Channel, Semaphore, Mutex, and/or including a Sleep operation. A thread that runs constantly we leave less processing power for other threads.

Also see the implicit member variable _running.

Example:

```
INCLUDE rtcu.inc

THREAD_BLOCK Piper;

DebugMsg(message:="Hello from Piper!");

END_THREAD_BLOCK;


PROGRAM test;
VAR
   th : Piper;
END_VAR;

th();
WHILE th._running DO
   Sleep(delay:=100);
END_WHILE;
DebugMsg(message:="Piper has terminated");
END_PROGRAM;
```

3.9.3 END_THREAD_BLOCK

This ends the declaration of a THREAD_BLOCK.

3.9.4 _running (implicit variable)

The _**running** is an implicit BOOL variable that is automatically declared and handled by the system when a THREAD_BLOCK is declared.
The usage of the _**running** BOOL variable is to signal whether the thread is running or not. This can be used for checking if a thread has successfully started, as well as to catch an attempt where more threads than the system allows are started. It can also be used to synchronize with a thread that has stopped running after some condition has been met.

Consider this THREAD_BLOCK:

```
THREAD_BLOCK TEST;
VAR_INPUT
   stop : BOOL;
END_VAR;
   ...
END_THREAD_BLOCK;
```

The VPL compiler will automatically allocate a hidden variable named _**running** of the type BOOL in the VAR_OUTPUT section of the thread-block. It will also insert code that updates the status:

```
THREAD_BLOCK TEST;
VAR_INPUT
   stop : BOOL;
END_VAR;
VAR_OUTPUT
   _running : BOOL;              <--- Automatically generated by the compiler.
END_VAR;
   ...
   _running:=FALSE;              <--- Automatically generated by the compiler.
END_THREAD_BLOCK;
```

The assignment of FALSE to _running will be the very last operation done by the thread before it ceases.

When the TEST thread-block is instantiated, the updating of _**running** will again be done automatically:

```
PROGRAM abc;
VAR
   myTest : TEST;
END_VAR;

myTest();
myTest._running:=TRUE;           <--- Automatically generated by the compiler.

END_PROGRAM;
```

If the thread has successfully started, the myTest._running variable will be set to TRUE which indicates that the thread is running. If the thread failed to start, it will be set to FALSE by the system.

Its possible to dynamically terminate an active thread. This can be seen in the example below. The program creates and starts a thread, waits for 20 seconds, and then terminates the thread.

```
INCLUDE rtcu.inc

THREAD_BLOCK TEST;
VAR_INPUT
```

```
   stop : BOOL;
END_VAR;

WHILE NOT stop DO
   DebugMsg(message:="Working hard...");
   Sleep(delay:=1000);
END_WHILE;

END_THREAD_BLOCK;


PROGRAM abc;
VAR
   myTest : TEST;
END_VAR;

myTest(stop:=FALSE);
Sleep(delay:=20000);
myTest.stop:=TRUE;
WHILE myTest._running DO
   DebugMsg(message:="Waiting for TEST thread to terminate...");
   Sleep(delay:=1000);
END_WHILE;

END_PROGRAM;
```

3.9.5 thGetID(Function)

This will return a unique ID for the calling thread.

Input:
None

Returns: **INT**
The Thread ID of the calling thread.

Declaration:
```
FUNCTION thGetID() : INT;
```

3.10 Operators

3.10.1 Operators

The VPL programming language offers the following operators.

The number to the left of each operator shows the priority of the operator. Operators with the same priority are evaluated from left to right.

1.	(expression)	Parenthesis (brackets).
2.	function evaluation	Function call.
	SIZEOF	Size of operator.
	ADDR	Address operator.
	@	Address-of a callback function operator.
	BYTE(expression)	Typecast expression to BYTE.
	UINT(expression)	Typecast expression to UINT.
	INT(expression)	Typecast expression to INT.
	USINT(expression)	Typecast expression to USINT.
	SINT(expression)	Typecast expression to SINT.
	DINT(expression)	Typecast expression to DINT.
	BOOL(expression)	Typecast expression to BOOL.
	DOUBLE(expression)	Typecase expression to DOUBLE.
	FLOAT(expression)	Typecast expression to FLOAT.
	FLOATB(expression)	Typecast expression to FLOAT binary.

Unary operators

3.	-	Negating.
	NOT	Logic negation.
4.	**	Exponential.

Arithmetic operators

5.	*	Multiplication.
	/	Division.
	MOD	Modulus.
6.	+	Addition.
	-	Subtraction.

Relational(comparison) operators

7.	<	Less than.
	<=	Less than or equal.
	>	Greater than.
	>=	Greater than or equal.
	=	Equal.
	<>	Different from.

Logical operators

8.	AND	Boolean AND/Bitwise AND.
9.	XOR	Boolean Exclusive OR/Bitwise Exclusive OR.
10.	OR	Boolean OR/Bitwise OR.

3.10.2 Typecast BYTE(expression)

The BYTE typecast operator is used to convert an expression to an BYTE type. If the typecast operator is used on a "larger" type, truncation will occur.

Example:

```
INCLUDE rtcu.inc

VAR
   a : BYTE;
   b : INT;
END_VAR;

PROGRAM test;

// Convert 'b' which is a INT type to a BYTE (truncation will occur)
a := BYTE(b);

END_PROGRAM;
```

3.10.3 Typecast USINT(expression)

The USINT typecast operator is used to convert an expression to an USINT type. If the typecast operator is used on a "larger" type, truncation will occur.

Example:

```
INCLUDE rtcu.inc

VAR
   a : USINT;
   b : INT;
END_VAR;

PROGRAM test;

// Convert 'b' which is a INT type to a USINT (truncation will occur)
a := USINT(b);

END_PROGRAM;
```

3.10.4 Typecast SINT(expression)

The SINT typecast operator is used to convert an expression to a SINT type. If the typecast operator is used on a "larger" type, truncation will occur.

Example:

```
INCLUDE rtcu.inc

VAR
   a : INT;
   b : SINT;
END_VAR;

PROGRAM test;

BEGIN
   // Convert 'a' which is a INT type to a SINT (truncation will occur)
   b := SINT(a);
   ...
END;

END_PROGRAM;
```

3.10.5 Typecast UINT(expression)

The UINT typecast operator is used to convert an expression to an UINT type. If the typecast operator is used on a "larger" type, truncation will occur.

Example:

```
INCLUDE rtcu.inc

VAR
   a : UINT;
   b : DINT;
END_VAR;

PROGRAM test;

// Convert 'b' which is a DINT type to a UINT (truncation will occur)
a := UINT(b);

END_PROGRAM;
```

3.10.6 Typecast INT(expression)

The INT typecast operator is used to convert an expression to an INT type. If the typecast operator is used on a "larger" type, truncation will occur. If the typecast is used to convert from a "smaller" type, sign extension will occur.

Example:

```
INCLUDE rtcu.inc

VAR
   a : INT;
   b : DINT;
END_VAR;

PROGRAM test;

// Convert 'b' which is a DINT type to a INT (truncation will occur)
a := INT(b);

END_PROGRAM;
```

3.10.7 Typecast DINT(expression)

The DINT typecast operator is used to convert an expression to a DINT type. If the typecast is used to convert from a "smaller" type, sign extension will occur.

Example:

```
INCLUDE rtcu.inc

VAR
   a : INT;
   b : DINT;
END_VAR;

PROGRAM test;

// Convert 'a' which is a INT type to a DINT (sign extension will occur)
b := DINT(a);

END_PROGRAM;
```

3.10.8 Typecast BOOL(expression)

The BOOL typecast operator is used to convert an expression to a BOOL type. Only if the expression is equal to 0 will the result be FALSE. Otherwise it will be TRUE.

Example:

```
INCLUDE rtcu.inc

VAR
   a : INT;
   b : BOOL;
END_VAR;

PROGRAM test;

BEGIN
   // Convert 'a' which is a INT type to a BOOL (if a <> 0, b will be TRUE,
else FALSE)
   b := BOOL(a);
   ...
END;

END_PROGRAM;
```

3.10.9 Typecast DOUBLE(expression)

The DOUBLE typecast operator is used to convert an expression to a DOUBLE type. This typecast is done numerically so that for example typecasting an integer value of 56 will yield a floating-point value of equally 56.0.

Example:

```
INCLUDE rtcu.inc

VAR
   a : DINT;
   f : DOUBLE;
END_VAR;

PROGRAM test;

// Convert 'a' which is a DINT type to a DOUBLE
a := 4711; // Integer value to convert.
f := DOUBLE(a); // f will now contain 4711.00

END_PROGRAM;
```

3.10.10 Typecast FLOAT(expression)

The FLOAT typecast operator is used to convert an expression to a FLOAT type. This typecast is done numerically so that for example typecasting an integer value of 56 will yield a floating-point value of equally 56.0.

Example:

```
INCLUDE rtcu.inc

VAR
   a : DINT;
   f : FLOAT;
END_VAR;

PROGRAM test;

// Convert 'a' which is a DINT type to a FLOAT
a := 4711; // Integer value to convert.
f := FLOAT(a); // f will now contain 4711.00

END_PROGRAM;
```

3.10.11 Typecast FLOATB(expression)

The FLOATB typecast operator is used to convert an expression to a FLOAT type. This typecast is done binary instead of numerically, making the FLOAT the raw value of the expression.

Example:

```
INCLUDE rtcu.inc

VAR
   a : DINT;
   f : FLOAT;
END_VAR;

PROGRAM test;

// Convert 'a' which is a DINT type to a FLOAT
a := 16#7F800000; // Positive Infinity in IEEE 754 single precision
f := FLOATB(a); // Infinity

END_PROGRAM;
```

3.10.12 ADDR

The ADDR operator returns the address of a VPL object. The address of an object is always a PTR type.

Example:

```
INCLUDE rtcu.inc

VAR
   a : INT;
   b : PTR;
END_VAR;

PROGRAM test;

BEGIN
   // assign the address of 'a' to b
   b := ADDR(a);
   ...
END;

END_PROGRAM;
```

3.10.13 @

The @ operator returns the address of an VPL callback function.
Please refer to the function attribute CALLBACK for more details.

3.10.14 AND

The logical operator AND operates on expressions with the BOOL, SINT, INT and DINT data types. When operating on data types other than BOOL, the operation is bitwise.
When operating on BOOL, it has the following truth table:

Input 1	Input 2	Output1
0	0	0
0	1	0
1	0	0
1	1	1

Example:

```
INCLUDE rtcu.inc

VAR
   a  : BOOL;
   b  : BOOL;
   ia : INT;
   ib : INT;
END_VAR;

PROGRAM test;

BEGIN
   ...
   // Mask off the lower 8 bits of ib
   ia := ib AND 16#00FF;
   ...
   IF a AND b THEN
      // This will be executed if both a and b are TRUE
      ...
   END_IF;
   ...
END;

END_PROGRAM;
```

3.10.15 MOD

The MOD operator returns the remainder of a division between two integer or floating-point numbers.

Example:

```
INCLUDE rtcu.inc

VAR
   a : INT;
   b : INT;
END_VAR;

PROGRAM test;

BEGIN
   ...
   a := b MOD 10;
   ...
END;

END_PROGRAM;
```

3.10.16 NOT

The logical operator NOT operates on expressions with the BOOL, SINT, INT, and DINT data types. When operating on data types other than BOOL, the operation is bitwise (which is to say that all "1" bits will be set to "0" and vice versa). When operating on BOOL, it has the following truth table:

Input	Output
0	1
1	0

It can also be used in #IFDEF conditional compilation directive.

Example:

```
INCLUDE rtcu.inc

VAR
   a : BOOL;
   i : INT;
   j : INT;
END_VAR;

PROGRAM test;

BEGIN
   ...
   i:=16#00FF;
   j:=NOT i;
   // Now j is 16#FF00
   ...
   IF NOT a THEN
      // This will be executed if a are FALSE
      ...
   END_IF;
   ...
END;

END_PROGRAM;
```

3.10.17 SIZEOF

The SIZEOF operator returns the size of VPL object in bytes.
The actual type returned depends on the size of the object. If the size is less than 32767 then the type returned will be an INT - else it will be returned as a DINT.

Example:

```
INCLUDE rtcu.inc

VAR
   a : DINT;
   b : INT;
END_VAR;

PROGRAM test;

BEGIN
   ...
   // assign the size of 'a' in bytes to 'b' (in this case 4)
   b := SIZEOF(a);
   ...
END;

END_PROGRAM;
```

3.10.18 XOR

The logical operator XOR operates on expressions with the BOOL, SINT, INT and DINT data types. When operating on data types other than BOOL, the operation is bitwise.
When operating on BOOL, it has the following truth table:

Input 1	Input 2	Output1
0	0	0
0	1	1
1	0	1
1	1	0

Example:

```
INCLUDE rtcu.inc

VAR
   a  : BOOL;
   b  : BOOL;
   ia : INT;
   ib : INT;
END_VAR;

PROGRAM test;

BEGIN
   ...
   // All bits in ib that are different than 16#00FF will be set in ia (see
thruth table)
   ia := ib XOR 16#00FF;
   ...
   IF a XOR b THEN
      // This will be executed if a and b are different
      ...
   END_IF;
   ...
END;

END_PROGRAM;
```

3.10.19 OR

The logical operator OR operates on expressions with the BOOL, SINT, INT and DINT data types. When operating on data types other than BOOL, the operation is bitwise.
When operating on BOOL, it has the following truth table:

Input 1	Input 2	Output1
0	0	0
0	1	1
1	0	1
1	1	1

Example:

```
INCLUDE rtcu.inc

VAR
   a  : BOOL;
   b  : BOOL;
   ia : INT;
   ib : INT;
END_VAR;

PROGRAM test;

BEGIN
   ...
   // Bitwise OR of ib and 16#00F0
   ia := ib OR 16#00F0;
   ...
   IF a OR b THEN
      // This will be executed if a or b are TRUE
      ...
   END_IF;
   ...
END;

END_PROGRAM;
```

3.10.20 Operator: / (Division)

The / operator divides two integer or floating-point numbers.

Example:

```
INCLUDE rtcu.inc

VAR
   a : INT;
   b : INT;
END_VAR;

PROGRAM test;

BEGIN
   ...
   a := b / 10;
   ...
END;

END_PROGRAM;
```

3.10.21 Operator: + (Addition)

The + operator adds two integer or floating-point numbers or strings together and returns the result.

Using the '+' operator is much easier and more efficient than using the strConcat() function. See example 2 below.

Example 1:

```
INCLUDE rtcu.inc

VAR
   a : INT;
   b : INT;
END_VAR;

PROGRAM test;

BEGIN
   ...
   a := b + 10;
   ...
END;

END_PROGRAM;
```

Example 2 :

```
INCLUDE rtcu.inc

VAR
   a : STRING;
   b : STRING;
   c : STRING;
END_VAR;

PROGRAM test;

   a := "Hello";
   b := "World";
   c := a + " big " + b;   // c will be equal to "Hello big World".
   ...
END_PROGRAM;
```

3.10.22 Operator: - (Subtraction/Negation)

The - operator subtracts two integer or floating-point numbers and returns the result. It is also used to change the sign of a number (see below).

Example:

```
INCLUDE rtcu.inc

VAR
   a : INT;
   b : INT;
END_VAR;

PROGRAM test;

BEGIN
   ...
   a := b - 10; // Subtract 10 from b and assign value to a
   ...
   b := -b; // Change sign of b
   ...
END;

END_PROGRAM;
```

3.10.23 Operator: * (Multiplication)

The * operator multiplies two integer or floating-point numbers and returns the result.

Example:

```
INCLUDE rtcu.inc

VAR
   a : INT;
   b : INT;
END_VAR;

PROGRAM test;

BEGIN
   ...
   a := b * 4;
   ...
END;

END_PROGRAM;
```

3.10.24 Operator: = (Equal)

The = operator compares two numbers or strings and returns TRUE if they are equal.
Care must be taken when used with floating-point numbers due to rounding errors, e.g. `1.1 + 2.2 <> 3.3`.

The comparison is case-sensitive.
See example 2 below.

Example 1:

```
INCLUDE rtcu.inc

VAR
   a : INT;
   b : INT;
   t : BOOL;
END_VAR;

PROGRAM test;

BEGIN
   ...
   IF a = b THEN
      // Will be executed if a equals b
      ...
   END_IF;
   ...
   t := b = 10; // t is TRUE if b equals 10
   ...
END;

END_PROGRAM;
```

Example 2:

```
INCLUDE rtcu.inc

VAR
   a : STRING := "Peter";
   b : STRING := "Jack";
END_VAR;

PROGRAM test;

BEGIN
   ...
   IF a = b THEN
      // Will not be executed as "Peter" is not equal to "Jack"
      ...
   END_IF;
END;

END_PROGRAM;
```

3.10.25 Operator: ** (Exponent)

The ** operator will calculate the exponential value.

Example:

```
INCLUDE rtcu.inc

VAR
   a : INT;
   b : INT;
END_VAR;

PROGRAM test;

BEGIN
   ...
   a := 3;
   b := a ** 4; // The result will be 3 * 3 * 3 * 3 = 81
   ...
END;

END_PROGRAM;
```

3.10.26 Operator: <> (Not equal)

The <> operator compares two numbers or strings and returns TRUE if they are not equal.

The comparison is case-sensitive.
See example 2 below.

Example 1:

```
INCLUDE rtcu.inc

VAR
   a : INT;
   b : INT;
   t : BOOL;
END_VAR;

PROGRAM test;

BEGIN
   ...
   IF a <> b THEN
      // Will be executed if a is different from b
      ...
   END_IF;
   ...
   t := b <> 10; // t is TRUE if b is not equal to 10
   ...
END;

END_PROGRAM;
```

Example 2 :

```
INCLUDE rtcu.inc

VAR
   a : STRING := "Peter";
   b : STRING := "Jack";
END_VAR;

PROGRAM test;

BEGIN
   ...
   IF a <> b THEN
      // Will be executed as "Peter" is not equal to "Jack"
      ...
   END_IF;
END;

END_PROGRAM;
```

3.10.27 Operator: < and > (Less than, Greater than)

The < operator compares two numbers or strings and returns TRUE if the left expression is smaller than the right expression.
The > operator compares two numbers or strings and returns TRUE if the left expression is greater than the right expression.

The comparison is case-sensitive.
See example 2 below.

Example 1:

```
INCLUDE rtcu.inc

VAR
   a : INT;
   b : INT;
   t : BOOL;
END_VAR;

PROGRAM test;

BEGIN
   ...
   IF a > b THEN
      // Will be executed if a is greater than b
      ...
   END_IF;
   ...
   IF a < b THEN
      // Will be executed if a is lesser than b
      ...
   END_IF;
   ...
   t := b < 10; // t is TRUE if b is lesser than 10
   ...
END;

END_PROGRAM;
```

Example 2 :

```
INCLUDE rtcu.inc

VAR
   a : STRING := "Peter";
   b : STRING := "Jack";
   t : BOOL;
END_VAR;

PROGRAM test;

BEGIN
   ...
   IF a < b THEN
      // Will not be executed as "Peter" is not less than "Jack"
      ...
   END_IF;
   ...
   t := a > b; // t will be TRUE as "Peter" > "Jack"
```

```
    ...
END;

END_PROGRAM;
```

3.10.28 Operator: <= and >= (Less than or equal, Greater than or equal)

The <= operator compares two numbers or strings and returns TRUE if the left expression is smaller than or equal to the right expression.
The >= operator compares two numbers or strings and returns TRUE if the left expression is greater than or equal to the right expression.

The comparison is case-sensitive.
See example 2 below.

Example 1:

```
INCLUDE rtcu.inc

VAR
   a : INT;
   b : INT;
   t : BOOL;
END_VAR;

PROGRAM test;

BEGIN
   ...
   IF a >= b THEN
      // Will be executed if a is greater than or equal to b
      ...
   END_IF;
   ...
   IF a <= b THEN
      // Will be executed if a is smaller than or equal to b
      ...
   END_IF;
   ...
   t := b <= 10; // t is TRUE if b is smaller than or equal to 10
   ...
END;

END_PROGRAM;
```

Example 2:

```
INCLUDE rtcu.inc

VAR
   a : STRING := "Peter";
   b : STRING := "Jack";
   t : BOOL;
END_VAR;

PROGRAM test;

BEGIN
   ...
   IF a <= b THEN
      // Will not be executed as "Peter" is not less than or equal to "Jack"
```

```
        ...
    END_IF;
    ...
    t := a >= b; // t will be TRUE as "Peter" >= "Jack"
    ...
END;

END_PROGRAM;
```

3.10.29 Parenthesis ()

The brackets () are used in expressions to force a specific order of evaluation.

Example:

```
INCLUDE rtcu.inc

VAR
   a : INT;
   b : INT;
END_VAR;

PROGRAM test;

BEGIN
   ...
   a := 2 + 3 * 4; // Result will be 14, as * (multiply) has higher priority
that +
   a := (2 + 3) * 4; // Result will be 20 as the 2+3 is given highest priority
because of the () brackets
   ...
END;

END_PROGRAM;
```

3.10.30 Evaluation of functions

Below is an example of the evaluation of a function.

Example:

```
INCLUDE rtcu.inc

FUNCTION Add : INT;

VAR_INPUT
   a : INT;
   b : INT;
   c : INT;
END_VAR;

   Add := a+b+c; // Make the our function return the sum of the 3 numbers

END_FUNCTION;


VAR
   a : INT;
END_VAR;

PROGRAM test;

BEGIN

   a:= Add(a:=3, b:=4, c:=10); // 'a' will be 17 after the call

END;

END_PROGRAM;
```

3.11 Conditional compilation and constants

3.11.1 Conditional compilation

Introduction

The conditional compilation directives allow sections of code to be selectively included for or excluded from compilation depending on whether the programmer-specified symbol (name) is defined or not. It is usually used as a portability tool for tailoring the program code to different hardware variants or to build multiple versions of an application with different functionality.

Another common use of conditional compilation is for temporarily omitting code. This is often done during testing and debugging when the programmer is experimenting with suspected areas of code. Although code may also be omitted by commenting its out, this approach does not work if the code already contains comments because such comments cannot be nested.

A conditional compilation name can either be defined with the **#DEFINE** directive or it can be defined in the Project Settings.
Both defining a name by using the **#DEFINE** directive or by doing it from the Project Settings have the same effect, but the latter is a more user-friendly approach that does not require modification or even access to the VPL source code itself (encrypted include file).

A conditional compilation name consists of up to 80 letters and numbers.

The following shows some examples of the usage of conditional compilation:

Example 1

```
#DEFINE DEBUG

#IFDEF DEBUG THEN
    ... this code will be included in the compilation.
#END_IF
```

Example 2

```
#DEFINE DEBUG

#IFDEF DEBUG THEN
    ... this code will be included in the compilation.
#ELSE
    ... this code will NOT be included in the compilation.
#END_IF
```

Example 3

```
#DEFINE DEBUG

#IFDEF NOT DEBUG THEN
    ... this code will NOT be included in the compilation.
#ELSE
    ... this code will be included in the compilation.
#END_IF
```

Example 4

```
#DEFINE BETA

#IFDEF BETA OR ALFA THEN
    ... this code will be included in the compilation.
#END_IF

#IFDEF ALFA THEN
    ... this code will not be included in the compilation.
#END_IF

#UNDEFINE BETA

#IFDEF BETA THEN
    ... this code will NOT be included in the compilation.
#END_IF
```

Example 5

```
#DEFINE BETA1
#DEFINE BETA2

#IFDEF BETA1 OR BETA3 THEN
    ... this code will be included in the compilation.
#END_IF

#IFDEF ALFA OR BETA1 THEN
    ... this code will be included in the compilation.
#END_IF

#IFDEF ALFA OR NOT BETA2  THEN
    ... this code will NOT be included in the compilation.
#END_IF
```

3.11.2 Macro constants

By using the #DEFINE macro functionality, it is possible to implement constant values and constant strings that can be referenced with symbolic names in the program. This increases program maintainability and reduces programming errors as a given value will only need to be changed in one central location instead of being scattered throughout the entire program code.

Constants can be very useful in VPL programming whenever you have any value that is repeated in your program. Declaring a constant allows one to quickly and easily change a value that is used throughout your code simply by changing the declaration.

Declaring constants

```
#DEFINE MY_NAME     "Candy Girl"
#DEFINE AGE         18
```

The #DEFINE directive is followed by the name of the symbols being defined - MY_NAME and AGE.
These symbols are named like variables, although using all caps for constants allows one to easily identify constants versus variables in your source code. The symbol must all be one word. Following the symbol is a space and then the value that the symbol represents - in this case the string "Candy Girl" and value "18".

Using constants in your code

The example below demonstrates the use of the constants declared earlier:

```
INCLUDE rtcu.inc

PROGRAM test;

DebugMsg(message:="My name is "+MY_NAME);
DebugFmt(message:="My age is \1 years",v1:=AGE);

END_PROGRAM;
```

This simple example will write out the debug messages: "My name is Candy Girl" and "My age is 18 years".

3.11.3 #DEFINE

The **#DEFINE** compiler directive defines a conditional compilation symbol that can be used in a #IFDEF statement to selectively include or omit sections of a program. Using **#DEFINE** makes it possible to define Macro constants as well.
A symbol can also be defined externally from the source code itself by using the Project Settings dialog.

For more information, please see the general Conditional compilation section and the Macro constants section.

3.11.4 #UNDEFINE

The **#UNDEFINE** compiler directive will undefine (remove) a conditional compilation symbol that may previously have been defined with the #DEFINE directive. Undefining a symbol that was not previously defined has no effect.

For more information, please see the general Conditional compilation section.

3.11.5 #IFDEF

The **#IFDEF** compiler directive is used to include or omit sections of code depending on whether a specific conditional compilation symbol is defined or not.

The general form of **#IFDEF** is:

#IFDEF [**NOT**] <name> { **OR** <name> }**THEN**
…
#END_IF

For more information, please see the general Conditional compilation section.

3.11.6 #ELSE

The **#ELSE** compiler directive is used in combination with #IFDEF to include a section of code in cases where the #IFDEF symbol is not defined.

The general form is:

```
#IFDEF <name> THEN
    ...
#ELSE
    ...
#END_IF
```

For more information, please see the general Conditional compilation section.

3.11.7 #END_IF

The **#END_IF** compiler directive is used in combination with #IFDEF to end the conditional section of code.

The general form is:

```
#IFDEF ..... THEN
    ...
#END_IF
```

For more information, please see the general Conditional compilation section.

3.12 Command Line Tools

3.12.1 Command Line Tools

The RTCU IDE is a true integrated development environment, where all tasks conveniently can be performed without requiring any additional tools.
For specialised or advanced build environments, it is, however, also possible to call certain tools from a command line directly. This can be usable from for example a MAKEFILE or other automated tools.

The following tools are included in the RTCU IDE installation directory:

- VPC.EXE — Compiler
- VCFG.EXE — Configurator
- VCFGIO.EXE — I/O Extension Configurator
- RPCTOOL.EXE — Manipulate RTCU Project Containers
- VINF.EXE — I/O Information tool.

All tools will write out usage information when invoked without any parameters.

Example batch files for building projects:

Using the I/O Extension, Master mode example.

Building an X32 project:

```
REM Compile VPL file, creates main.vpx
vpc main.vpl
REM Configure project to create main.vsx
vcfg -a=main,100 main.cfg
REM Add IO extension data
vcfgio main.vsx main.prj
REM done
```

Building an NX32 project:

```
REM Compile VPL file, creates main.vpx
vpc -nx main.vpl
REM Configure project to create main.vsx
vcfg -nx -a=main,100 main.cfg
REM Add IO extension data
vcfgio main.vsx main.prj
REM Create a new RPC file with main.vsx as the default application.
rpctool -n -a main.vsx DEFAULT.VSX main.rpc
REM done
```

Building an NX32L project:

```
REM Compile VPL file, creates main.vpx
vpc -nxl main.vpl
REM Configure project to create main.vsx
vcfg -nxl -a=main,100 main.cfg
REM Add IO extension data
vcfgio main.vsx main.prj
REM Create a new RPC file with main.vsx as the default application and
REM _config_ as the configuration file for flex input.
rpctool -nxl -n -a _config_ _CONFIG_ -a main.vsx DEFAULT.VSX main.rpc
REM done
```

For additional questions about the use of the above tools, please contact Logic IO.

3.12.2 VPC.EXE - Compiler

VPC.EXE compiles a VPL file into a binary object file with the extension **.VPX**, that can be assembled with the configurator tool.
Additionally, the following text files are generated:

- **.LST** - A list file with the generated assembler p-code and data (optional).
- **.VPI** - Input/output information files containing information about the I/O's of the program.

The following compiler options are supported:

-dbg	:	Embed debug information for fault localization into the object file.
-nx	:	Compile for the NX32 architecture.
-nxl		Compile for the NX32L architecture.
-dm		Use double precision math library.
-lss	:	Compile with large string support enabled.
-d=<sym>{,<sym>}	:	Defines symbol <sym>. Can be a list separated with ','.
-d<sym>=<val>		Define symbol <sym> with value <val>. This command can be repeated. The type of <val> will be auto-determined, but can be forced to a STRING by prepending with '
-lst		Generate list file (.LST). Useful when using the RTCU M2M Platform SDK.

By default the compiler targets the X32 architecture.

3.12.3 VCFG.EXE - Configurator

The configurator generates the actual target execution image, by linking a number of **.VPX** files according to the specification of a **.CFG file**.
The configuration file **.CFG**, is a text file automatically generated by the RTCU IDE, but can also be edited or created manually. Basically, the configuration file describes the jobs defined and the physical I/O mapping, as defined in the Job configuration dialog.
The output of the configurator is a **.VSX** file.
If I/O extensions are used in the project, VCFGIO.EXE must be used to add information about the extensions to the **.VSX** file..

For X32 projects, the **.VSX** file is the image file that is to be transferred to the RTCU device or executed by the RTCU Emulator.
For NX32 and NX32L projects, the **.VSX** file must be added to a container using RPCTOOL.EXE.

The following configurator options are supported:

-nx	:	Build for the NX32 architecture.
-nxl		Build for the NX32L architecture.
-e[0,1,2,3,4]		Specifies use of Enhanced Memory from 0=0x10000 to 4=0x40000. Only used for NX32 builds.
-a=<name>,<ver>	:	Sets the application name / version. Version is scaled by 100 so that version 3.20 becomes 320.
-l	:	Build for LARGE EIS architecture (legacy).
-lss	:	Build with large string support enabled.

By default the configurator targets the X32 architecture.

3.12.4 VCFGIO.EXE - I/O Extension Configurator

The I/O Extension Configurator is used to add information about any I/O extensions present in a project, to a VSX file generated with the VCFG.EXE tool.
For a project without I/O extensions, calling VCFGIO.EXE is not required.

Invoking the I/O Extension Configurator requires specifying the name of the VSX file and the name of the PRJ file. On a successful invocation, the VSX file will include the I/O configuration as defined by the PRJ file.

3.12.5 RPCTOOL.EXE - Manipulate RTCU Project Containers

The RTCU Project Container files (**.RPC** files) are image files containing the application and additional data for NX32 and NX32L projects.
It is the **.RPC** files that are transferred to the RTCU device or executed by the RTCU Emulator.

This tool can be used to create, extract and modify **.RPC** files containing the data for the Intellisync Project Drive. NX32L **.RPC** files also has support for files on the internal drive and for RTCU Platform Extensions.

The path to the .RPC file must be provided as the last entry on the command line for all options except -h.

The following basic options are available:

-h	:	Show the help message with information on all the supported options.
-e <*folder*>	:	Extract the project drive from the container to the specified folder. If the folder does not exist, it is created.
-c <*folder*>	:	Creates a container with a project drive based on the contents of the provided folder. Has support for using the wild-card * to limit the files to include. Directories are only included when -nxl is used. Hidden files will not be included.
-n	:	Create a new container instead of modifying an existing container. Can only be used in combination with the modification actions found in the tables below.
-i	:	Show the contents of the container.
-nxl	:	This will make the container use the NX32L format. It must be provided when modifying an NX32L container.
-fnxl	:	This will convert the container to the NX32L format, It also activates the -nxl flag.
-sn	:	Set the name of the container. It will be shown when printing the info about the container. Max 20 characters. This would typically be the application name.
-sv	:	Set the version of the container. It must be and integer between 1 and 30000. It will typically be scaled by 100, i.e. a version of 123 could be shown as 1.23. This would typically be the application version.

The following options are used to modify the container. They can be used multiple times and can be combined on the same command line.

Project drive

Options for both NX32 and NX32L containers:

-a <*file*> <*name*>	:	Add the provided file to the project drive, using the specified name as the file name on the project drive.
-d <*name*>	:	Delete the file with the provided name from the project drive
-u <*file*> <*name*>	:	Replace the file with the specified name on the project drive with the provided file.

Options for NX32L containers:

-af <*name*>	:	Creates a folder with the specified name inside the project drive.
-df <*name*>	:	Deletes the folder with the provided name from the project drive

Internal drive

NX32L containers makes it possible to add and remove files and folders on the internal drive.

It is done by creating actions that will be performed on the device, such as adding or deleting a file.
Files that do not have any actions will not be modified.

Options for NX32L containers:

-ai <*file*> <*name*>	:	Creates an action to add the file at the specified path to the internal drive, using the specified name as the file name on the internal drive. The installation will fail if the file already exists, so it is normally better to use -ui.
-di <*name*>	:	Creates an action to delete the specified file from the internal drive.
-ui <*file*> <*name*>	:	Creates an action to replace the file with the specified name on the internal drive with the provided file.
-ri <*name*>	:	Removes all actions to modify the file with the specified name on the internal drive.
-aif <*name*>	:	Creates an action to create a folder with the specified name on the internal drive.
-dif <*name*>	:	Creates an action to remove the folder with the specified name from the internal drive. Note that only empty folders can be deleted.
-rif <*name*>	:	Removes all actions to modify the specified folder on the internal drive.

RTCU Platform extensions

The RTCU platform extensions installed on a device can be controlled using NX32L containers by creating actions to install and uninstall the extensions.

Options for NX32L containers:

-ae <*name*>	:	Creates an action to install the RTCU Platform extension with the specified name. The file name of the given file will be used as the name on the device.
-de <*name*>	:	Creates an action to uninstall the RTCU Platform extension with the specified name.
-re <*name*>	:	Removes all actions to modify the specified RTCU Platform extension.

By default the rpctool targets the NX32 architecture.

3.12.6 VINF.EXE - Information tool

The information tool lists information about the I/O's of a specified **.VPX** file to stdout.

3.13 Build events

3.13.1 Introduction

"Build events" is a mechanism that allow for customisation when building a project.
This customisation is achieved in three steps:

1. Each build event runs a batch file unique to the event.
   This ensures that there are no limitations on the custom actions possible.

2. The batch files can be included with a project or with the RTCU IDE.
   This ensures that custom actions can be done both a single project and for all projects.

3. The list of files and other project information is transferred via a temporary build file.
   In addition to the project information, custom actions can add information for later custom actions.
   The build file is created before the first build event and deleted after the last.

The supported build events are Pre-Build and Post-Build.

The "BuildEvent example" application is included to demonstrate a simple use of build events. The application can be found with the other example projects at "My Documents\RTCU Projects\Examples".

3.13.2 Pre-Build

The "Pre-Build event" is the build event that is called before the building of the project is initiated.

When the Pre-Build event is triggered, the RTCU IDE will run the "PreBuild.bat" batch file if present.
The RTCU IDE will look for the batch file - first in the project folder and then in the RTCU IDE install folder.
The "PreBuild.bat" file has 2 parameters: <path to project> and <path to build file>.

<path to project> is the absolute path to the folder where the project is located.
<path to build file> is the absolute path to the build file - including the file name (see Build file format for more info).

Note: any changes made to files in the project will not propagate to files opened in the RTCU IDE.

For example, the PreBuild.bat for the BuildEvent example project looks like this:

```
@ECHO OFF

IncBuild.exe %2
```

3.13.3 Post-Build

The Post-Build event is the build event that is called after the building of the project completes.

When the Post-Build event is triggered, the RTCU IDE will run the "PostBuild.bat" batch file if present.
The RTCU IDE will look for the batch file - first in the project folder and then in the RTCU IDE install folder.
The "PostBuild.bat" file has 3 parameters: <path to project>, <path to build file>, and <error code>

<path to project> is the absolute path to the folder where the project is located.
<path to build file> is the absolute path to the build file - including the file name (see Build file format for more info).
<error code> specifies whether the build was a success or not:

- 0 The build is successfully completed.
- 1 The build failed.

Note: any changes made to files in the project will not propagate to files opened in the RTCU IDE.

3.13.4 Build file format

The build file is a human readable text file that contains information about the project that is being built.
The build file is used to inform Pre-Build and Post-Build events which files are being built and where to find them.

The build file can also easily be expanded with information from the Pre-Build event to the Post-Build event.

The build file has the following parameters:

```
[PROJECT]
prj=<name of the project>
path=<path to the project>

[FILES]
file=<path to VPL file 1>
..
file=<path to VPL file n>
file=<path to INC file 1>
..
file=<path to INC file n>
```

For example, the build file for the BuildEvent example project looks like this:

```
[PROJECT]
prj=BuildEvents
path=C:\Documents and Settings\<USER>\My Documents\RTCU
Projects\Examples\BuildEvent example\

[FILES]
file=.\BuildEvents.vpl
```

Where <USER> is the user name of the current user.

Standard Function Library

4 Standard Function Library

4.1 SFL: Standard functions and functionblocks

4.1.1 SFL: Standard functions and functionblocks

The VPL environment offers a number of functions and function blocks. These are divided into different categories. Please have a look at the following pages to see the individual functions:

- Binary operations — Functions for binary operations.
- Calculation routines — Various calculation routines.
- Channel functions — Channel. Thread communication (multithreading).
- Counter functionblocks — Function blocks for counters.
- Datalogger — Functions for data logging.
- Debugger functions — Functions for debugging the application.
- Edge triggers — Functions for edge triggers.
- Floating-point math functions — Math functions for floating-point operations.
- Miscellaneous — Miscellaneous functions and function blocks.
- Mutex functions — Mutex. Thread synchronization (multithreading).
- Persistent memory — Functions for storing and loading strings from persistent memory.
- Real Time Clock functions — Functions to interact with the Real Time Clock.
- Semaphore functions — Semaphore. Thread synchronization (multithreading).
- String functions — Functions for manipulating strings.
- Timer functions — Function blocks for timers.

4.1.2 SFL: Binary operations

4.1.2.1 SFL: Binary operation functions

The VPL environment contains a number of binary operation functions:

- shr32/16/8 — Bitwise right-shift of a value.
- shl32/16/8 — Bitwise left-shift of a value.
- bcd_to_sint/int/dint — Converts BCD digits to a SINT, INT, or DINT.
- sint_to_bcd/int/dint — Converts a SINT, INT, or DINT to BCD digits.

4.1.2.2 shl32/16/8 (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

The bitwise shift function "shlxx" operates on the SINT, INT, and DINT data types. It will shift the value a number of bits to the left. Bits coming in from the right are always 0.

Input:

in : DINT/INT/SINT
The value to shift.

n : SINT (0..32/16/8)
The number of bits to shift.

Returns: **DINT/INT/SINT**
The resulting value.

Declaration:

```
FUNCTION shl32 : DINT;
VAR_INPUT
   in : DINT;
   n  : SINT;
END_VAR;

FUNCTION shl16 : INT;
VAR_INPUT
   in : INT;
   n  : SINT;
END_VAR;

FUNCTION shl8 : SINT;
VAR_INPUT
   in : SINT;
   n  : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   a : INT;
   b : INT;
END_VAR;
```

```
PROGRAM test;

BEGIN
   a := 16#0FF0;
   // Shift contents of a 4 bits to the left
   b := shl16(in:=a, n:=4);
   // b is now 16#FF00
   ...
END;

END_PROGRAM;
```

4.1.2.3 shr32/16/8 (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

The bitwise shift function "shrxx" operates on the SINT, INT, and DINT data types. It will shift the value a number of bits to the right. Bits coming in from left are always 0.

Input:

in : DINT/INT/SINT
The value to shift.

n : SINT (0..32/16/8)
The number of bits to shift..

Returns: DINT/INT/SINT

The resulting value.

Declaration:

```
FUNCTION shr32 : DINT;
VAR_INPUT
   in : DINT;
   n  : SINT;
END_VAR;

FUNCTION shr16 : INT;
VAR_INPUT
   in : INT;
   n  : SINT;
END_VAR;

FUNCTION shr8 : SINT;
VAR_INPUT
   in : SINT;
   n  : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   a : INT;
   b : INT;
END_VAR;
```

```
PROGRAM test;

BEGIN
   a := 16#FF00;
   // Shift contents of a 4 bits to the right
   b := shr16(in:=a, n:=4);
   // b is now 16#0FF0
   ...
END;

END_PROGRAM;
```

4.1.2.4 bcd_to_sint/int/dint (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

The "bcd_to_xxx" function operates on the SINT, INT, and DINT data types. It will convert a BCD-packed value into its binary representation.

Input:

in : DINT/INT/SINT
The BCD-packed value to convert.

Returns: DINT/INT/SINT

The resulting binary value.

Declaration:

```
FUNCTION bcd_to_dint : DINT;
VAR_INPUT
   in : DINT;
END_VAR;

FUNCTION bcd_to_int : INT;
VAR_INPUT
   in : INT;
END_VAR;

FUNCTION bcd_to_sint : SINT;
VAR_INPUT
   in : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   a : INT;
   b : INT;
END_VAR;

PROGRAM test;

BEGIN
   a := 16#2215;
   // Convert 16#2215 (BCD representation), to 2215 in binary
   b := bcd_to_int(in:=a);
   // b is now 2215
```

```
   ...
END;

END_PROGRAM;
```

4.1.2.5 sint_to_bcd/int/dint (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

The "xxx_to_bcd function" operates on the SINT, INT, and DINT data types. It will convert a value to its BCD representation.

Input:

in : DINT/INT/SINT
The binary value to convert.
Note that only a positive number can be converted.

Returns: DINT/INT/SINT

The resulting BCD-packed value.

Declaration:

```
FUNCTION dint_to_bcd : DINT;
VAR_INPUT
   in  : DINT;
END_VAR;

FUNCTION int_to_bcd : INT;
VAR_INPUT
   in  : INT;
END_VAR;

FUNCTION sint_to_bcd : SINT;
VAR_INPUT
   in  : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   a : INT;
   b : INT;
END_VAR;

PROGRAM test;

BEGIN
   a := 1234;
   // Convert 1234 in binary to its BCD representation (16#1234)
   b := int_to_bcd(in:=a);
   // b is now 16#1234
   ...
END;

END_PROGRAM;
```

4.1.3 SFL: Calculation routines

4.1.3.1 SFL: Calculation routines

4.1.3.2 abs (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 3.10 / 1.00.00

This function will calculate the absolute value of a DINT.

Input:
V : DINT
Value to convert.

Returns: **DINT**
The absolute value of V.

Declaration:
```
FUNCTION abs : DINT;
VAR_INPUT
   v : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

DebugFmt(message := " abs( -123) = \4", v4 := abs( v:= -123));

END_PROGRAM;
```

4.1.3.3 min (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

The min function takes two numbers as parameters and returns the value of the lowest of the two numbers.

Input:

a : DINT
The first value.

b : DINT
The second value.

Returns: **DINT**
The lowest value..

Declaration:
```
FUNCTION min : DINT;
VAR_INPUT
   a  : DINT;
   b  : DINT;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

VAR
   number1 : INT;
   number2 : INT;
   result  : INT;
END_VAR;

PROGRAM test;

BEGIN
   ...
   result:=INT(min(a:=number1, b:=number2));
   ...
END;

END_PROGRAM;
```

4.1.3.4 max (Function)

Architecture: **X32 / NX32 / NX32L**
Device support: **ALL**
Firmware version: **1.00 / 1.00.00**

The max function takes two numbers as parameters and returns the value of the highest of the two numbers.

Input:
a : DINT
The first value.

b : DINT
The second value.

Returns: **DINT**
The highest value.

Declaration:
```
FUNCTION max : DINT;
VAR_INPUT
   a  : DINT;
```

```
   b  : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   number1 : INT;
   number2 : INT;
   result  : INT;
END_VAR;

PROGRAM test;

BEGIN
   ...
   result:=INT(max(a:=number1, b:=number2));
   ...
END;

END_PROGRAM;
```

4.1.3.5 VolumeHorizontalTank (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

This function will calculate the amount of fluid in a horizontal tank. The calculation is based on the tank possessing uncurved ends and being perfectly circular in shape.

Input:

R : DINT (0..2147483648)
Radius of the tank in millimeters.

L : DINT (0..2147483648)
Length of the tank in millimeters.

H : DINT (0..2147483648)
Height of the fluid in the tank (if "H" equals "R", the tank is completely full) in milliliters.

Returns: DINT

The amount of fluid contained in the tank in milliliters.

Declaration:

```
FUNCTION VolumeHorizontalTank : DINT;
VAR_INPUT
   R : DINT;
   L : DINT;
   H : DINT;
END_VAR;
```

Example:

```
//-------------------------------------------------------------------------
-
// test.vpl, created 2002-02-03 17:56
//
// Test of volume calculations, for horizontal and vertical tanks, with
```

```
straight
// ends (non-curved)
//
//--------------------------------------------------------------------------
-
INCLUDE rtcu.inc

PROGRAM test;

   // Horizontal: Length = 5000 mm, Radius=2000 mm, Height of fluid=2000 mm
   // (tank is 5 meters long and 4 meters diameter, half full)
   // Should be 31415926 (PI) ml (31.4 mill liters !)
   DebugFmt(message:="Horizontal volume (31415926 ml) = \4 ml",
v4:=VolumeHorizontalTank(H:=2000, L:=5000, R:=2000));

   // Horizontal: Length = 1000 mm, Radius=500 mm, Height of fluid=500 mm
   // Tank is half full, should be 392699.0817 ml
   DebugFmt(message:="Horizontal volume (392699 ml) = \4 ml",
v4:=VolumeHorizontalTank(H:=500, L:=1000, R:=500));

   // Horizontal: Length = 1000 mm, Radius=500 mm, Height of fluid=1000 mm
   // Tank is full, should be 785398.1634 ml
   DebugFmt(message:="Horizontal volume (785398 ml) = \4 ml",
v4:=VolumeHorizontalTank(H:=1000, L:=1000, R:=500));

   // Vertical: Radius=500 mm, Height of fluid=500 mm
   // Should be 392699.0817 ml
   DebugFmt(message:="Vertical volume (392699 ml) = \4 ml",
v4:=VolumeVerticalTank(H:=500, R:=500));

BEGIN

END;

END_PROGRAM;
```

4.1.3.6 VolumeVerticalTank (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

This function will calculate the amount of fluid in a vertical tank. The calculation is based on the tank possessing uncurved ends and being perfectly circular in shape.

Input:

R : DINT (0..2147483648)
Radius of the tank in millimeters.

H : DINT (0..2147483648)
Height of the fluid in the tank in millimeters

Returns: DINT

The amount of fluid contained in the tank in milliliters.

Declaration:

```
FUNCTION VolumeVerticalTank : DINT;
VAR_INPUT
   R : DINT;
   H : DINT;
END_VAR;
```

Example:

```
//-------------------------------------------------------------------------
-
// test.vpl, created 2002-02-03 17:56
//
// Test of volume calculations, for horizontal and vertical tanks, with
straight
// ends (non-curved)
//
//-------------------------------------------------------------------------
-
INCLUDE rtcu.inc

PROGRAM test;

   // Horizontal: Length = 5000 mm, Radius=2000 mm, Height of fluid=2000 mm
   // (tank is 5 meters long and 4 meters diameter, half full)
   // Should be 31415926 (PI) ml (31.4 mill liters !)
   DebugFmt(message:="Horizontal volume (31415926 ml) = \4 ml",
v4:=VolumeHorizontalTank(H:=2000, L:=5000, R:=2000));

   // Horizontal: Length = 1000 mm, Radius=500 mm, Height of fluid=500 mm
   // Tank is half full, should be 392699.0817 ml
   DebugFmt(message:="Horizontal volume (392699 ml) = \4 ml",
v4:=VolumeHorizontalTank(H:=500, L:=1000, R:=500));

   // Horizontal: Length = 1000 mm, Radius=500 mm, Height of fluid=1000 mm
   // Tank is full, should be 785398.1634 ml
   DebugFmt(message:="Horizontal volume (785398 ml) = \4 ml",
v4:=VolumeHorizontalTank(H:=1000, L:=1000, R:=500));

   // Vertical: Radius=500 mm, Height of fluid=500 mm
   // Should be 392699.0817 ml
   DebugFmt(message:="Vertical volume (392699 ml) = \4 ml",
v4:=VolumeVerticalTank(H:=500, R:=500));

BEGIN

END;

END_PROGRAM;
```

4.1.3.7 crcCalculate (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.08 / 1.00.00

This function calculates, by standard, the CRC-CCITT 16 (16 bit polynomial) checksum of a buffer.
By using the CRC-CCITT 16 polynomial, an overall error detection rate of 99.955% is achieved.

By adjusting the parameters, a wide range of CRC polynomials/algorithms can be realised. Please consult relevant literature for further information.

Input:

buffer : PTR
Pointer to the buffer that is used to calculate the CRC.

length : DINT
Number of bytes in the buffer.

polynom : DINT (Default: 16#00001021)
The polynom used to calculate the CRC.
Default is X^16 + X^12 + X^5 + 1.

preset : DINT (Default: 16#0000FFFF)
The preset CRC value.

finalxor : DINT (Default: 16#00000000)
This value is XOR'ed with the CRC calculated from the buffer just before it is returned.

order : SINT (1..32, Default: 16)
The number of bits used in the CRC.

direct : BOOL (Default: TRUE)
Calculates the CRC with (FALSE) or without (TRUE) augmented zero bits.

reversedata : BOOL (Default: FALSE)
Reverses data bytes (TRUE) before CRC is calculated.

reversecrc : BOOL (Default: FALSE)
Reverse calculated CRC (TRUE) before finalxor is applied.

Returns: DINT
The calculated CRC of the buffer.

Declaration:
```
FUNCTION crcCalculate : DINT;
VAR_INPUT
   // Data buffer
   buffer      : PTR;
   length      : DINT;
   // CRC algorithm parameters
   polynom     : DINT := 16#00001021;
   preset      : DINT := 16#0000FFFF;
   finalxor    : DINT := 16#00000000;
   order       : SINT := 16;
   direct      : BOOL := TRUE;
   reversedata : BOOL := FALSE;
   reversecrc  : BOOL := FALSE;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   buffer : ARRAY[1..9] OF SINT;
   crc16  : INT;
   crc32  : DINT;
END_VAR;

   // Test data
   strToMemory(dst:=ADDR(buffer), str:="123456789", len:=9);

   // Calculate CRC-CCITT16
```

```
    DebugMsg(message:="****************************************");
    DebugMsg(message:="CALC_CRC: (CRC-CCITT16)");
    crc16 := INT( crcCalculate(buffer:=ADDR(buffer), length:=9) );
    DebugFmt(message:="-CRC = \1", v1:=crc16);

    DebugMsg(message:="****************************************");
    DebugMsg(message:="CALC_CRC: (CRC16)");
    DebugMsg(message:="-Order = 16");
    DebugMsg(message:="-Polynom = 16#00008005");
    DebugMsg(message:="-Preset = 16#00000000");
    DebugMsg(message:="-Direct = TRUE");
    DebugMsg(message:="-Finalxor = 16#00000000");
    DebugMsg(message:="-ReverseData = TRUE");
    DebugMsg(message:="-ReverseCRC = TRUE");
    crc16 := INT(
                crcCalculate(
                             buffer:=ADDR(buffer),
                             length:=9,
                             Order:=16,
                             Polynom:=16#00008005,
                             Preset:=0,
                             Direct:=TRUE,
                             Finalxor:=0,
                             ReverseData:=TRUE,
                             ReverseCRC:=TRUE
                            )
                );
    DebugFmt(message:="-CRC = \1", v1:=crc16);

    DebugMsg(message:="****************************************");
    DebugMsg(message:="CALC_CRC: (CRC32)");
    DebugMsg(message:="-Order = 32");
    DebugMsg(message:="-Polynom = 16#04C11DB7");
    DebugMsg(message:="-Preset = 16#FFFFFFFF");
    DebugMsg(message:="-Direct = TRUE");
    DebugMsg(message:="-Finalxor = 16#FFFFFFFF");
    DebugMsg(message:="-ReverseData = TRUE");
    DebugMsg(message:="-ReverseCRC = TRUE");
    crc32 := crcCalculate(
                          buffer:=ADDR(buffer),
                          length:=9,
                          Order:=32,
                          Polynom:=16#04C11DB7,
                          Preset:=16#FFFFFFFF,
                          Direct:=TRUE,
                          Finalxor:=16#FFFFFFFF,
                          ReverseData:=TRUE,
                          ReverseCRC:=TRUE
                         );
    DebugFmt(message:="-CRC = \4", v4:=crc32);

BEGIN
END;

END_PROGRAM;
```

4.1.3.8 random (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.20 / 1.00.00

This function will generate a random number within a specified range.

The random number generated by this function must be in the range -2147483647 to +2147483647.
Invalid parameters will return 0.

Input:

lower : DINT
Lower bound for the random number to be generated.

upper : DINT
Upper bound for the random number to be generated.

Returns: DINT

Random number X, where: lower <= X <= upper.

Declaration:

```
FUNCTION random : DINT;
VAR_INPUT
   lower : DINT;
   upper : DINT;
END_VAR;
```

Example:

```
//--------------------------------------------------------------------------
-
// test.vpl, created 2008-03-03 17:56
//
// Generate and print out random numbers.
//
//--------------------------------------------------------------------------
-
INCLUDE rtcu.inc

PROGRAM test;

   DebugFmt(message:="The dice shows: \4", v4:=random(lower:=1,upper:=6));
   DebugFmt(message:="Your lucky number is: \4",
v4:=random(lower:=10,upper:=20));

END_PROGRAM;
```

4.1.3.9 calcPow (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 2.32 / 1.00.00

The "calcPow" function calculates the power functions of two values given as scaled floating point values.

The values "x" and "y" are integers interpreted as floating point with the scaling factor given in the "scale" parameter.
When, for example, (x=812 y=215, scale=100), this is interpreted as (x=8.12, y=2.15), and the returned value from calcPow will be 9027 which is equal to 90.27 with the specified scaling.

For large values overflow may occur in the calculation, and the returned value will be undefined.
For calculations yielding a complex number, 0 will be returned.

It is also possible to use floating-point with the exponent operator.

Input:

x : DINT
Scaled floating point number.

y : DINT
Scaled floating point number.

scale : INT (*default 1*)
The scale that is used for the floating point numbers to convert them to integers.

Returns: DINT

The resulting scaled floating point number.

Declaration:

```
FUNCTION calcPow : DINT;
VAR_INPUT
   x     : DINT;
   y     : DINT;
   scale : INT := 1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   res : DINT;
END_VAR;

   res := calcPow(x := 200, y := 247, scale := 100);
   DebugFmt(message := "Power function: 2.00 ** 2.47 = \4.\1", v4 := res, v1
:= INT(res / 100));

BEGIN
END;
END_PROGRAM;
```

4.1.3.10 rdDecrypt128 (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 2.64 / 1.00.00

The rdDecrypt128 function will decrypt data by using Rijndael encryption algorithm with a 128 bit key.
The function decrypts the data in blocks of 16 bytes. If the data is not a multiple of 16, the result of the operation is undefined.

Input:

key : PTR
Array containing the key used to decrypt the data.
Length of data must be 16 bytes = 128 bits.

buf_in : PTR
Buffer containing the encrypted data to decrypt.

buf_out : PTR
Buffer where the decrypted data is stored.

out_len : INT
Size of "buf_out" in number of bytes.

blk : INT
The number of block from the raw data to decrypt. Each block is 16 bytes long.

Returns: INT

0	Data is decrypted.
-1	Input buffer and output buffer cannot be the same.
-2	Invalid number of blocks.
-3	Encryption key is missing.
-4	Input buffer is missing.
-5	Output buffer is missing.
-6	"buf_out" buffer too small for decrypted data.

Declaration:

```
FUNCTION rdDecrypt128 : INT;
VAR_INPUT
   key     : PTR;
   buf_in  : PTR;
   buf_out : PTR;
   out_len : INT;
   blk     : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   key   : ARRAY [1..16] OF SINT;
   src   : ARRAY [1..64] OF SINT;
   crypt : ARRAY [1..64] OF SINT;
   dst   : ARRAY [1..64] OF SINT;
END_VAR;

PROGRAM example;
VAR
   len : INT;
   str : STRING;
END_VAR;

   // Setup key
   key[01] := 16#F4;   key[02] := 16#98;   key[03] := 16#D8;   key[04] :=
16#16;
   key[05] := 16#22;   key[06] := 16#4F;   key[07] := 16#47;   key[08] :=
16#3F;
   key[09] := 16#91;   key[10] := 16#85;   key[11] := 16#96;   key[12] :=
16#B8;
   key[13] := 16#DD;   key[14] := 16#57;   key[15] := 16#51;   key[16] :=
16#BE;

   // Encrypt a string
   str := "This is a test of encryption";
   DebugMsg(message := "Original= " + str);
   strToMemory(dst := ADDR(src[2]), str := str, len := strLen(str := str));
```

```
   src[1] := SINT(strLen(str := str));
   len    := (src[1] + 1) / 16;
   IF (src[1] + 1) MOD 16 > 0 THEN len := len + 1; END_IF;
   rdEncrypt128(key := ADDR(key), buf_in := ADDR(src), buf_out := ADDR(crypt),
out_len := 64, blk := len);

   ...

   // Decrypt a string
   rdDecrypt128(key := ADDR(key), buf_in := ADDR(crypt), buf_out := ADDR(dst),
out_len := 64, blk := len);
   DebugMsg(message := "Decrypted= " + strFromMemory(src := ADDR(dst[2]), len
:= dst[1]));

END_PROGRAM;
```

4.1.3.11 rdEncrypt128 (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 2.64 / 1.00.00

The rdEncrypt128 function will encrypt data by using Rijndael encryption algorithm with a 128 bit key.
The function encrypts the data in blocks of 16 bytes. If the data is not a multiple of 16, the result of the operation is undefined.

Input:

key : PTR
Array containing the key used to encrypt the data.
Length of data must be 16 bytes = 128 bits.

buf_in : PTR
Buffer containing the raw data to encrypt.

buf_out : PTR
Buffer where the encrypted data is stored.

out_len : INT
Size of "buf_out" in number of bytes.

blk : INT
The number of block from the raw data to encrypt. Each block is 16 bytes long.

Returns: INT

0	Data is encrypted.
-1	Input buffer and output buffer cannot be the same.
-2	Invalid number of blocks.
-3	Encryption key is missing.
-4	Input buffer is missing.
-5	Output buffer is missing.
-6	"buf_out" buffer too small for decrypted data.

Declaration:

```
FUNCTION rdEncrypt128 : INT;
VAR_INPUT
   key     : PTR;
   buf_in  : PTR;
   buf_out : PTR;
```

```
   out_len : INT;
   blk     : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   key   : ARRAY [1..16] OF SINT;
   src   : ARRAY [1..64] OF SINT;
   crypt : ARRAY [1..64] OF SINT;
   dst   : ARRAY [1..64] OF SINT;
END_VAR;

PROGRAM example;
VAR
   len : INT;
   str : STRING;
END_VAR;

   // Setup key
   key[01] := 16#F4;   key[02] := 16#98;   key[03] := 16#D8;   key[04] :=
16#16;
   key[05] := 16#22;   key[06] := 16#4F;   key[07] := 16#47;   key[08] :=
16#3F;
   key[09] := 16#91;   key[10] := 16#85;   key[11] := 16#96;   key[12] :=
16#B8;
   key[13] := 16#DD;   key[14] := 16#57;   key[15] := 16#51;   key[16] :=
16#BE;

   // Encrypt a string
   str := "This is a test of encryption";
   DebugMsg(message := "Original= " + str);
   strToMemory(dst := ADDR(src[2]), str := str, len := strLen(str := str));
   src[1] := SINT(strLen(str := str));
   len    := (src[1] + 1) / 16;
   IF (src[1] + 1) MOD 16 > 0 THEN len := len + 1; END_IF;
   rdEncrypt128(key := ADDR(key), buf_in := ADDR(src), buf_out := ADDR(crypt),
out_len := 64, blk := len);

   ...

   // Decrypt a string
   rdDecrypt128(key := ADDR(key), buf_in := ADDR(crypt), buf_out := ADDR(dst),
out_len := 64, blk := len);
   DebugMsg(message := "Decrypted= " + strFromMemory(src := ADDR(dst[2]), len
:= dst[1]));

END_PROGRAM;
```

4.1.4 SFL: Channel functions (multithreading)

4.1.4.1 ch: Channel functions

The "Channel" communication mechanism is a FIFO queue mechanism that can be used for passing messages between threads.

A message is defined as an arbitrary block of data with a given length. The channel is initialized with chInit(), and the maximum number of messages in the channel is specified. A message is written to a channel by using chWrite() and read using chRead(). The Channel mechanism will automatically synchronize the access to the channel so that reading from an empty channel will block the calling thread until a message is written. Writing to a full channel will also block the calling thread until a message has been read.

Please refer to the section on Using Multithreading for more information on the use of channels.

The following channel functions exists:

- chInit — Initializes a channel.
- chDestroy — Destroys a channel.
- chStatus — Queries a channel for its status.
- chRead — Reads data from a channel.
- chWrite — Writes data to a channel.
- chPeek — Takes a peek at the first message in the channel.

The following resource limitations apply:

- X32 Execution Architecture: Maximum 16 channels with a total message size of 19 Kbytes.
- NX32 Execution Architecture: Maximum 32 channels with a total message size of 39 Kbytes.
- NX32L Execution Architecture: Maximum 128 channels with a total message size of 256 Kbytes.

The total message size available is for all channels combined. Each message occupies the size of the message + an overhead of 10 bytes.
With a message size of 10 bytes, a total of 13107 messages (256 Kbyte / (10+10)) can be present in an NX32L based system.

4.1.4.2 chInit (Function)

```
Architecture:       X32 / NX32 / NX32L
Device support:     ALL
Firmware version:   1.00 / 1.00.00
```

The" chInit()" function will initialize a CHANNEL variable with the specified maximum number of messages.
The chInit() function must be called before any other operation on the channel can be performed.

Input:

msgmax : INT
Maximum number of messages that can be present in the channel before the chWrite() function will block.

Returns: **CHANNEL**
The initialized channel.

Declaration:

```
FUNCTION chInit : CHANNEL;
VAR_INPUT
   msgmax : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   al_1 : BOOL R_EDGE;
   al_2 : BOOL R_EDGE;
END_VAR;

VAR
   chalarm : CHANNEL;
END_VAR;

// SMS thread
THREAD_BLOCK smsalarm;
VAR
   alarmno : SINT;
END_VAR;

WHILE TRUE DO
   chRead(ch:=chalarm,msg:=ADDR(alarmno),lenmax:=SIZEOF(alarmno));
   gsmSendSMS(phonenumber := "22448899", message := strFormat(format:="Alarm.
Door \1", v1:=alarmno));
END_WHILE;

END_THREAD_BLOCK;


PROGRAM test;
VAR
   smshandler : smsalarm;
   alarm      : sint;
END_VAR;

//main thread
chalarm := chInit(msgMax:=10);
smshandler();
gsmPower(power := TRUE);
BEGIN
   IF al_1 THEN
      alarm:=1;
      chWrite(ch:=chalarm,msg:=ADDR(alarm),len:=SIZEOF(alarm));
   END_IF;
   IF al_2 THEN
      alarm:=2;
      chWrite(ch:=chalarm,msg:=ADDR(alarm),len:=SIZEOF(alarm));
   END_IF;
END;

END_PROGRAM;
```

4.1.4.3 chDestroy (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 (3.10 for the 'flush' parameter) / 1.00.00

The "chDestory()" function will destroy a CHANNEL variable.
After the call to chDestroy(), the channel variable cannot be used in further operations.

Input:

ch : CHANNEL
The channel to destroy.

flush : BOOL ***(default: false)***
When set to TRUE any content in the channel will automatically be disposed.

Returns: INT

-2	Channel is busy.
-1	Channel is not initialized.
0	Channel is destroyed.

Declaration:

```
FUNCTION chDestroy : INT;
VAR_INPUT
   ch : CHANNEL;
   flush : BOOL;
END_VAR;
```

4.1.4.4 chStatus (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

The "chStatus()" function will return the status of the specified channel.
If the specified channel has been initialized, the chStatus() function will return the current number of messages.

Input:

ch : CHANNEL
The channel to query.

Returns: INT

-1	Channel is not initialized.
>0	Number of messages in the channel.

Declaration:

```
FUNCTION chStatus : INT;
VAR_INPUT
   ch : CHANNEL;
END_VAR;
```

Example:

```
THREAD_BLOCK Thread;
VAR_INPUT
   ch : CHANNEL;
   id : SINT;
   ...
```

```
END_VAR;
VAR
   buf : ARRAY[1..30] OF SINT;
   len : INT;
   ...
END_VAR;
   ...
   // Do while channel initialized
   WHILE chStatus(ch:=ch) > -1 DO
      // peek at data
      len := chPeek(ch:=ch,msg:=ADDR(buf),lenmax:=SIZEOF(buf));
      IF len > 0 THEN
         // Is package sent to me?
         IF buf[1] = id THEN
            // Read data
            chRead(ch:=ch,msg:=ADDR(buf),lenmax:=SIZEOF(buf));
            // Act on data
            ...
         END_IF;
      END_IF;
      ...
   END_WHILE;
END_THREAD_BLOCK;
```

4.1.4.5 chRead (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

The "chRead()" function will read a message from the specified channel.
If there is no message present in the channel, the calling thread will be blocked due to waiting for another thread to write a message by using the chWrite() function. Optionally, a timeout for the read operation can be specified.

Also see chWrite() and the section on Using Multithreading.

Special note on writing strings to a channel:
A STRING cannot be written directly to a channel with chWrite() but must be converted into pure memory with the strToMemory() function before being written to a channel. When this message is received by chRead(), it can be converted back to a string again by using the strToMemory() function.

Input:

ch : CHANNEL
The channel to read the message from.

msg : PTR
Pointer to buffer where the message is to be stored.

lenmax : INT
Size of the buffer that *msg* points to.
If the size of the message in the channel is larger than *lenmax,* only *lenmax* bytes will be read. The remaining part of the message will be lost in this case.

timeout : INT *(default: -1)*
The time, in ms, to wait while trying to read data from the channel.
Use -1 to wait forever. This is the default.

Returns: INT

-1 Channel not initialized.

0 Timeout occurred.
>0 Number of bytes actually read from channel.

Declaration:

```
FUNCTION chRead : INT;
VAR_INPUT
   ch      : CHANNEL;
   msg     : PTR;
   lenmax  : INT;
   timeout : INT := -1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   al_1 : BOOL R_EDGE;
   al_2 : BOOL R_EDGE;
END_VAR;

VAR
   chalarm : CHANNEL;
END_VAR;

//SMS thread
THREAD_BLOCK smsalarm;
VAR
   alarmno : SINT;
END_VAR;

WHILE TRUE DO
   chRead(ch:=chalarm,msg:=ADDR(alarmno),lenmax:=SIZEOF(alarmno));
   gsmSendSMS(phonenumber := "22448899", message := strFormat(format:="Alarm.
Door \1", v1:=alarmno));
END_WHILE;

END_THREAD_BLOCK;


PROGRAM test;
VAR
   smshandler : smsalarm;
   alarm      : sint;
END_VAR;

//main thread
chalarm := chInit(msgMax:=10);
smshandler();
gsmPower(power := TRUE);
BEGIN
   IF al_1 THEN
      alarm:=1;
      chWrite(ch:=chalarm,msg:=ADDR(alarm),len:=SIZEOF(alarm));
   END_IF;
   IF al_2 THEN
      alarm:=2;
      chWrite(ch:=chalarm,msg:=ADDR(alarm),len:=SIZEOF(alarm));
   END_IF;
END;

END_PROGRAM;
```

4.1.4.6 chWrite (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

The "chWrite()" function will write a message to the specified channel.
If the number of messages currently in the channel has reached the maximum as specified with chInit(), the calling thread will be blocked due to waiting for another thread to read a message by using the chRead() function. Optionally, a timeout for the write operation can be specified.

Also see chRead() and the section on Using Multithreading.

Special note on writing strings to a channel:
A STRING cannot be written directly to a channel but must be converted into pure memory with the strToMemory() function before being written to a channel. When this message is received by the chRead() function, it can be converted back to a string again by using the strFromMemory() function.

Input:

ch : CHANNEL
The channel to write the message to.

msg : PTR
Pointer to buffer where the message is to be stored.

len : INT
The length of the message.

timeout : INT *(default: -1)*
The time, in ms, to wait before giving up when the channel is full.
Use -1 to wait forever. This is the default.

Returns: INT

-3	Memory allocation error.
-1	Channel not initialized.
0	Timeout.
>0	Number of bytes written to channel. This will always be the same as *len*.

Declaration:

```
FUNCTION chWrite : INT;
VAR_INPUT
   ch       : CHANNEL;
   msg      : PTR;
   len      : INT;
   timeout  : INT := -1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   al_1 : BOOL R_EDGE;
   al_2 : BOOL R_EDGE;
END_VAR;
```

```
VAR
   chalarm : CHANNEL;
END_VAR;

//SMS thread
THREAD_BLOCK smsalarm;
VAR
   alarmno : SINT;
END_VAR;

WHILE TRUE DO
   chRead(ch:=chalarm,msg:=ADDR(alarmno),lenmax:=SIZEOF(alarmno));
   gsmSendSMS(phonenumber := "22448899", message := strFormat(format:="Alarm.
Door \1", v1:=alarmno));
END_WHILE;

END_THREAD_BLOCK;

PROGRAM test;
VAR
   smshandler : smsalarm;
   alarm      : sint;
END_VAR;

//main thread
chalarm := chInit(msgMax:=10);
smshandler();
gsmPower(power := TRUE);
BEGIN
   IF al_1 THEN
      alarm:=1;
      chWrite(ch:=chalarm,msg:=ADDR(alarm),len:=SIZEOF(alarm));
   END_IF;
   IF al_2 THEN
      alarm:=2;
      chWrite(ch:=chalarm,msg:=ADDR(alarm),len:=SIZEOF(alarm));
   END_IF;
END;

END_PROGRAM;
```

4.1.4.7 chPeek (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

The "chPeek()" function works like the chRead() function except for the fact that the message will not be removed from the channel and will never be blocked.

Input:

ch : CHANNEL
The channel to peek the message from.

msg : PTR
Pointer to buffer where the message is to be stored.

lenmax : INT
Size of the buffer that *msg* points to.
If the size of the message in the channel is larger than *lenmax*, only *lenmax* bytes will be read.

Returns: INT

-1 Channel not initialized.
>0 Number of bytes actually read from channel.

Declaration:

```
FUNCTION chPeek : INT;
VAR_INPUT
   ch     : CHANNEL;
   msg    : PTR;
   lenmax : INT;
END_VAR;
```

Example:

```
THREAD_BLOCK Thread;
VAR_INPUT
   ch : CHANNEL;
   id : SINT;
   ...
END_VAR;
VAR
   buf : ARRAY[1..30] OF SINT;
   len : INT;
   ...
END_VAR;
   ...
   // Do while channel initialized
   WHILE chStatus(ch:=ch) > -1 DO
      // peek at data
      len := chPeek(ch:=ch,msg:=ADDR(buf),lenmax:=SIZEOF(buf));
      IF len > 0 THEN
         // Is package sent to me?
         IF buf[1] = id THEN
            // Read data
            chRead(ch:=ch,msg:=ADDR(buf),lenmax:=SIZEOF(buf));
            // Act on data
            ...
         END_IF;
      END_IF;
      ...
   END_WHILE;
END_THREAD_BLOCK;
```

4.1.5 SFL: Counter functionblocks

4.1.5.1 SFL: Counter functionblocks

The VPL environment contains a number of standard function blocks:

- CTD — Down counter.
- CTU — Up counter.
- CTUD — Up/Down counter.
- PCT — High-speed counter.

4.1.5.2 CTD (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

CTD is a Down counter. This function block will count the "cv" value one down on each rising edge on the "cd" input. When the "cv" variable reaches 0, the output "q" will go high - otherwise it will be low. If a high signal is present on the "ld" input, the value from "pv' will be copied to "cv".

Input:

cd : BOOL (true/false)
On the leading edge of this input the "cv" will be decremented with 1. When the "cv" value is equal to 0, "q" will be high

ld : BOOL (true/false)
Load input. When this input is high, the "cv" will set to the value in "pv".

pv : INT (0..32767)
Preset value for the counter.

Output:

cv : INT (0..32767) ***Default -1***
The current value of the counter.

q : BOOL (true/false)
Output from the counter. See description for "cd".

Declaration:

```
FUNCTION_BLOCK CTD;
VAR_INPUT
   cd : BOOL R_EDGE; | Signal that decrements the counter by 1 on a rising
edge
   ld : BOOL;        | Signal that sets 'cv' to the value in 'pv' when high
   pv : INT;         | Preset value for the counter
END_VAR;
VAR_OUTPUT
   q  : BOOL;        | Output from counter
   cv : INT;         | Current counter value
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   input1 : BOOL; | Input that will load the counter with the preset value
```

```
   input2 : BOOL; | Input that will decrement the counter with one on a rising
edge
END_VAR;

VAR_OUTPUT
   signal : BOOL; | Output that will follow the q output from the counter
END_VAR;

VAR
   counter : CTD; // Declare an instance of the CTD functionblock
END_VAR;

PROGRAM test;

BEGIN
   // Call the counter, and assign its input variables
   counter(
           pv := 100,    // Preset value for the counter
           ld := input1, // The counter will be loaded with the preset value
when input1 is high
           cd := input2  // The counter will decrement 'cv' with 1 when rising
edge on input2
          );

   signal:=counter.q; // The output signal will go high when the counter
reaches 0
END;
END_PROGRAM;
```

4.1.5.3 CTU (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

CTU is an Up counter. This function block will increment the "cv" value with one on each rising edge on the "cu" input. When the "cv" variable reaches "pv", or is above, the output "q" will go high - otherwise it will be low. If a high signal is present on the "r" input, the counter will be reset ("cv" will be set to 0).

Input:

cu : BOOL (true/false)
On the leading edge of this input, the "cv" will be incremented with 1. When the "cv" value is equal to, or above, the value for "pv", "q" will be high.

r : BOOL (true/false)
Reset input. When this input is high, the counter value will be kept at 0.

pv : INT (0..32767)
Preset value for the counter.

Output:

cv : INT (-32768..32767)
The current value of the counter.

q : BOOL (true/false)
Output from the counter. See description for "cu".

Declaration:

```
FUNCTION_BLOCK CTU;
VAR_INPUT
```

```
   cu : BOOL R_EDGE; | Signal that increments the counter by 1 on a rising
edge
   r  : BOOL;        | Signal that sets 'cv' to 0 when high
   pv : INT;         | Preset value for the counter
END_VAR;
VAR_OUTPUT
   q  : BOOL;        | Output from counter
   cv : INT;         | Current counter value
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   input1 : BOOL; | Input that will reset the counter
   input2 : BOOL; | Input that will increment the counter with one on a rising
edge
END_VAR;

VAR_OUTPUT
   signal : BOOL; | Output that will follow the q output from the counter
END_VAR;

VAR
   counter : CTU; // Declare an instance of the CTU functionblock
END_VAR;

PROGRAM test;

BEGIN
   // Call the counter, and assign its input variables
   counter(
           pv := 100,    // Preset value for the counter
           r  := input1, // The counter will be reset when input1 is high
           cu := input2  // The counter will increment 'cv' with 1 when rising
edge on input2
          );

   signal:=counter.q; // The output signal will go high when the counter
reaches 'pv'
END;
END_PROGRAM;
```

4.1.5.4 CTUD (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

CTUD is an Up/Down counter. This function block is a mixture between the CTU and CTD function blocks. It combines the functionality of the CTU and CTD to form a flexible Up/Down counter.

This function block will increment the "cv" value with one on each rising edge on the "cu" input, and it will decrement "cv" with 1 when a leading edge on the "cu" is detected. When the "cv" variable reaches "pv", or is above, the output, "qu" will go high - otherwise it will be low. When the "cv" variable reaches 0, the output "q" will go high - otherwise it will be low. If a high signal is present on the "r" input, the counter will be reset ("cv" will be set to 0). If a high signal is present on the "ld" input, the value from "pv" will be copied to "cv".

Input:

cu : BOOL (true/false)
On the leading edge of this input, the "cv" will be incremented with 1. When the "cv" value is equal to, or above, the value in "pv", "qu" will be high.

cd : BOOL (true/false)
On the leading edge of this input, the "cv" will be decremented with 1. When the "cv" value is equal to 0, "qd" will be high

ld : BOOL (true/false)
Load input. When this input is high, the "cv" will set to the value in "pv".

r : BOOL (true/false)
Reset input. When this input is high, the counter value will be kept at 0.

pv : INT (0..32767)
Preset value for the counter.

Output:

cv : INT (0..32767)
The current value of the counter.

qu : BOOL (true/false)
Output from the counter. See description for "cu".

qd : BOOL (true/false)
Output from the counter. See description for "cd".

Declaration:

```
FUNCTION_BLOCK CTUD;
VAR_INPUT
   cu : BOOL R_EDGE; | Signal that increments the counter by 1 on a rising
edge
   cd : BOOL R_EDGE; | Signal that decrements the counter by 1 on a rising
edge
   ld : BOOL;        | Signal that sets 'cv' to the value in 'pv' when high
   r  : BOOL;        | Signal that sets 'cv' to 0 when high
   pv : INT;         | Preset value for the counter
END_VAR;
VAR_OUTPUT
   qu : BOOL;        | Output from counter
   qd : BOOL;        | Output from counter
   cv : INT;         | Current counter value
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   input1 : BOOL; | Input that will reset the counter
   input2 : BOOL; | Input that will increment the counter with one on a rising
edge
   input3 : BOOL; | Input that will load the counter with the preset value
   input4 : BOOL; | Input that will decrement the counter with one on a rising
edge
END_VAR;

VAR_OUTPUT
   signal_u : BOOL; | Output that will follow the 'qu' output from the counter
   signal_d : BOOL; | Output that will follow the 'qd' output from the counter
END_VAR;
```

```
VAR
   counter : CTUD; // Declare an instance of the CTUD functionblock
END_VAR;

PROGRAM test;

BEGIN
   counter(
            pv := 100,     // Preset value for the counter
            r  := input1, // The counter will be reset when input1 is high
            cu := input2, // The counter will increment 'cv' with 1 when rising
edge on input2
            ld := input3, // The counter will be loaded with the preset value
when input3 is high
            cd := input4  // The counter will decrement 'cv' with 1 when rising
edge on input4
           );

   signal_u:=counter.qu; // The two outputs from the counter are copied to
signal_u and signal_d
   signal_d:=counter.qd;
END;
END_PROGRAM;
```

4.1.5.5 PCT (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

PCT is a high-speed pulse counter. This function block will increment the "cv" value with one on each rising edge on the "pinput" input. If a high signal is present on the "ld" input, the counter will be set to the value present on the "ncv". Unlike the normal CTU (Up counter), the PCT counter is realized at a lower level - thereby making it possible to count high-speed pulses.
By using the PCT function block, the pulse counting will occur in the background without being dependent on the actual execution speed of the VPL program.

On NX32 devices a frequency of 400 Hz can be counted on 1 channel, 200 Hz on 2 channels, 133 Hz on 3 channels, and 100 Hz on 4 channels.
The frequency on NX32L devices is up to 13 kHz on all simultaneous channels. Please refer to the technical manual for the specific device in question.
On the RTCU LX4, inputs 6-8 have support for up to 400 Hz on 1 to 3 channels.
NX32L devices can use the mode parameter to debounce the pulses to avoid counting a noisy signal as multiple pulses.

For other high-speed IO functions on NX32L, see the High-Speed IO functions.

Input:

pinput : PTR
Address of a digital input signal (please see the example below).
On the leading edge of this input, the "cv" will be incremented with 1.

Note:
The address can only be a simple variable and will therefore not work with an ARRAY variable.

ld : BOOL (true/false)
Load input. When this input is high, the counter value will be set to the value in "ncv".

ncv : DINT
New counter value. It will be written to "cv" when "ld" is high.

mode: SINT *(default _PCT_T_NONE)* *(requires firmware R2.00.00 or newer)*
Debounce mode. Only supported on NX32L devices.
When enabled, the signal must be active for at least the specified time to be counted. The signal must also be inactive for at least the specified time before the next pulse can be counted.
Note that the mode must be set when the PCT function block is called for the first time, it can not be changed once it is running.

Approximate minimum pulse width for the different modes:
_PCT_T_Debounce disabled.
NONE
_PCT_T_0.5 ms.
500US
_PCT_T_1 ms
1MS
_PCT_T_2 ms
2MS
_PCT_T_5 ms
5MS
_PCT_T_10 ms
10MS
_PCT_T_20 ms
20MS
_PCT_T_50 ms
50MS
_PCT_T_100 ms
100MS
_PCT_T_200 ms
200MS
_PCT_T_500 ms
500MS
_PCT_T_1000 ms
1000MS

Output:

cv : DINT
The current value of the counter.

Declaration:

```
FUNCTION_BLOCK PCT;
VAR_INPUT
   pinput : PTR;  | Address of input variable (by using the ADDR() function)
   ld     : BOOL; | Load new counter value
   mode   : SINT; | Debounce mode
   ncv    : DINT; | New counter value
END_VAR;
VAR_OUTPUT
   cv     : DINT; | Current counter value
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   input1 : BOOL; | Input that will increment the counter
END_VAR;

VAR
```

```
   hscounter : PCT; // Declare an instance of the PCT functionblock
END_VAR;

PROGRAM test;

hscounter.pinput:=ADDR(input1); // Assign address of input to use as pulse
input

BEGIN
   hscounter(); //
   DebugFmt(message:="Counter=\4",v4:=hscounter.cv));
   Sleep(delay:=1000);
END;
END_PROGRAM;
```

4.1.6 SFL: Datalogger

4.1.6.1 SFL: Datalogger

The Datalogger functions implement an advanced and flexible facility for logging data records in a circular buffer schema.
The size of each record in the Datalogger can be determined by the application and there is functionality to navigate and traverse the logged data.
X32 and NX32 devices have support for a single Datalogger that is stored in Data Flash.
NX32L devices also have support for additional Dataloggers that are stored in the file-system or in memory, see the NX32L file-based Datalogger section below for details.

Please have a look at the Datalogger Example Program.

• logInitialize	Initializes the Datalogger system.
• logClear	Clears the contents of Datalogger.
• logNext	Advances read position (forward in time).
• logPrev	Advances read position (backward in time).
• logFirst	Sets the read pointer to the oldest record.
• logLast	Sets the read pointer to the newest record.
• logNumOfRecords	Returns the number of records currently in the Datalogger.
• logMaxNumOfRecords	Returns the maximum number of records the Datalogger is configured for.
• logValuesPerRecord	Returns the number of values logged in each record.
• logGetMaxRecordSize	Returns the maximum record size the Datalogger is configured for.
• logIsInitialized	Check whether the Datalogger is initialized.
• logWrite	Writes an entry into the Datalogger.
• logWriteRaw	Writes raw data to an entry in the Datalogger.
• logReWrite	Writes (updates) an entry at the current read position.
• logReWriteTag	Writes (updates) the tagvalue for the entry at the current read position.
• logRewriteRaw	Writes (updates) a raw entry at the current read position.
• logRead	Reads data from Datalogger at the current read position.
• logReadRaw	Reads raw data from an entry in the Datalogger at the current read position.
• logDestroy	Destroys the setup and contents of the Datalogger.
• logGotoLinsec	Searches and places a read pointer at specified timestamps.
• logSeek	Moves the read pointer a number of elements forward/backward in time on a specific tag-id.
• logCreate	Create a new Datalogger in the filesystem.
• logCreateMem	Create a new Datalogger in memory.
• logOpen	Open an existing Datalogger.
• logClose	Close an open Datalogger.
• logSaveToFile	Saves the Datalogger to a file.

NX32L file-based Datalogger

On NX32L devices it is possible to create additional Dataloggers in the filesystem and in memory using the functions logCreate and logCreateMem.
The file-based Dataloggers support larger entries that contain raw data and are stored in the internal file-system.

The Dataloggers can be copied to external storage either by closing the log with logClose and then copying the file with the file-system functions or by using logSaveToFile. The stored files can then be opened read-only at a later time.

Memory usage of the Datalogger

The capacity of the Datalogger depends on what kind of device in use. For NX32 based devices the capacity is approx. 1024 KB.
For an X32 based the capacity depends on the the specific device in question and is in most cases 470 KB.

Each entry in the Datalogger has the following internal format:

Field	Size in bytes	Function
DateTime	4	Time of logentry (seconds since 1980-1-1 00:00:00)
Tag	1	Usersupplied Tagvalue (-128..127)
Value[1]	*4*	*Log value*
Value[2]	*4*	*Log value*
Value[3]	*4*	*Log value*
Value[4]	*4*	*Log value*
Value[5]	*4*	*Log value*
Value[6]	*4*	*Log value*
Value[7]	*4*	*Log value*
Value[8]	*4*	*Log value*

The number of bytes occupied by each entry depends on the number of values stored in each entry and is defined by using the function logInitialize(). The minimum number of bytes a log entry uses is 5 bytes, and the maximum is 37 bytes. When the logInitialize() function is called, one of the parameters is the number of entries that the Datalogger should be able to contain. When the logInitialize() function is called, it will calculate how much flash memory is needed to store the number of log entries based on the number of records needed, the number of log values stored in each record, and the amount of memory available in flash memory. If there is not enough memory for the requested Datalogger size, the logInitialize() function will return FALSE - otherwise it will return TRUE.

Write limitations:

The Datalogger is implemented using flash memory technology, which means that there is a limited number of writes possible before Flash memory wear-out occurs.
Using the logWrite() functionblock usually does not impose a problem as the firmware automatically performs write balancing on the flash memory.
The following minimum number of logWrite() operations can be performed without any risk of flash wear-out:

NX32 architecture devices:

Values per. entry	Total number of log entries	Number of logWrite() operations

1	111360	6620689
2	76800	9600000
3	57600	12800000
4	46080	16000000
5	38400	19200000
6	34560	21333333
7	30720	24000000
8	26880	27428571

X32 architecture devices:

Values per. entry	Total number of log entries	Number of logWrite() operations
1	51968	3089655
2	35840	4480000
3	26880	5973333
4	21504	7466666
5	17920	8960000
6	16128	9955555
7	14336	11200000
8	12544	12800000

Caution must be taken when using logRewrite() and logRewriteTag() because for each rewrite operation, the number of logWrite operations will be divided by the number of rewrites done on each log entry.

Number of rewrite operations to each log-entry	Number of logWrite operations divider
1	2
2	3
3	4
4	5
5	6
n	n+1

As an example, this means that with 8 log values per log entry, there will be approx. 27.4 million (NX32) logWrite operations with no rewrite operations. With 1 rewrite operation, this number will be 13.7 million, and with 2 rewrite operations, it will be 6.85 million etc.

4.1.6.2 Datalogger Example Program

Please see the Examples - Datalogger for an example program that uses the Datalogger system. An example for the new file based Datalogger can be found on the documentation for logWriteRaw.

4.1.6.3 logInitialize (Function)

```
Architecture:       X32 / NX32 / NX32L
Device support:     ALL
Firmware version:   1.00 / 1.00.00
```

---

logInitialize initializes the memory used for the Datalogger. Whenever this function is called, any data already present in the Datalogger memory will be deleted. In order to detect if the Datalogger memory is already configured, use the functions logIsInitialized(), logMaxNumOfRecords(), and logValuesPerRecord(). The supplied key-value is checked when logIsInitialized() is called, which

will only return true if the key-value is the same as used in the logIsInitialized() function call.
The number of records requested can be set to 0, which in turn will set the number of records to the maximum number available.

To create file- or memory-based Dataloggers, please use logCreate or logCreateMem.

Input:

numlogvalues : SINT (0..8)
Number of values that are logged in each record.

numlogrecords : DINT (0..65535)
Number of records that should be contained in the Datalogger (before wrap-around).
If 0 is specified on the number of records will be the maximum number available (depends on the "numlogvalues").

key : DINT
User-defined key-value.

Returns: BOOL

True if successful, false if there is no room for the specified number of records/number of values.

Declaration:

```
FUNCTION logInitialize : BOOL;
VAR_INPUT
   numlogvalues  : SINT;
   numlogrecords : DINT;
   key           : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Initialize log system to contain 2 values per record, and room for 100
records, keyvalue is 4711
   logInitialize(key:=4711, numlogvalues:=2, numlogrecords:=100);
   ...
END;

END_PROGRAM;
```

4.1.6.4 logClear (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

---

logClear will remove any records in the Datalogger memory. The selected number of values and number of records that was used during the call to logInitialize() will still be preserved. The Datalogger will simply be empty.

Input:

handle : SYSHANDLE
A handle to the Datalogger to clear. If not provided, it will clear the flash-based log. Only supported on NX32L with firmware 1.50.00 or newer.

Returns: INT

0	- Successful.
4	- Log is not initialized.
6	- Failed to read log.
7	- The log could not be found.
8	- The log is not writable.
10	- Failed to write to log.

Declaration:

```
FUNCTION logClear : INT;
VAR_INPUT
   handle     : SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Clear the logsystem, after this call, no records will be present in the
Datalogger
   logClear();
   ...
END;

END_PROGRAM;
```

4.1.6.5 logNext (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

logNext will advance the current read-pointer one step ahead (forward in time).

Note for multithreading application:
There is only one global read pointer available for each Datalogger which means that several threads navigating the Datalogger should implement critical sections to avoid problems.

Input:

handle : SYSHANDLE
A handle to the Datalogger to move the read-pointer in. If not provided, it will work on the flash-based log. Only supported on NX32L with firmware 1.50.00 or newer.

Returns: BOOL

True if successful, false if there is no record to move to (if there is no next record or Datalogger is empty).

Declaration:

```
FUNCTION logNext : BOOL;
VAR_INPUT
   handle     : SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   success : BOOL;
END_VAR;

BEGIN
   ...
   // Advance readpointer to next record (forward in time)
   success:=logNext();
   ...
END;

END_PROGRAM;
```

4.1.6.6 logPrev (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

logPrev will advance the current read pointer one step backwards (backward in time).

Note for multithreading application:
There is only one global read pointer available for each Datalogger which means that several threads navigating the Datalogger should implement critical sections to avoid problems.

Input:

handle : SYSHANDLE
A handle to the Datalogger to move the read-pointer in. If not provided, it will work on the flash-based log. Only supported on NX32L with firmware 1.50.00 or newer.

Returns: BOOL

True if successful, false if there is no record to move to (if there is no previous record or Datalogger is empty).

Declaration:

```
FUNCTION logPrev : BOOL;
VAR_INPUT
   handle     : SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   success : BOOL;
```

```
END_VAR;

BEGIN
   ...
   // Advance readpointer to previous record (backward in time)
   success:=logPrev();
   ...
END;

END_PROGRAM;
```

4.1.6.7 logFirst (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

---

logFirst will position the current read pointer at the first (oldest) record in the Datalogger.

Note for multithreading application:
There is only 1 global read-pointer available for each Datalogger which means that several threads navigating the datalogger should implement critical sections to avoid problems.

Input:

handle : SYSHANDLE
A handle to the Datalogger to move the read-pointer in. If not provided, it will work on the flash-based log. Only supported on NX32L with firmware 1.50.00 or newer.

Returns: BOOL

True if successful, false if there is no record to move to (i.e. Datalogger is empty).

Declaration:

```
FUNCTION logFirst : BOOL;
VAR_INPUT
   handle     : SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   success : BOOL;
END_VAR;

BEGIN
   ...
   // Position readpointer to the first (oldest) record in the Datalogger
   success:=logFirst();
   ...
END;

END_PROGRAM;
```

4.1.6.8 logLast (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

---

logLast will position the current read pointer at the last (newest) record in the Datalogger. When the read pointer is positioned on the last (newest) record, it will, in case of additional logWrite operations, automatically be updated to point to the last record.

Note for multithreading application:
There is only one global read pointer available for each Datalogger which means that several threads navigating the Datalogger should implement critical sections to avoid problems.

Input:

handle : SYSHANDLE
A handle to the Datalogger to move the read-pointer in. If not provided, it will work on the flash-based log. Only supported on NX32L with firmware 1.50.00 or newer.

Returns: BOOL

True if successful, false if no record to move to (i.e. Datalogger is empty).

Declaration:

```
FUNCTION logLast : BOOL;
VAR_INPUT
   handle     : SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   success : BOOL;
END_VAR

BEGIN
   ...
   // Position readpointer to the last (newest) record in the Datalogger
   success:=logLast();
   ...
END;

END_PROGRAM;
```

4.1.6.9 logNumOfRecords (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

---

logNumOfRecords will report the number of records currently in the Datalogger.

Input:

handle : SYSHANDLE
A handle to the Datalogger to get the number of records from. If not provided, it will work on the flash-based log. Only supported on NX32L with firmware 1.50.00 or newer.

Returns: DINT
Number of records in the Datalogger.

Declaration:
```
FUNCTION logNumOfRecords : DINT;
VAR_INPUT
   handle     : SYSHANDLE;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   numrec : DINT;
END_VAR;

BEGIN
   ...
   // Get number of records in datalogger
   numrec:=logNumOfRecords();
   ...
END;

END_PROGRAM;
```

4.1.6.10 logMaxNumOfRecords (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

---

logMaxNumOfRecords will report the number of records that was set as maximum in the logInitialize(), logCreate() or logCreateMem() function (or the maximum number available if the number requested was 0).

Input:
handle : SYSHANDLE
A handle to the Datalogger to get the maximum number of records from. If not provided, it will work on the flash-based log. Only supported on NX32L with firmware 1.50.00 or newer.

Returns: DINT
Maximum number of records in the Datalogger.

Declaration:
```
FUNCTION logMaxNumOfRecords : DINT;
VAR_INPUT
   handle     : SYSHANDLE;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc
```

```
PROGRAM test;
VAR
   maxnumrec : DINT;
END_VAR;

BEGIN
   ...
   // Get maximum number of records in datalogger
   maxnumrec:=logMaxNumOfRecords();
   ...
END;

END_PROGRAM;
```

4.1.6.11 logValuesPerRecord (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

---

logValuesPerRecord will report the number of values logged in each record. This was set with the logInitialize() function.

Input:
None

Returns: **DINT**
Number of values logged in each record.

Declaration:

```
FUNCTION logValuesPerRecord : SINT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   numvalues : SINT;
END_VAR

BEGIN
   ...
   // Get number of values per record
   numvalues:=logValuesPerRecord();
   ...
END;

END_PROGRAM;
```

4.1.6.12 logGetMaxRecordSize (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 1.50.00

---

logGetMaxRecordSize will report the record size that was set as maximum in the logCreate() and logCreateMem() functions or the maximum size of the data in a record created with logInitialize().

Input:

handle : SYSHANDLE
A handle to the Datalogger to get the maximum record size from. If not provided, it will work on the flash-based log.

Returns: DINT
Maximum size of a record in the Datalogger.

Declaration:
```
FUNCTION logGetMaxRecordSize : DINT;
VAR_INPUT
   handle     : SYSHANDLE;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   maxsizerec : DINT;
END_VAR;

BEGIN
   ...
   // Get maximum size of a record in datalogger
   maxsizerec:=logGetMaxRecordSize();
   ...
END;

END_PROGRAM;
```

4.1.6.13 logIsInitialized (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

logIsInitialized will check if the Datalogger system is correct initialized (The logInitialize() function has been called at some point). If the logInitialize() function has been called with a keyvalue that is different from the key value used in this function, logIsInitialized() will return false.

Input:
key : DINT
User defined keyvalue.

handle : SYSHANDLE
A handle to the Datalogger to check if is initialized. If not provided, it will check the flash-based log. Only supported on NX32L with firmware 1.50.00 or newer.

Returns: BOOL
True if Datalogger system has been initialized (and it was initialized with the same key value), false if not.

Declaration:
```
FUNCTION logIsInitialized : BOOL;
VAR_INPUT
   key        : DINT;
   handle     : SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Check if Datalogger has been initialized
   IF logIsInitialized(key:=4711) THEN
      ...
   END_IF;
   ...
END;

END_PROGRAM;
```

4.1.6.14 logWrite (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

logWrite is used for making entries in the Datalogger. Records logged with the logWrite can automatically be timestamped with the current time.
To write to file- or memory-based Dataloggers, please use logWriteRaw.

Input:

tag : SINT
Tag value for the record.

value : ARRAY [1..8] OF DINT
Value for each of the up to 8 separate values logged in each record.

linsec : DINT
Optional timestamp in seconds since 1980-1-1 00:00:00 to use (if -1, the current time is used). Please note that the function LogGotoLinsec() function will not work correctly if log entries are not written with increasing timestamps.

Output:

None

Declaration:

```
FUNCTION_BLOCK logWrite;
VAR_INPUT
   tag    : SINT;                | Tagvalue for the record
   value  : ARRAY[1..8] OF DINT; | Up to 8 values for the record
   linsec : DINT := -1;          | Optional timestamp for log-entry. -1
denotes current time.
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   LogWriter : logWrite; // Declare an instance of the LogWrite functionblock
END_VAR;

PROGRAM test;
```

```
BEGIN
   ...
   LogWriter(tag:=1, value[1]:=10, value[2]:=20); // Make an entry in the
Datalogger
   ...
END;
END_PROGRAM;
```

4.1.6.15 logWriteRaw (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 1.50.00

---

logWriteRaw writes raw data to an entry in the datalogger.
To write to the flash-based Datalogger, please use logWrite.

Input:

handle : SYSHANDLE
A handle to the Datalogger to write to.

tag : SINT
Tag value for the entry.

data : PTR
The data to write to the Datalogger.

size : DINT
The size of the data.

linsec : DINT *default -1*
Optional timestamp in seconds since 1980-1-1 00:00:00 to use (if -1, the current time is used). Please note that the function LogGotoLinsec() function will not work correctly if log entries are not written with increasing timestamps.

Returns: INT

0	- Successful.
4	- Log is not initialized.
5	- The data is too large for the entry.
7	- The log could not be found. This function is only supported on file- and memory-based Dataloggers.
8	- The log is not writable.
10	- Failed to write to the log.

Declaration:

```
FUNCTION logWriteRaw : INT;
VAR_INPUT
   handle     : SYSHANDLE;
   tag        : SINT;
   data       : PTR;
   size       : INT;
   linsec     : DINT := -1;
END_VAR;
```

Example:

```
//-------------------------------------------------------------------------
-
// Datalogger example.
// Writes an entry to the datalogger on each boot.
//-------------------------------------------------------------------------
-
INCLUDE rtcu.inc
INCLUDE math.inc

STRUCT_BLOCK data;
   temperature : INT;
   voltage     : INT;
   supply_type : SINT;
END_STRUCT_BLOCK;

//  These are the global variables of the program
VAR
   wrData : data;
   rdData : data;
END_VAR;

FUNCTION writeEntry;
VAR_INPUT
   handle : SYSHANDLE;
END_VAR;
VAR
   rc     : INT;
END_VAR;
   wrData.supply_type := SINT(boardSupplyType());
   wrData.voltage     := boardSupplyVoltage();
   wrData.temperature := boardTemperature();

   rc := logWriteRaw(handle := handle, tag := 0, data := ADDR(wrData), size :=
SIZEOF(wrData));
   DebugFmt(message := "logWriteRaw: \1", v1 := rc);
END_FUNCTION;

FUNCTION printLog;
VAR_INPUT
   handle   : SYSHANDLE;
END_VAR;
VAR
   rc       : INT;
   maxSize  : INT;
   status   : BOOL;
   i        : INT;
   t        : SINT;
   l        : DINT;
END_VAR;
   maxSize := INT(logGetMaxRecordSize(handle := handle));
   DebugFmt(message := "Log: \1 * \4", v1 := maxSize,
      v4 := logMaxNumOfRecords(handle := handle));
   IF maxSize > SIZEOF(rdData) THEN
      maxSize := SIZEOF(rdData);
   END_IF;

   status  := logFirst(handle := handle);
   i := 0;
   WHILE status DO
      rc := logReadRaw(handle := handle, data := ADDR(rdData), tag := t, size
:= maxSize, linsec := l);
      IF rc <> 0 THEN
```

```
            DebugFmt(message := "logReadRaw: \1", v2 := rc);
            EXIT;
         END_IF;
         DebugFmt(message := "\1] time \4, temp: " + floatToStr(v :=
FLOAT(rddata.temperature)/100.0) +
            ", voltage: " + floatToStr(v := FLOAT(rddata.voltage)/10.0) + ",
supply: \2",
            v1 := i, v2 := rddata.supply_type, v4 := l);
         i := i+1;
         status := logNext(handle := handle);
      END_WHILE;
END_FUNCTION;

PROGRAM test;
// These are the local variables of the program block
VAR
   handle : SYSHANDLE;
   rc     : INT;
   name   : STRING := "B:\SYSTEM\DATALOGS\log.bin";
END_VAR;
// The next code will only be executed once after the program starts
   fsMediaOpen(media := 1);
   IF fsFileExists(name := name) THEN
      rc := logOpen(handle := handle, filename := name, key := 1234);
      DebugFmt(message := "logOpen: \1", v1 := rc);
   END_IF;
   IF NOT BOOL(handle) THEN
      // log is not open, it probably failed because of the wrong key, so
create it.
      rc := logCreate(handle := handle, filename := name, key := 1234,
         rec_size := SIZEOF(data), rec_count := 1000);
      DebugFmt(message := "logCreate: \1", v1 := rc);
   END_IF;

   writeEntry(handle := handle);

   printLog(handle := handle);
BEGIN
// Code from this point until END will be executed repeatedly


END;
END_PROGRAM;
```

4.1.6.16 logRewrite (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

---

logRewrite is used for changing an entry in the Datalogger. logRewrite will change both the tag and values of the record currently pointed to by the read pointer.

Caution must be taken when using logRewrite() and logRewriteTag() as it will reduce the number of possible write operations to the Datalogger. Please read the Datalogger introductory section for more information.
To change entries in file- or memory-based Dataloggers, please use logRewriteRaw.

Input:

tag : SINT
The new tag value for the record.

value : ARRAY [1..8] OF DINT
The new values for each of the up to 8 separate values logged in each record.

Output:
None

Declaration:
```
FUNCTION_BLOCK logRewrite;
VAR_INPUT
   tag   : SINT;                | Tagvalue for the record
   value : ARRAY[1..8] OF DINT; | Upto 8 values for the record
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

VAR
   LogRewriter : logRewrite; // Declare an instance of the LogRewrite
functionblock
END_VAR;

PROGRAM test;

BEGIN
   ...
   // Position readpointer to the first (oldest) record in the Datalogger
   logFirst();
   LogRewriter(tag:=1, value[1]:=10, value[2]:=20); // Change the entry
currently pointed to by the readpointer
   ...
END;
END_PROGRAM;
```

4.1.6.17 logRewriteTag (Function)

Architecture: **X32 / NX32 / NX32L**
Device support: **ALL**
Firmware version: **1.00 / 1.00.00**

---

logRewriteTag is used for changing just the tag value of the record currently pointed to by the read pointer.

Caution must be taken when using logRewrite() and logRewriteTag() as it will reduce the number of possible write operations to the Datalogger. Please read the Datalogger introductory section for more information.
This function can be used on all types of Dataloggers.

Input:
tag : SINT
The new tag value for the record.

handle : SYSHANDLE
A handle to the Datalogger to change the tag in. If not provided, it will work on the flash-based log. Only supported on NX32L with firmware 1.50.00 or newer.

Returns: INT

0 - Successful.
1 - There is not enough memory to rewrite.
2 - Log empty (no records available).
4 - Log is not initialized.
6 - Invalid read position. (log has been cleared)
7 - The log could not be found.
8 - The log is not writable.
10 - Failed to write to the log.

Declaration:

```
FUNCTION logRewriteTag : INT;
VAR_INPUT
   tag    : SINT;
   handle : SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Position readpointer to the first (oldest) record in the Datalogger
   logFirst();
   logRewriteTag(tag:=2); // Change the tagvalue for the entry currently
pointed to by the readpointer
   ...
END;
END_PROGRAM;
```

4.1.6.18 logRewriteRaw (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 1.50.00

logRewriteRaw is used to change an entry in the Datalogger. It will change both the tag and the data of the entry currently pointed to by the read pointer.
To change entries in the flash-based Datalogger, please use LogRewrite.

Input:

handle : SYSHANDLE
A handle to the Datalogger to change the entry in.

tag : SINT
The new tag value for the entry.

data : PTR
The new data for the entry.

size : DINT
The size of the new data.

Returns: INT

0 - Successful.
1 - There is not enough memory to rewrite.
2 - Log empty (no records available).
4 - Log is not initialized.
5 - The data is too large for the Datalogger.
6 - Invalid read position. (log has been cleared)
7 - The log could not be found. This function is only supported on file- and memory-based Dataloggers.
8 - The log is not writable.
10 - Failed to write to the log.

Declaration:

```
FUNCTION logRewriteRaw : INT;
VAR_INPUT
   handle     : SYSHANDLE;
   tag        : SINT;
   data       : PTR;
   size       : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   handle : SYSHANDLE;
   data   : DINT := 42;
END_VAR;
BEGIN
   ...
   // Position readpointer to the first (oldest) record in the Datalogger
   logFirst(handle := handle);
   logRewriteRaw(handle := handle, tag := 2, data := ADDR(data), size :=
SIZEOF(data)); // Change the tag and the value for the entry currently pointed
to by the readpointer
   ...
END;
END_PROGRAM;
```

4.1.6.19 logRead (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

logRead is used for reading entries in the Datalogger. It will return the record from the current read position.
To read from file- or memory-based Dataloggers, please use logReadRaw.

Input:

None

Output:

tag : SINT
Tag value for the record.

year : SINT
Year value from the timestamp.

month : SINT
Month value from the timestamp.

day : SINT
Date value from the timestamp.

hour : SINT
Hour value from the timestamp.

minute : SINT
Minute value from the timestamp.

second : SINT
Second value from the timestamp.

linsec : DINT
The timestamp in seconds since 1980-1-1 00:00:00.

value : ARRAY [1..8] OF DINT
Value for each of the up to 8 separate values logged in the record.

status : SINT
0 - Successful.
1 - Unspecified error.
2 - Log empty (no records available).
4 - Log is not initialized.
6 - Invalid read position. (log has been cleared)

Declaration:

```
FUNCTION_BLOCK logRead;
VAR_OUTPUT
   tag    : SINT;                | Tagvalue for the record
   year   : INT;                 | 1980..2048
   month  : SINT;                | 1..12
   day    : SINT;                | 1..31
   hour   : SINT;                | 0..23
   minute : SINT;                | 0..59
   second : SINT;                | 0..59
   linsec : DINT;                | Time in seconds since 1980-1-1 00:00:00
   value  : ARRAY[1..8] OF DINT; | Upto 8 values for the record
   status : SINT;                | status
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   LogReader : logRead; // Declare an instance of the LogRead functionblock
END_VAR;

PROGRAM test;

BEGIN
```

```
    ...
    logFirst();
    LogReader(); // Read the logentry
    // LogReader.tag contains the tagvalue
    // LogReader.value[1] contains the first value
    // LogReader.year contains the year from the timestamp
    // etc etc
    ...
END;
END_PROGRAM;
```

4.1.6.20 logReadRaw (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 1.50.00

---

logReadRaw reads raw data from an entry in the datalogger.
To read from the flash-based Datalogger, please use logRead.

Input:

handle : SYSHANDLE
A handle to the Datalogger to read from.

data : PTR
A buffer to read the data into.

size : DINT
The number of bytes of data to read.

Output:

tag : SINT
Tag value for the entry.

linsec : DINT
The timestamp in seconds since 1980-1-1 00:00:00.

Returns: INT

0 - Successful.
2 - Log empty (no records available).
4 - Log is not initialized.
6 - Failed to read entry.
7 - The log could not be found. This function is only supported on file- and memory-based Dataloggers.
8 - Invalid size.

Declaration:

```
FUNCTION logReadRaw : INT;
VAR_INPUT
    handle      : SYSHANDLE;
    data        : PTR;
    size        : INT;
    tag         : ACCESS SINT;
    linsec      : ACCESS DINT;
END_VAR;
```

Example:

```
//-------------------------------------------------------------------------
-
// Datalogger example.
// Writes an entry to the datalogger on each boot.
//-------------------------------------------------------------------------
-
INCLUDE rtcu.inc
INCLUDE math.inc

STRUCT_BLOCK data;
   temperature : INT;
   voltage     : INT;
   supply_type : SINT;
END_STRUCT_BLOCK;

//  These are the global variables of the program
VAR
   wrData : data;
   rdData : data;
END_VAR;

FUNCTION writeEntry;
VAR_INPUT
   handle : SYSHANDLE;
END_VAR;
VAR
   rc     : INT;
END_VAR;
   wrData.supply_type := SINT(boardSupplyType());
   wrData.voltage     := boardSupplyVoltage();
   wrData.temperature := boardTemperature();

   rc := logWriteRaw(handle := handle, tag := 0, data := ADDR(wrData), size :=
SIZEOF(wrData));
   DebugFmt(message := "logWriteRaw: \1", v1 := rc);
END_FUNCTION;

FUNCTION printLog;
VAR_INPUT
   handle   : SYSHANDLE;
END_VAR;
VAR
   rc       : INT;
   maxSize  : INT;
   status   : BOOL;
   i        : INT;
   t        : SINT;
   l        : DINT;
END_VAR;
   maxSize := INT(logGetMaxRecordSize(handle := handle));
   DebugFmt(message := "Log: \1 * \4", v1 := maxSize,
      v4 := logMaxNumOfRecords(handle := handle));
   IF maxSize > SIZEOF(rdData) THEN
      maxSize := SIZEOF(rdData);
   END_IF;

   status  := logFirst(handle := handle);
   i := 0;
   WHILE status DO
      rc := logReadRaw(handle := handle, data := ADDR(rdData), tag := t, size
```

```
:= maxSize, linsec := l);
      IF rc <> 0 THEN
         DebugFmt(message := "logReadRaw: \1", v2 := rc);
         EXIT;
      END_IF;
      DebugFmt(message := "\1] time \4, temp: " + floatToStr(v :=
FLOAT(rddata.temperature)/100.0) +
         ", voltage: " + floatToStr(v := FLOAT(rddata.voltage)/10.0) + ",
supply: \2",
         v1 := i, v2 := rddata.supply_type, v4 := l);
      i := i+1;
      status := logNext(handle := handle);
   END_WHILE;
END_FUNCTION;

PROGRAM test;
// These are the local variables of the program block
VAR
   handle : SYSHANDLE;
   rc     : INT;
   name   : STRING := "B:\SYSTEM\DATALOGS\log.bin";
END_VAR;
// The next code will only be executed once after the program starts
   fsMediaOpen(media := 1);
   IF fsFileExists(name := name) THEN
      rc := logOpen(handle := handle, filename := name, key := 1234);
      DebugFmt(message := "logOpen: \1", v1 := rc);
   END_IF;
   IF NOT BOOL(handle) THEN
      // log is not open, it probably failed because of the wrong key, so
create it.
      rc := logCreate(handle := handle, filename := name, key := 1234,
         rec_size := SIZEOF(data), rec_count := 1000);
      DebugFmt(message := "logCreate: \1", v1 := rc);
   END_IF;

   writeEntry(handle := handle);

   printLog(handle := handle);
BEGIN
// Code from this point until END will be executed repeatedly


END;
END_PROGRAM;
```

4.1.6.21 logDestroy (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

---

logDestroy will completely clear the Datalogger memory so that a call to logIsInitialized() will return false. Call logInitialize() to initialize the Datalogger again.

Input:
None

Returns:
None

Declaration:

```
FUNCTION logDestroy;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Destroy the logsystem.
   logDestroy();
   ...
END;

END_PROGRAM;
```

4.1.6.22 logGotoLinsec (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

---

logGotoLinsec will search for an entry in the Datalogger that matches the specified timestamp, and if no match is found, it will select the nearest record (if any). It is possible to specify the search direction as either forward or backward.

Note for multithreading application:
There is only 1 global read pointer available for each Datalogger which means that several threads navigating the Datalogger should implement critical sections to avoid problems.

Input:

linsec : DINT
Timestamp in seconds since 1980-1-1 00:00:00 to search for.

forward : BOOL
If true, the function searches forward, if false, it searches backwards.

handle : SYSHANDLE
A handle to the Datalogger to navigate. If not provided, it will work on the flash-based log. Only supported on NX32L with firmware 1.50.00 or newer.

Returns: BOOL

True if successful, false if no record to move to (i.e. Datalogger is empty).

Declaration:

```
FUNCTION logGotoLinsec : BOOL;
VAR_INPUT
   linsec  : DINT; | Timestamp to search for.
   forward : BOOL; | Direction to search, false is backward, true is forward
   handle  : SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
```

```
PROGRAM test;
VAR
   success : BOOL;
END_VAR;

BEGIN
   ...
   // Position readpointer to the last (newest) record in the Datalogger
   logLast();
   // Search for record
   success:=logGotoLinsec(linsec:=clockTimeToLinsec(year:=2003, month:=5,
day:=6, Hour:=8, minute:=46, Second:=39), forward:=false);
   ...
END;

END_PROGRAM;
```

4.1.6.23 logSeek (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

---

logSeek will move the current read position a number of records forward/backward (in time) in the Datalogger. It is possible to specify a specific tag value and the number of rows, and whether the search should be forward or backward in time. If the tag value is specified, the function will move the current read position'"n" a number of records with the specified tag value in the specified direction. If no tag value is specified, the current read position will just move "n" number of rows - regardless of the tag values of the rows. The direction of the search is given by the sign of "n". If it is positive, the search will be forward in time, if it is negative, the search will be backwards.

Please note:
The search will start at the next record in the specified direction. This means that in case the tag value of the current record is the same as specified in logSeek(), this record will not be returned. This is especially important when using logLast() / logFirst().

Note for multithreading application:
There is only 1 global read-pointer available for each Datalogger which means that several threads navigating the Datalogger should implement critical sections to avoid problems.

Input:

tag : INT
Which tag value to search for. -1 indicates that no tag value is specified.

n : INT
Number of rows to move. If positive, the search will be forward, if negative, the search will be backwards.

handle : SYSHANDLE
A handle to the Datalogger to navigate. If not provided, it will work on the flash-based log. Only supported on NX32L with firmware 1.50.00 or newer.

Returns: INT

Number of records actually moved (maintains the sign of "n").

Declaration:

```
FUNCTION logSeek : INT;
VAR_INPUT
```

```
   tag    : INT := -1; | Tagvalue to search for (-1 seeks without using a
specific tagvalue).
   n      : INT;       | Number of rows to move
   handle : SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Position readpointer to the first (oldest) record in the Datalogger
   logFirst();
   // Move 5 records forward on tagvalue 12
   logSeek(tag:=12, n:=5);
   ...
END;

END_PROGRAM;
```

4.1.6.24 logCreate (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 1.50.00

logCreate creates a new Datalogger in a file on the file system.

The number of records requested can be set to 0, which in turn will set the number of records to the maximum number available.
The maximum size of a datalogger is 64MB.

Input:

filename : STRING
The path to the file to create. The file must be placed in the B:\SYSTEM\DATALOGS\ folder.

rec_size : INT
The maximum size of each record in bytes. This does not include the data for the timestamp, tag and other fixed data.

rec_count : DINT
Number of records that should be contained in the Datalogger (before wrap-around).
If 0 is specified on the number of records will be the maximum number available (depends on the "rec_size").

key : DINT
User-defined key-value.

Output:

handle : SYSHANDLE
A handle to the created datalogger.

Returns: **INT**

0 - Successful.
1 - Failed to create the file.
5 - Requested size and count is loo large.
7 - The log could not be created, too many logs are open.

8 - The drive is not writable.
9 - Invalid drive. Trying to write to an external media or the Project drive.
10 - Invalid folder.

Declaration:

```
FUNCTION logCreate : INT;
VAR_INPUT
   filename   : STRING;
   key        : DINT;
   rec_size   : SINT;
   rec_count  : DINT;
   handle     : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   handle:SYSHANDLE;
   rc:INT;
   name: STRING := "B:\SYSTEM\DATALOGS\log.bin";
END_VAR;
BEGIN
   ...
   // Initialize log system to contain 20 bytes per record, and room for 1000
records, keyvalue is 1234
   logCreate(handle:=handle, filename:=name, key:=1234, rec_size:=20,
rec_count:=1000);
   ...
END;

END_PROGRAM;
```

4.1.6.25 logCreateMem (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 1.50.00

logCreateMem creates a new Datalogger in memory. This is useful for data that does not need to be persistent and is written often, as it can not be worn out.
To save the data persistently, logSaveToFile can be used to store the log on the filesystem.

The number of records requested can be set to 0, which in turn will set the number of records to the maximum number available.
The maximum size of a datalogger is 1 MB.

Input:

rec_size : INT
The maximum size of each record in bytes. This does not include the data for the timestamp, tag and other fixed data.

rec_count : DINT
Number of records that should be contained in the Datalogger (before wrap-around).
If 0 is specified on the number of records will be the maximum number available (depends on the "rec_size").

key : DINT
User-defined key-value.

Output:
handle : SYSHANDLE
A handle to the created datalogger.

Returns: INT
0 - Successful.
1 - Failed to create log.
5 - The requested size is too large.
7 - The log could not be created, too many logs are open.

Declaration:
```
FUNCTION logCreateMem : INT;
VAR_INPUT
   key        : DINT;
   rec_size   : SINT;
   rec_count  : DINT;
   handle     : ACCESS SYSHANDLE;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   handle:SYSHANDLE;
   rc:INT;
END_VAR;
BEGIN
   ...
   // Initialize log system to contain 20 bytes per record, and room for 1000
records, keyvalue is 1234
   logCreateMem(handle:=handle, key:=1234, rec_size:=20, rec_count:=1000);
   ...
END;

END_PROGRAM;
```

4.1.6.26 logOpen (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 1.50.00

---

logOpen opens an existing Datalogger from a file on the file system.

Input:
filename : STRING
The path to the file to open. If the file is not located in the B:\SYSTEM\DATALOGS\ folder, it will be opened read-only.
If no filename is provided, the handle to the default log file will be returned.

key : DINT
User-defined key-value. If it does not match the key stored in the datalogger, an error will be reported.

Output:
handle : SYSHANDLE
A handle to the opened datalogger.

Returns: INT
0 - Successful.
1 - Invalid filename or failed to open the file.
7 - The log could not be opened, too many logs are open.
8 - Invalid datalogger, it has the wrong key.

Declaration:
```
FUNCTION logOpen : INT;
VAR_INPUT
   filename    : STRING;
   key         : DINT;
   handle      : ACCESS SYSHANDLE;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   handle:SYSHANDLE;
   rc:INT;
   name: STRING := "B:\SYSTEM\DATALOGS\log.bin";
END_VAR;
BEGIN
   ...
   // Open datalogger with keyvalue 1234
   logOpen(handle:=handle, filename:=name, key:=1234);
   ...
END;

END_PROGRAM;
```

4.1.6.27 logClose (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 1.50.00

---

logClose closes an open Datalogger.

Input:
handle : SYSHANDLE
A handle to an open datalogger.

Output:
handle : SYSHANDLE
The handle is invalid once logClose returns.

Returns: INT

0 - Successful.
7 - The log could not be found.

Declaration:

```
FUNCTION logClose : INT;
VAR_INPUT
   handle     : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   handle:SYSHANDLE;
END_VAR;
BEGIN
   ...
   // Close datalogger
   logClose(handle:=handle);
   ...
END;

END_PROGRAM;
```

4.1.6.28 logSaveToFile (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 1.50.00

---

logSaveToFile saves an open Datalogger to a file on the file system.
Only file- and memory-based Dataloggers are supported.

Input:

filename : STRING
The path to the file to save the datalogger to.

handle : SYSHANDLE
A handle to the datalogger to save.

Returns: INT

0 - Successful.
1 - Failed to create file.
4 - Log is not initialized.
6 - Failed to read from the log.
7 - The log could not be found. This function is only supported on file- and memory-based Dataloggers.
8 - The drive is not writable.
9 - Invalid drive.
10 - Failed to write to the file.

Declaration:

```
FUNCTION logSaveToFile : INT;
VAR_INPUT
```

```
   filename    : STRING;
   handle      : SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   handle:SYSHANDLE;
END_VAR;
BEGIN
   ...
   // Save log to a file on the SD Card
   logSaveToFile(handle:=handle, filename:="A:\LOG.BIN");
   ...
END;

END_PROGRAM;
```

4.1.7 SFL: Debugger functions

4.1.7.1 SFL: Debugger functions

The VPL environment contains a number of debugger functions:

- DebugMsg Prints debug messages to the Debug window.
- DebugFmt Prints debug messages to the Debug window.
- DebugSetLogParam Set the parameters for the Log to file service.
- DebugGetLogParam Retrieve the parameters for the Log to file service.

4.1.7.2 DebugFmt (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

The DebugFmt is an "easier to use" version of DebugMsg. It is a combination of DebugMsg and strFormat, which will enable you to print debug messages in an easy and convenient way.

Please see DebugMsg for an explanation of the workings of the DebugFmt function.

Input:

message : STRING
Format/message string. Please refer to strFormat for an explanation of the format string.

v1 : INT
Value for {1} in the format string.

v2 : INT
Value for {2} in the format string.

v3 : INT
Value for {3} in the format string.

v4 : DINT
Value for {4} in the format string.

Returns:

None

Declaration:

```
FUNCTION DebugFmt;
VAR_INPUT
   message  : STRING;
   v1,v2,v3 : INT;
   v4       : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

VAR
   a : INT;
END_VAR;

BEGIN
   ...
   a:=4711;
   DebugFmt(message:="Variable a is \1", v1:=a);
   ...
END;

END_PROGRAM;
```

4.1.7.3 DebugMsg (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

The DebugMsg function will print a text string to the debug window - either in the RTCU Emulator or from a connected RTCU device.

If DebugMsg is called from a program running on an RTCU device, and there is a connection between the RTCU IDE and an RTCU device, the DebugMsg() messages is shown in the Debug Messages window.
If the message is longer than 239 characters, it will be split across multiple lines in the RTCU IDE and the split lines will be terminated with "...".

Input:

message : STRING
Message string.

Returns:

None

Declaration:

```
FUNCTION DebugMsg;
VAR_INPUT
   message : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // show a message in the debug window
   DebugMsg(message:="Hello from VPL Program");
   ...
END;

END_PROGRAM;
```

4.1.7.4 DebugSetLogParam (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 4.70 / 1.30.00

This function will set the parameters for the log to file service.
The configuration is persistent over resets until set again by the function or from the RTCU IDE in the configuration dialog.

The parameter set by DebugSetLogParam can be read by using DebugGetLogParam.

Parameters take effect after a device reset, using the boardReset function.

Input:

media : SINT (0/1, *default 0*)
The media where the log files are stored.
Please note that this option can only be set on NX32L architecture devices.

dbg_enable : BOOL
Enable/Disable logging of debug messages.

dbg_files : INT (1..999)
The maximum number of log files with debug messages which is saved.

con_enable : BOOL
Enable/Disable logging of network console output.

con_files : INT (1..*999*)
The maximum number of log files with network console output which is saved.

Returns:

1 - Success.
0 - Interface not available or not supported.
-1 - Illegal parameter.

Declaration:

```
FUNCTION DebugSetLogParam : INT;
VAR_INPUT
   media      : SINT := 0;
   dbg_enable : BOOL;
   dbg_files  : INT;
   con_enable : BOOL;
   con_files  : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   gprscfg : gprsGetMonitorParm;
END_VAR;

// Check settings
```

```
gprscfg();
IF NOT gprscfg.Enabled OR gprscfg.MaxAttempt <> 3 OR gprscfg.AliveFreq <> 180
THEN
   // Set mobile network monitor parameters
   gprsSetMonitorParm(Enabled := TRUE, AliveFreq := 180);
   boardReset();
END_IF;

// Turn on power to the GSM module
gsmPower(power := ON);
gprsOpen();

BEGIN
   ...
END;
END_PROGRAM;
```

4.1.7.5 DebugGetLogParam (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 4.70 / 1.30.00

This function will read out the parameters for the log to file service previously set from the RTCU IDE or by the DebugSetLogParam function.

Input:

None.

Output:

ready : BOOL
True if valid data is available.

media : SINT
The media where the log files are stored.

dbg_enable : BOOL
Determines whether debug message logging is enabled.

dbg_files : INT
The maximum number of log files with debug messages which is saved.

con_enable : BOOL
Determines whether network console output logging is enabled.

con_files : INT
The maximum number of log files with network console output which is saved.

Declaration:

```
FUNCTION_BLOCK DebugGetLogParam;
VAR_OUTPUT
   ready      : BOOL;
   media      : SINT;
   dbg_enable : BOOL;
   dbg_files  : INT;
   con_enable : BOOL;
```

```
   con_files  : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   param : DebugGetLogParam;
END_VAR;

// Check settings
param();
IF param.ready AND (NOT param.dbg_enable OR param.dbg_files <> 20) THEN
   // Set parameters
   DebugSetLogParam(param.dbg_enable := TRUE, param.dbg_files := 20);
   boardReset();
END_IF;


BEGIN
   ...
END;
END_PROGRAM;
```

4.1.8 SFL: Edge triggers

4.1.8.1 SFL: Edge triggers

The VPL environment contains a number of edge trigger function blocks.

- R_TRIG — Rising edge trigger.
- F_TRIG — Falling edge trigger.
- RF_TRIG — Rising and falling edge trigger.

4.1.8.2 F_TRIG (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

F_TRIG is a falling edge detector. It will activate the "q" output when a falling edge is detected on the "trig" input.

For a rising edge trigger, please see the R_TRIG function block.

Input:

trig : BOOL (true/false)
On the falling edge of this input, the output "q" will go high for one scan.

Output:

q : BOOL (true/false)
Output from the falling edge trigger detector.

Declaration:

```
FUNCTION_BLOCK F_TRIG;
VAR_INPUT
   trig : BOOL F_EDGE; | Falling edge on this signal activates 'q'.
END_VAR;
VAR_OUTPUT
   q    : BOOL;        | Output from detector
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   input1 : BOOL; | Input for the falling edge detector
END_VAR;

VAR_OUTPUT
   signal : BOOL; | Output that will follow the q output from the falling edge
detector.
END_VAR;

VAR
   trigger : F_TRIG; // Declare an instance of the F_TRIG functionblock
END_VAR;

PROGRAM test;

BEGIN
   ...
```

```
   trigger(trig:= input1); // Input the input1 signal to the falling edge
detector
   signal := trigger.q; // output will follow the q output from the detector
   ...
END;
END_PROGRAM;
```

4.1.8.3 R_TRIG (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

R_TRIG is a rising edge detector. It will activate the "q" output when a rising edge is detected on the "trig" input.

For a falling edge trigger, please see the F_TRIG function block.

Input:

trig : BOOL (true/false)
On the rising edge of this input, the output "q" will go high.

Output:

q : BOOL (true/false)
Output from the rising edge trigger detector.

Declaration:

```
FUNCTION_BLOCK R_TRIG;
VAR_INPUT
   trig : BOOL R_EDGE; | Rising edge on this signal activates 'q' for one
execution
END_VAR;
VAR_OUTPUT
   q    : BOOL;        | Output from detector
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   input1 : BOOL; | Input for the rising edge detector
END_VAR;

VAR_OUTPUT
   signal : BOOL; | Output that will follow the q output from the rising edge
detector.
END_VAR;

VAR
   trigger : R_TRIG; // Declare an instance of the R_TRIG functionblock
END_VAR;

PROGRAM test;

BEGIN
   ...
   trigger(trig:= input1); // Input the input1 signal to the rising edge
detector
   signal := trigger.q; // output will follow the q output from the detector
   ...
```

```
END;
END_PROGRAM;
```

4.1.8.4 RF_TRIG (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

RF_TRIG is a rising and falling edge detector. It will activate the "rq" output for one execution when a rising edge is detected on the "trig" input. The "fq" output will be active for one scan if a falling edge is detected on "trig" .

For a rising edge trigger, please see the R_TRIG function block.
For a falling edge trigger, please see the F_TRIG function block.

Input:

trig : BOOL (true/false)
On the falling edge of this input, the output "q" will go high for one scan.

Output:

rq : BOOL (true/false)
Output from the rising edge trigger detector.

fq : BOOL (true/false)
Output from the falling edge trigger detector.

Declaration:

```
FUNCTION_BLOCK RF_TRIG;
VAR_INPUT
   trig : BOOL; | Falling edge on this signal triggers 'rq' and falling edge
triggers 'fq' for one scan
END_VAR;
VAR_OUTPUT
   rq   : BOOL; | Will be active if trig has a rising edge
   fq   : BOOL; | Will be active if trig has a falling edge
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   input1 : BOOL; | Input for the falling edge detector
END_VAR;

VAR_OUTPUT
   r_signal : BOOL; | Output that will follow the 'rq' output from the
detector.
   f_signal : BOOL; | Output that will follow the 'fq' output from the
detector.
END_VAR;

VAR
   trigger : RF_TRIG; // Declare an instance of the RF_TRIG functionblock
END_VAR;

PROGRAM test;

BEGIN
   ...
```

```
   trigger(trig:= input1); // Input the input1 signal to the falling/rising
edge detector
   r_signal := trigger.rq; // output will follow the 'rq' output from the
detector
   f_signal := trigger.fq; // output will follow the 'fq' output from the
detector
   ...
END;

END_PROGRAM;
```

4.1.9 SFL: Floating-point Math functions

4.1.9.1 SFL: Floating-point Math functions

The math function library contains typical mathematical, trigonometric and conversion functions.

Trigonometric functions:

- cos — Cosine of an angle
- acos — Arc cosine of an angle
- sin — Sine of an angle
- asin — Arc sine of an angle
- tan — Tangent of an angle
- atan — Arc tangent of an angle
- atan2 — Arc tangent of an angle with two parameters

Exponential and logarithmic functions:

- exp — Base-e exponential function
- exp2 — Base-2 exponential function
- frexp — Decompose to exponent
- ldexp — Combine significand and exponent
- log — Natural Logarithm
- log2 — Base-2 Logarithm
- log10 — Base-10 Logarithm
- modf — Breaks a number into an integral and a fractional part.
- sqrt — Square root

Rounding functions

- ceil — Round to smallest integral value that is larger than the value
- floor — Round to largest integral value that is smaller than the value

Other functions

- fabs — Absolute value of float
- fmin — Returns the smallest of two values
- fmax — Returns the largest of two values
- IsInf — Checks if an value is infinite
- IsNaN — Checks if an value is not a number

Conversion functions

- floatToStr — Converts a float to a string
- strToFloat — Converts a string to a float
- doubleToStr — Converts a double to a string
- strToDouble — Converts a string to a double
- degToRad — Converts an angle from degrees to radians
- radToDeg — Converts an angle from radians to degrees
- degToPosition — Converts an coordinate from decimal degrees to RTCU GPS position
- positionToDeg — Converts an coordinate from RTCU GPS position to decimal degrees

Besides the situations where the functions are described to return Infinity values or NaN, ordinary operations can also cause these error values, if the numbers either becomes too large, or perform illegal operations, such as division with zero, which can be checked with the isInf and isNaN functions.

Please note that when using the Floating-Point Math functions in X32 and NX32 compilation mode then **math.inc** must be explicitly included by the application.
When using the NX32L compilation mode, **math.inc** is included by default.

DOUBLE and FLOAT support:

The math library is implemented with support for dual precision depending on whether the Double math library option in the project settings is checked.
When this option is unchecked (default) then all functions in this library will automatically change to operating with FLOAT instead of the documented DOUBLE.

4.1.9.2 cos (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 3.10 / 1.00.00

This function will calculate the cosine of an angle.

Note: This function works with DOUBLE, but can also work with FLOAT as described in the Floating-point math introductory section.

Input:
V : DOUBLE
Angle expressed in radians.

Returns: **DOUBLE**
The cosine of the angle.

Declaration:
```
FUNCTION cos : DOUBLE;
VAR_INPUT
   v : DOUBLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;
VAR
   a : DOUBLE;
END_VAR;

a := cos(v:=0.4);
DebugMsg(message:=" cos " + doubleToStr(v := a));
DebugMsg(message:="acos " + doubleToStr(v := acos(v:=a)));
a := sin(v:=0.4);
DebugMsg(message:=" sin " + doubleToStr(v := a));
DebugMsg(message:="asin " + doubleToStr(v := asin(v:=a)));
```

```
a := tan(v:=0.4);
DebugMsg(message:=" tan " + doubleToStr(v := a));
DebugMsg(message:="atan " + doubleToStr(v := atan(v:=a)));
a := atan2(a:=10.0, b:= 10.0);
DebugMsg(message:="atan2 " + doubleToStr(v := a));

// Output:
//  cos 0.9210609916817709
// acos 0.4000000059604644
//  sin 0.3894183477986018
// asin 0.4000000059604644
//  tan 0.4227932257640837
// atan 0.4000000059604645
// atan2 0.7853981633974483

END_PROGRAM;
```

4.1.9.3 acos (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 3.10 / 1.00.00

This function will calculate the arc cosine of an angle.

Note: This function works with DOUBLE, but can also work with FLOAT as described in the Floating-point math introductory section.

Input:

V : DOUBLE (-1 .. +1)
Value to compute the arc cosine of.

Returns: DOUBLE

The arc cosine of the angle in radians. If V is invalid, NaN is returned.

Declaration:

```
FUNCTION acos : DOUBLE;
VAR_INPUT
   v : DOUBLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;
VAR
   a : DOUBLE;
END_VAR;

a := cos(v:=0.4);
DebugMsg(message:=" cos " + doubleToStr(v := a));
DebugMsg(message:="acos " + doubleToStr(v := acos(v:=a)));
a := sin(v:=0.4);
DebugMsg(message:=" sin " + doubleToStr(v := a));
DebugMsg(message:="asin " + doubleToStr(v := asin(v:=a)));
a := tan(v:=0.4);
DebugMsg(message:=" tan " + doubleToStr(v := a));
DebugMsg(message:="atan " + doubleToStr(v := atan(v:=a)));
```

```
a := atan2(a:=10.0, b:= 10.0);
DebugMsg(message:="atan2 " + doubleToStr(v := a));

// Output:
//  cos 0.9210609916817709
// acos 0.4000000059604644
//  sin 0.3894183477986018
// asin 0.4000000059604644
//  tan 0.4227932257640837
// atan 0.4000000059604645
// atan2 0.7853981633974483

END_PROGRAM;
```

4.1.9.4 sin (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 3.10 / 1.00.00

This function will calculate the sine of an angle.

Note: This function works with DOUBLE, but can also work with FLOAT as described in the Floating-point math introductory section.

Input:
V : DOUBLE
Angle expressed in radians.

Returns: **DOUBLE**
The sine of the angle.

Declaration:

```
FUNCTION sin : DOUBLE;
VAR_INPUT
   v : DOUBLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;
VAR
   a : DOUBLE;
END_VAR;

a := cos(v:=0.4);
DebugMsg(message:=" cos " + doubleToStr(v := a));
DebugMsg(message:="acos " + doubleToStr(v := acos(v:=a)));
a := sin(v:=0.4);
DebugMsg(message:=" sin " + doubleToStr(v := a));
DebugMsg(message:="asin " + doubleToStr(v := asin(v:=a)));
a := tan(v:=0.4);
DebugMsg(message:=" tan " + doubleToStr(v := a));
DebugMsg(message:="atan " + doubleToStr(v := atan(v:=a)));
a := atan2(a:=10.0, b:= 10.0);
DebugMsg(message:="atan2 " + doubleToStr(v := a));
```

```
// Output:
//  cos 0.9210609916817709
// acos 0.4000000059604644
//  sin 0.3894183477986018
// asin 0.4000000059604644
//  tan 0.4227932257640837
// atan 0.4000000059604645
// atan2 0.7853981633974483

END_PROGRAM;
```

4.1.9.5 asin (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 3.10 / 1.00.00

This function will calculate the arc sine of an angle.

Note: This function works with DOUBLE, but can also work with FLOAT as described in the Floating-point math introductory section.

Input:

V : DOUBLE (-1 .. +1)
Value to compute the arc sine of.

Returns: DOUBLE

The arc sine of the angle in radians. If V is invalid, NaN is returned.

Declaration:

```
FUNCTION asin : DOUBLE;
VAR_INPUT
   v : DOUBLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;
VAR
   a : DOUBLE;
END_VAR;

a := cos(v:=0.4);
DebugMsg(message:=" cos " + doubleToStr(v := a));
DebugMsg(message:="acos " + doubleToStr(v := acos(v:=a)));
a := sin(v:=0.4);
DebugMsg(message:=" sin " + doubleToStr(v := a));
DebugMsg(message:="asin " + doubleToStr(v := asin(v:=a)));
a := tan(v:=0.4);
DebugMsg(message:=" tan " + doubleToStr(v := a));
DebugMsg(message:="atan " + doubleToStr(v := atan(v:=a)));
a := atan2(a:=10.0, b:= 10.0);
DebugMsg(message:="atan2 " + doubleToStr(v := a));

// Output:
//  cos 0.9210609916817709
// acos 0.4000000059604644
```

```
//  sin 0.3894183477986018
// asin 0.4000000059604644
//  tan 0.4227932257640837
// atan 0.4000000059604645
// atan2 0.7853981633974483

END_PROGRAM;
```

4.1.9.6 tan (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 3.10 / 1.00.00

This function will calculate the tangent of an angle.

Note: This function works with DOUBLE, but can also work with FLOAT as described in the Floating-point math introductory section.

Input:
V : DOUBLE
Angle expressed in radians.

Returns: **DOUBLE**
The tangent of the angle.

Declaration:

```
FUNCTION tan : DOUBLE;
VAR_INPUT
   v : DOUBLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;
VAR
   a : DOUBLE;
END_VAR;

a := cos(v:=0.4);
DebugMsg(message:=" cos " + doubleToStr(v := a));
DebugMsg(message:="acos " + doubleToStr(v := acos(v:=a)));
a := sin(v:=0.4);
DebugMsg(message:=" sin " + doubleToStr(v := a));
DebugMsg(message:="asin " + doubleToStr(v := asin(v:=a)));
a := tan(v:=0.4);
DebugMsg(message:=" tan " + doubleToStr(v := a));
DebugMsg(message:="atan " + doubleToStr(v := atan(v:=a)));
a := atan2(a:=10.0, b:= 10.0);
DebugMsg(message:="atan2 " + doubleToStr(v := a));

// Output:
//  cos 0.9210609916817709
// acos 0.4000000059604644
//  sin 0.3894183477986018
// asin 0.4000000059604644
//  tan 0.4227932257640837
```

```
// atan 0.4000000059604645
// atan2 0.7853981633974483

END_PROGRAM;
```

4.1.9.7 atan (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 3.10 / 1.00.00

This function will calculate the principal value of the arc tangent of an angle.
This function can not determine the quadrant of the angle. Use atan2 if it is needed.

Note: This function works with DOUBLE, but can also work with FLOAT as described in the Floating-point math introductory section.

Input:

V : DOUBLE
Value to compute the arc tangent of.

Returns: DOUBLE

The arch tangent of the value in radians.

Declaration:

```
FUNCTION atan : DOUBLE;
VAR_INPUT
   v : DOUBLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;
VAR
   a : DOUBLE;
END_VAR;

a := cos(v:=0.4);
DebugMsg(message:=" cos " + doubleToStr(v := a));
DebugMsg(message:="acos " + doubleToStr(v := acos(v:=a)));
a := sin(v:=0.4);
DebugMsg(message:=" sin " + doubleToStr(v := a));
DebugMsg(message:="asin " + doubleToStr(v := asin(v:=a)));
a := tan(v:=0.4);
DebugMsg(message:=" tan " + doubleToStr(v := a));
DebugMsg(message:="atan " + doubleToStr(v := atan(v:=a)));
a := atan2(a:=10.0, b:= 10.0);
DebugMsg(message:="atan2 " + doubleToStr(v := a));

// Output:
//  cos 0.9210609916817709
// acos 0.4000000059604644
//  sin 0.3894183477986018
// asin 0.4000000059604644
//  tan 0.4227932257640837
// atan 0.4000000059604645
// atan2 0.7853981633974483
```

```
END_PROGRAM;
```

4.1.9.8 atan2 (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 3.10 / 1.00.00

This function returns the principal value of the arc tangent of argument a divided by argument b. The sign of both arguments are taken into account in order to determine the quadrant of the resulting value.

Note: This function works with DOUBLE, but can also work with FLOAT as described in the Floating-point math introductory section.

Input:

A : DOUBLE
Value representing the proportion of the y-coordinate.

B : DOUBLE
Value representing the proportion of the x-coordinate.

Returns: DOUBLE

The arch tangent of the value in radians.

Declaration:

```
FUNCTION atan2 : DOUBLE;
VAR_INPUT
   a : DOUBLE;
   b : DOUBLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;
VAR
   a : DOUBLE;
END_VAR;

a := cos(v:=0.4);
DebugMsg(message:=" cos " + doubleToStr(v := a));
DebugMsg(message:="acos " + doubleToStr(v := acos(v:=a)));
a := sin(v:=0.4);
DebugMsg(message:=" sin " + doubleToStr(v := a));
DebugMsg(message:="asin " + doubleToStr(v := asin(v:=a)));
a := tan(v:=0.4);
DebugMsg(message:=" tan " + doubleToStr(v := a));
DebugMsg(message:="atan " + doubleToStr(v := atan(v:=a)));
a := atan2(a:=10.0, b:= 10.0);
DebugMsg(message:="atan2 " + doubleToStr(v := a));

// Output:
//  cos 0.9210609916817709
// acos 0.4000000059604644
//  sin 0.3894183477986018
// asin 0.4000000059604644
```

```
//  tan 0.4227932257640837
// atan 0.4000000059604645
// atan2 0.7853981633974483

END_PROGRAM;
```

4.1.9.9 exp (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 3.10 / 1.00.00

This function will calculate the base-e exponential function of the argument, e^V.

Note: This function works with DOUBLE, but can also work with FLOAT as described in the Floating-point math introductory section.

Input:

V : DOUBLE
Value of exponent.

Returns: DOUBLE

The exponential value of V. If the value is larger than can be represented, Infinity is returned.

Declaration:

```
FUNCTION exp : DOUBLE;
VAR_INPUT
   v : DOUBLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;

DebugMsg( message := "e^5 = " + doubleToStr(v := exp(v:=5.0)));
DebugMsg( message := "2^5 = " + doubleToStr(v := exp2(v:=5.0)));


END_PROGRAM;
```

4.1.9.10 exp2 (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 3.10 / 1.00.00

This function will calculate the base-2 exponential function of the argument, 2^V.

Note: This function works with DOUBLE, but can also work with FLOAT as described in the Floating-point math introductory section.

Input:

V : DOUBLE
Value of exponent.

Returns: DOUBLE
The exponential value of V. If the value is larger than can be represented, Infinity is returned.

Declaration:
```
FUNCTION exp2 : DOUBLE;
VAR_INPUT
   v : DOUBLE;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;

DebugMsg( message := "e^5 = " + doubleToStr(v := exp(v:=5.0)));
DebugMsg( message := "2^5 = " + doubleToStr(v := exp2(v:=5.0)));

END_PROGRAM;
```

4.1.9.11 frexp (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 3.10 / 1.00.00

This function will break the value into its binary significand and an integral exponent for 2.
The values will follow the following formula:
V = significant * 2^exp

Note: This function works with DOUBLE, but can also work with FLOAT as described in the Floating-point math introductory section.

Input:
V : DOUBLE
Value to decompose.

Output:
exp : DOUBLE
Exponent

Returns: DOUBLE
The binary significant of v.

Declaration:
```
FUNCTION frexp : DOUBLE;
VAR_INPUT
   v   : DOUBLE;
   exp : ACCESS DOUBLE;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc
INCLUDE math.inc
```

```
PROGRAM test;
VAR
   a : DOUBLE;
   b : DOUBLE;
END_VAR;

a := frexp(v:= 1024.0, exp := b);
DebugMsg( message := " 1024 = " + doubleToStr(v := a) + " * 2 ^ " +
doubleToStr(v := b));
DebugMsg( message := " 0.5 * 2 ^ 11 = " + doubleToStr(v := ldexp(a := a, b :=
b)));


END_PROGRAM;
```

4.1.9.12 ldexp (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 3.10 / 1.00.00

Returns the result of multiplying argument 1 (the significand) by 2 raised to the power of argument 2 (the exponent).
ldexp := a * 2^b

Note: This function works with DOUBLE, but can also work with FLOAT as described in the Floating-point math introductory section.

Input:

A : DOUBLE
Significand.

B : DOUBLE
Exponent

Returns: DOUBLE

The calculated value.

Declaration:

```
FUNCTION ldexp : DOUBLE;
VAR_INPUT
   a : DOUBLE;
   b : DOUBLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;
VAR
   a : DOUBLE;
   b : DOUBLE;
END_VAR;

a := frexp(v:= 1024.0, exp := b);
DebugMsg( message := " 1024 = " + doubleToStr(v := a) + " * 2 ^ " +
doubleToStr(v := b));
DebugMsg( message := " 0.5 * 2 ^ 11 = " + doubleToStr(v := ldexp(a := a, b :=
```

```
b)));

END_PROGRAM;
```

4.1.9.13 log (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 3.10 / 1.00.00

This function will calculate the natural logarithm of the argument.

Note: This function works with DOUBLE, but can also work with FLOAT as described in the Floating-point math introductory section.

Input:

V : DOUBLE
Value.

Returns: DOUBLE

The natural logarithm of V. If the value is 0, -Infinity is returned. If the value is less than 0, NaN is returned.

Declaration:

```
FUNCTION log : DOUBLE;
VAR_INPUT
   v : DOUBLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;

DebugMsg( message := "  log(100) = " + doubleToStr(v := log( v:= 100.0)));
DebugMsg( message := " log2(100) = " + doubleToStr(v := log2( v:= 100.0)));
DebugMsg( message := "log10(100) = " + doubleToStr(v := log10( v:= 100.0)));

END_PROGRAM;
```

4.1.9.14 log2 (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 3.10 / 1.00.00

This function will calculate the base-2 logarithm of the argument.

Note: This function works with DOUBLE, but can also work with FLOAT as described in the Floating-point math introductory section.

Input:

V : DOUBLE
Value.

Returns: DOUBLE

The base-2 logarithm of V. If the value is 0, -Infinity is returned. If the value is less than 0, NaN is returned.

Declaration:

```
FUNCTION log2 : DOUBLE;
VAR_INPUT
   v : DOUBLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;

DebugMsg( message := "  log(100) = " + doubleToStr(v := log( v:= 100.0)));
DebugMsg( message := " log2(100) = " + doubleToStr(v := log2( v:= 100.0)));
DebugMsg( message := "log10(100) = " + doubleToStr(v := log10( v:= 100.0)));

END_PROGRAM;
```

4.1.9.15 log10 (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 3.10 / 1.00.00

This function will calculate the base-10 logarithm of the argument.

Note: This function works with DOUBLE, but can also work with FLOAT as described in the Floating-point math introductory section.

Input:

V : DOUBLE
Value.

Returns: DOUBLE

The base-10 logarithm of V. If the value is 0, -Infinity is returned. If the value is less than 0, NaN is returned.

Declaration:

```
FUNCTION log10 : DOUBLE;
VAR_INPUT
   v : DOUBLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;

DebugMsg( message := "  log(100) = " + doubleToStr(v := log( v:= 100.0)));
DebugMsg( message := " log2(100) = " + doubleToStr(v := log2( v:= 100.0)));
DebugMsg( message := "log10(100) = " + doubleToStr(v := log10( v:= 100.0)));

END_PROGRAM;
```

4.1.9.16 modf (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 3.10 / 1.00.00

This function will break the argument into an integral and a fractional part.

Note: This function works with DOUBLE, but can also work with FLOAT as described in the Floating-point math introductory section.

Input:

V : DOUBLE
Value to break into parts.

Output:

intpart : DOUBLE
The integral part of the value.

Returns: DOUBLE

The fraction part of the value.

Declaration:

```
FUNCTION modf : DOUBLE;
VAR_INPUT
   v       : DOUBLE;
   intpart : ACCESS DOUBLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;
VAR
   a : DOUBLE;
   b : DOUBLE;
END_VAR;

a := modf(v := 10.0/8.0, intpart := b);
DebugFmt(message:=" 10/8 = " + doubleToStr(v := b) +" + " + doubleToStr(v :=
a));

END_PROGRAM;
```

4.1.9.17 sqrt (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 3.10 / 1.00.00

This function will calculate the square root of the argument.

Note: This function works with DOUBLE, but can also work with FLOAT as described in the Floating-point math introductory section.

Input:

V : DOUBLE
Value to calculate the square root of.

Returns: DOUBLE
The square root of the value. If the value is less than 0, NaN is returned.

Declaration:
```
FUNCTION sqrt : DOUBLE;
VAR_INPUT
   v : DOUBLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;
VAR
   a : DOUBLE;
   b : DOUBLE;
   c : DOUBLE;
END_VAR;

a := 3.0;
b := 4.0;
c := sqrt(v:= a**2.0 + b**2.0);
DebugMsg( message := doubleToStr(v := a) + "^2 + "
   + doubleToStr(v := b) + "^2 = " + doubleToStr(v := c) + "^2");

END_PROGRAM;
```

4.1.9.18 ceil (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 3.10 / 1.00.00

This function will calculate the smallest integral value that is not smaller than the value.

Note: This function works with DOUBLE, but can also work with FLOAT as described in the Floating-point math introductory section.

Input:
V : DOUBLE
Value to convert.

Returns: DOUBLE
The smallest integral value as DOUBLE.

Declaration:
```
FUNCTION ceil : DOUBLE;
VAR_INPUT
   v : DOUBLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
include math.inc

PROGRAM test;

DebugFmt(message := "ceil  99.9999 = " + doubleToStr(v:= ceil(v:=99.9999)));
DebugFmt(message := "floor 99.9999 = " + doubleToStr(v:= floor(v:=99.9999)));

END_PROGRAM;
```

4.1.9.19 floor (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 3.10 / 1.00.00

This function will calculate the largest integral value that is not larger than the value.

Note: This function works with DOUBLE, but can also work with FLOAT as described in the Floating-point math introductory section.

Input:
V : DOUBLE
Value to convert.

Returns: DOUBLE
The largest integral value as DOUBLE.

Declaration:
```
FUNCTION floor : DOUBLE;
VAR_INPUT
   v : DOUBLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;

DebugFmt(message := "ceil  99.9999 = " + doubleToStr(v:= ceil(v:=99.9999)));
DebugFmt(message := "floor 99.9999 = " + doubleToStr(v:= floor(v:=99.9999)));

END_PROGRAM;
```

4.1.9.20 fabs (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 3.10 / 1.00.00

This function will calculate the absolute value.

Note: This function works with DOUBLE, but can also work with FLOAT as described in the Floating-point math introductory section.

Input:
V : DOUBLE
Value to convert.

Returns: DOUBLE
The absolute value of V.

Declaration:
```
FUNCTION fabs : DOUBLE;
VAR_INPUT
   v : DOUBLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;

DebugFmt(message := " abs( -123) = \4", v4 := abs( v:= -123));
DebugFmt(message := "fabs(-1.23) = " + doubleToStr(v:= fabs(v:=-1.23)));

END_PROGRAM;
```

4.1.9.21 fmin (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 3.10 / 1.00.00

This function will return the smallest of the two provided values.

Note: This function works with DOUBLE, but can also work with FLOAT as described in the Floating-point math introductory section.

Input:
A : DOUBLE
Value to compare.
B : DOUBLE
Value to compare.

Returns: DOUBLE
The smallest argument.

Declaration:
```
FUNCTION fmin : DOUBLE;
VAR_INPUT
   a : DOUBLE;
   b : DOUBLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;
```

```
DebugFmt(message := "min: " + doubleToStr(v:= fmin(a := 1.0, b := 1.0001)));
DebugFmt(message := "max: " + doubleToStr(v:= fmax(a := 1.0, b := 1.0001)));

END_PROGRAM;
```

4.1.9.22 fmax (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 3.10 / 1.00.00

This function will return the largest of the two provided values.

Note: This function works with DOUBLE, but can also work with FLOAT as described in the Floating-point math introductory section.

Input:

A : DOUBLE
Value to compare.
B : DOUBLE
Value to compare.

Returns: DOUBLE

The largest argument.

Declaration:

```
FUNCTION fmax : DOUBLE;
VAR_INPUT
   a : DOUBLE;
   b : DOUBLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;

DebugFmt(message := "min: " + doubleToStr(v:= fmin(a := 1.0, b := 1.0001)));
DebugFmt(message := "max: " + doubleToStr(v:= fmax(a := 1.0, b := 1.0001)));

END_PROGRAM;
```

4.1.9.23 isInf (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 3.10 / 1.00.00

This function checks if the provided value is infinite.
This should be used to verify results of calculations where it is possible to have an illegal value.

Note: Besides the functions that return infinite in case of an error, it is also possible to get infinite using normal operators, if either a result is too large, or e.g. by dividing with 0.

Note: This function works with DOUBLE, but can also work with FLOAT as described in the Floating-point math introductory section.

Input:
V : DOUBLE
Value to check.

Returns: BOOL
True if the value if +Infinity or - Infinity, False otherwise.

Declaration:
```
FUNCTION isInf : BOOL;
VAR_INPUT
   v : DOUBLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;
VAR
   a : DOUBLE;
   b : DOUBLE;
END_VAR;

a := 2.0;
WHILE NOT isInf(v := a) DO
   b := a;
   a := a * 2.0;
END_WHILE;
DebugMsg(message := " b = " + doubleToStr(v := b));

END_PROGRAM;
```

4.1.9.24 isNaN (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 3.10 / 1.00.00

This function checks if the provided value is not a number. Infinity is considered a number. This should be used to verify results of calculations where it is possible to have an illegal value. Besides the functions that return NaN on invalid input, it is also possible to get NaN by using normal operators e.g. 0/0, or other cases where the result is undefined.

Note: This function works with DOUBLE, but can also work with FLOAT as described in the Floating-point math introductory section.

Input:
V : DOUBLE
Value to check.

Returns: BOOL
True if the value is not a number, False otherwise.

Declaration:
```
FUNCTION isNaN : BOOL;
VAR_INPUT
```

```
   v : DOUBLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;
VAR
   a : DOUBLE;
END_VAR;

a := asin(v := 1.2);
IF isNaN(v := a) THEN
   DebugMsg(message := "Invalid value");
END_IF;

END_PROGRAM;
```

4.1.9.25 floatToStr(Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 3.10 / 1.00.00

This function converts a FLOAT to a string.

Note: This function works with DOUBLE, but can also work with FLOAT as described in the Floating-point math introductory section.

Input:

v : FLOAT
Value to convert.

Returns: STRING

A string representation the value of v with 7 significant digits.

Declaration:

```
FUNCTION floatToStr : STRING;
VAR_INPUT
   v : FLOAT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;

DebugMsg(message := " sin(270) = " + floatToStr(v := sin(v := degToRad(v :=
270.0))));
DebugMsg(message := " asin(1) = " + floatToStr(v := radToDeg(v := asin(v :=
1.0))));

END_PROGRAM;
```

4.1.9.26 strToFloat (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 3.10 / 1.00.00

This function converts a string to a FLOAT.

Note: This function works with DOUBLE, but can also work with FLOAT as described in the Floating-point math introductory section.

Input:

str : STRING
String to convert.

Returns: FLOAT

The value of the string. If the string can not be parsed, 0 is returned.

Declaration:

```
FUNCTION strToFloat : FLOAT;
VAR_INPUT
   str : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;
VAR
   a : FLOAT;
   b : FLOAT;
END_VAR;

a := strToFloat( str := "1234.56789");
b := strToFloat( str := "-1.23456789e+3");
DebugMsg( message := "a + b = " + floatToStr(v:= a + b));

END_PROGRAM;
```

4.1.9.27 doubleToStr (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 3.10 / 1.00.00

This function converts a DOUBLE to a string.

Note: This function works with DOUBLE, but can also work with FLOAT as described in the Floating-point math introductory section.

Input:

v : DOUBLE
Value to convert.

Returns: STRING

A string representation the value of v with 16 significant digits.

Declaration:

```
FUNCTION doubleToStr : STRING;
VAR_INPUT
   v : DOUBLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;

DebugMsg(message := " sin(270) = " + doubleToStr(v := sin(v := degToRad(v :=
270.0))));
DebugMsg(message := " asin(1) = " + doubleToStr(v := radToDeg(v := asin(v :=
1.0))));

END_PROGRAM;
```

4.1.9.28 strToDouble (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 3.10 / 1.00.00

This function converts a string to a DOUBLE.

Note: This function works with DOUBLE, but can also work with FLOAT as described in the Floating-point math introductory section.

Input:

str : STRING
String to convert.

Returns: DOUBLE

The value of the string. If the string can not be parsed, 0 is returned.

Declaration:

```
FUNCTION strToDouble : DOUBLE;
VAR_INPUT
   str : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;
VAR
   a : DOUBLE;
   b : DOUBLE;
END_VAR;

a := strToDouble( str := "1234.56789");
b := strToDouble( str := "-1.23456789e+3");
```

```
DebugMsg( message := "a + b = " + doubleToStr(v:= a + b));

END_PROGRAM;
```

4.1.9.29 degToRad (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 3.10 / 1.00.00

This function converts an angle from degrees to radians.

Note: This function works with DOUBLE, but can also work with FLOAT as described in the Floating-point math introductory section.

Input:
v : DOUBLE
Angle to convert in degrees.

Returns: **DOUBLE**
The value of the angle in radians.

Declaration:

```
FUNCTION degtoRad : DOUBLE;
VAR_INPUT
   v : DOUBLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;

DebugMsg(message := " sin(270) = " + doubleToStr(v := sin(v := degToRad(v :=
270.0))));
DebugMsg(message := " asin(1) = " + doubleToStr(v := radToDeg(v := asin(v :=
1.0))));

END_PROGRAM;
```

4.1.9.30 radToDeg (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 3.10 / 1.00.00

This function converts an angle from radians to degrees.

Note: This function works with DOUBLE, but can also work with FLOAT as described in the Floating-point math introductory section.

Input:
v : DOUBLE
Angle to convert in radians.

Returns: **DOUBLE**

The value of the angle in degrees.

Declaration:

```
FUNCTION radToDeg : DOUBLE;
VAR_INPUT
   v : DOUBLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;

DebugMsg(message := " sin(270) = " + doubleToStr(v := sin(v := degToRad(v :=
270.0))));
DebugMsg(message := " asin(1) = " + doubleToStr(v := radToDeg(v := asin(v :=
1.0))));

END_PROGRAM;
```

4.1.9.31 degToPosition (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 3.10 / 1.00.00

This function converts a coordinate from decimal degrees to GPS position format.

Note: This function works with DOUBLE, but can also work with FLOAT as described in the Floating-point math introductory section.

Input:

v : DOUBLE
Value to convert.

Returns: DINT

The GPS position.

Declaration:

```
FUNCTION degToPosition : DINT;
VAR_INPUT
   v : DOUBLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;

DebugMsg( message := "Pos = (" +
   dintToStr( v:= degToPosition(v := 55.85513)) + ", " +
   dintToStr( v:= degToPosition(v :=  9.850883)) + ")");

END_PROGRAM;
```

4.1.9.32 positionToDeg (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 3.10 / 1.00.00

This function converts a coordinate from GPS position format to decimal degrees.

Note: This function works with DOUBLE, but can also work with FLOAT as described in the Floating-point math introductory section.

Input:

v : DINT
Value to convert.

Returns: DOUBLE

The value in decimal degrees.

Declaration:

```
FUNCTION positionToDeg : DOUBLE;
VAR_INPUT
   v : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;

DebugMsg( message := "Pos = (" +
   doubleToStr( v:= positionToDeg(v := 55513078)) + ", " +
   doubleToStr( v:= positionToDeg(v :=  9510530)) + ")");

END_PROGRAM;
```

4.1.10 SFL: Miscellaneous

4.1.10.1 SFL: Miscellaneous

The VPL environment contains a number of miscellaneous functions and function blocks.

- Debounce — Debounces a signal.
- memcpy — Copies the contents of one memory area to another memory area.
- RS — Set/Reset flip-flop, reset dominant.
- SR — Set/Reset flip-flop, set dominant.

4.1.10.2 DEBOUNCE (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

DEBOUNCE is a function block that will debounce a signal on both the active and inactive state. This is especially useful on noisy digital input signals that are driven by switches, relays etc. that can generate switch noise when switched on or off.

Note: this function block uses 2 TON function blocks.

Input:

in : BOOL (true/false)
This is the signal that will be debounced.

db_time : DINT (0..2147483648)
The number of milliseconds the signal must be active/inactive before the state is considered valid.

Output:

out : BOOL (true/false)
The "in" signal after debounce. It is also free from noises that are shorter than the time defined in "db_time".

Declaration:

```
FUNCTION_BLOCK Debounce;
VAR_INPUT
   in      : BOOL; | The signal that is to be debounced
   db_time : DINT; | The number of milliseconds the 'in' signal must be stable
END_VAR;
VAR_OUTPUT
   out     : BOOL; | The debounced 'in' signal
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   input1 : BOOL; | Digital input
END_VAR;

VAR_OUTPUT
   signal : BOOL; | Noise-free output
END_VAR;
```

```
VAR
   deb : Debounce; // Declare an instance of the DEBOUNCE functionblock
END_VAR;

PROGRAM test;

deb(db_time:=100); // Set debounce time to 100 milliseconds

BEGIN
   deb(in:= input1); // Input the 'input1' signal to the debounce
functionblock
   signal := deb.out; // output will follow the 'input1', but without noise
END;
END_PROGRAM;
```

4.1.10.3 memcpy (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

---

memcpy will copy the contents of one memory area to another memory area.

Please note that memcpy does not support the copying of overlapped source and destination.

Input:

dst : PTR
Pointer to the destination memory.

src : PTR
Pointer to the source memory.

len : INT
Number of bytes to be copied.

Returns:

None

Declaration:

```
FUNCTION memcpy;
VAR_INPUT
   dst : PTR;
   src : PTR;
   len : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   data_1 : ARRAY[0..100] OF SINT;
   data_2 : ARRAY[0..100] OF SINT;
END_VAR;

PROGRAM test;

BEGIN
   ...
   memcpy(dst:=ADDR(data_1), src:=ADDR(data_2), len:=SIZEOF(data_1));
   // data_1 contains now the contents of data_2
```

```
   ...
END;

END_PROGRAM;
```

4.1.10.4 RS (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

RS is a function block that acts as a Reset/Set flip flop. On a leading edge on the "S" input, the "q" output will stay active. On a leading edge on the "R1" input, the "out" will stay inactive. If a leading edge is seen at the same time on both "S" and "R1", the reset function "wins" and the '"out" will stay inactive.
Please see the SR function block for a "Set" dominant flip-flop.

Input:

S : BOOL (true/false)
A leading edge on this signal will set the "out" to active.

R1 : BOOL (true/false)
A leading edge on this signal will set the "out" to inactive.

Output:

q : BOOL (true/false)
This signal will be active if a leading edge is seen on "S", and it will be inactive if a leading edge is seen on "R1".

Declaration:

```
FUNCTION_BLOCK RS;
VAR_INPUT
   S  : BOOL; | Set signal
   R1 : BOOL; | Reset signal
END_VAR;
VAR_OUTPUT
   q  : BOOL; | The output from the flip-flop
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   set   : BOOL; | Digital input for set
   reset : BOOL; | Digital input for reset
END_VAR;

VAR_OUTPUT
   signal : BOOL; | Output from the flip-flop
END_VAR;

VAR
   flipflop : RS; // Declare an instance of the RS functionblock
END_VAR;

PROGRAM test;

BEGIN
   flipflop(S:= set, R1:=reset); // Input the signals to the flip-flop
   signal := flipflop.q; // output from the flip-flop
```

```
END;
END_PROGRAM;
```

4.1.10.5 SR (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

SR is a function block that acts as a Set/Reset flip flop. On a leading edge on the "S1" input, the "q" output will stay active. On a leading edge on the "R" input, the "out" will stay inactive. If a leading edge is seen at the same time on both "S1" and "R", the set function "wins", and the "out" will stay active.
Please see the RS function block for a "Reset" dominant flip-flop.

Input:

S1 : BOOL (true/false)
A leading edge on this signal will set the "out" to active.

R : BOOL (true/false)
A leading edge on this signal will set the "out" to inactive.

Output:

q : BOOL (true/false)
This signal will be active if a leading edge is seen on "S1", and it will be inactive if a leading edge is seen on "R".

Declaration:

```
FUNCTION_BLOCK SR;
VAR_INPUT
   S1 : BOOL; | Set signal
   R  : BOOL; | Reset signal
END_VAR;
VAR_OUTPUT
   q  : BOOL; | The output from the flip-flop
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   set   : BOOL; | Digital input for set
   reset : BOOL; | Digital input for reset
END_VAR;

VAR_OUTPUT
   signal : BOOL; | Output from the flip-flop
END_VAR;

VAR
   flipflop : SR; // Declare an instance of the SR functionblock
END_VAR;

PROGRAM test;

BEGIN
   flipflop(S1:= set, R:=reset); // Input the signals to the flip-flop
   signal := flipflop.q; // output from the flip-flop
END;
END_PROGRAM;
```

4.1.11 SFL: Mutex functions (multithreading)

4.1.11.1 mx: Mutex functions

The mutex (Mutual Exclusion) data type is most importantly used for implementing critical sections in a multithreading VPL program. Critical sections are used in the cases where more than one thread is accessing the same shared resource - like a global variable/data structure accessed from several threads.
When using a mutex, it is ensured that only one thread will be executing in the critical section at one time. Other threads that also want to enter the critical section will be blocked until the thread executing in the critical section leaves it. Please also see the section on thread synchronization.

Before a mutex variable can be used, it must be initialized by using the mxInit() function. When requesting ownership of the mutex (when entering a critical section), the function mxLock() must be called. When releasing ownership of the mutex (when leaving a critical section), the function mxUnlock() must be called.

The following mutex functions exists:

- mxInit — Initializes a mutex.
- mxDestroy — Destroys a mutex.
- mxStatus — Queries the status of a mutex.
- mxLock — Locks a mutex.
- mxUnlock — Unlocks a mutex.

4.1.11.2 mxInit (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

The mxInit() function will initialize a MUTEX variable.
The mxInit() function must be called before any other operations can be performed on the mutex.

Input:
None

Returns: **MUTEX**
The initialized mutex.

Declaration:

```
FUNCTION mxInit : MUTEX;
```

Example:

```
INCLUDE rtcu.inc

VAR
   mxCnt : MUTEX;
   Count : DINT := 0;
END_VAR;

THREAD_BLOCK Thread_A;
WHILE TRUE DO
   mxLock(mx:=mxCnt);
   Count := Count + 1;
```

```
   mxUnlock(mx:=mxCnt);
END_WHILE;
END_THREAD_BLOCK;

THREAD_BLOCK Thread_B;
WHILE TRUE DO
   mxLock(mx:=mxCnt);
   Count := Count + 5;
   mxUnlock(mx:=mxCnt);
END_WHILE;
END_THREAD_BLOCK;

PROGRAM test;
VAR
   TA    : Thread_A;
   TB    : Thread_B;
   i     : INT;
END_VAR;

mxCnt := mxInit();

TA();
TB();

BEGIN
   Sleep(delay:=1000);
   DebugFmt(message:="Count: \4",v4:=Count);
END;

END_PROGRAM;
```

4.1.11.3 mxDestroy (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

The mxDestory() function will destroy a MUTEX variable.
After the call to mxDestroy(), the mutex variable cannot be used in further operations.

Input:

mx : MUTEX
The mutex to destroy.

Returns: INT

0 Mutex is destroyed.
1 Mutex is not initialized.
2 Mutex is busy.

Declaration:

```
FUNCTION mxDestroy : INT;
VAR_INPUT
   mx : MUTEX;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

THREAD_BLOCK Thread;
VAR_INPUT
```

```
   mx : MUTEX;
   ...
END_VAR;
   ...
   // Do until mutex is destroyed
   WHILE mxStatus(mx:=mx) <> 1 DO
      ...
   END_WHILE;
END_THREAD_BLOCK;

PROGRAM MutexTest;
VAR
   mx : MUTEX;
   th : ARRAY[1..3] OF Thread;
   i  : INT;
   ...
END_VAR;
   // Initialize Mutex
   mx := mxInit();
   IF mxStatus(mx:=mx) = 1 THEN DebugMsg(message:="mxCnt failed to init!");
END_IF;

   // Initialize Threads
   FOR i := 1 TO 3 DO th[i](mx:=mx,...); END_FOR;

   ...

BEGIN
   ...
   // Only when mutex exsists
   IF mxStatus(mx:=mx) <> 1 THEN
      ...
   END_IF;
   ...
   // Stop threads and destroy mutex
   IF ... THEN
      // Try to destroy until success
      WHILE mxDestroy(mx:=mx) = 2 DO END_WHILE;   // Ignore allready destroyed
mutex
   END_IF;
   ...
END;

END_PROGRAM;
```

4.1.11.4 mxStatus (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

The mxStatus function will return the status on the specified MUTEX variable.

Input:

mx : MUTEX
The mutex to query the status on.

Returns: **INT**

0 Mutex is unlocked.
1 Mutex is not initialized.
2 Mutex is locked.

Declaration:

```
FUNCTION mxStatus : INT;
VAR_INPUT
   mx : MUTEX;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

THREAD_BLOCK Thread;
VAR_INPUT
   mx : MUTEX;
   ...
END_VAR;
   ...
   // Do until mutex is destroyed
   WHILE mxStatus(mx:=mx) <> 1 DO
      ...
   END_WHILE;
END_THREAD_BLOCK;

PROGRAM MutexTest;
VAR
   mx : MUTEX;
   th : ARRAY[1..3] OF Thread;
   i  : INT;
   ...
END_VAR;
   // Initialize Mutex
   mx := mxInit();
   IF mxStatus(mx:=mx) = 1 THEN DebugMsg(message:="mxCnt failed to init!");
END_IF;

   // Initialize Threads
   FOR i := 1 TO 3 DO th[i](mx:=mx,...); END_FOR;

   ...

BEGIN
   ...
   // Only when mutex exists
   IF mxStatus(mx:=mx) <> 1 THEN
      ...
   END_IF;
   ...
   // Stop threads and destroy mutex
   IF ... THEN
      // Try to destroy until success
      WHILE mxDestroy(mx:=mx) = 2 DO END_WHILE;   // Ignore already destroyed
mutex
   END_IF;
   ...
END;

END_PROGRAM;
```

4.1.11.5 mxLock (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

The mxLock() function will lock the specified MUTEX variable.
This mutex mechanism ensures that only one thread will continue to run between the call to mxLock() and mxUnlock(). If the mutex is already locked, the calling thread will be blocked on a FIFO waiting queue until the current owner of the mutex calls mxUnlock(). If the owner of the mutex calls mxLock() several times, it will not block, but the number of mxLock() and mxUnlock() called must balance, and the mutex will not be released before the last and closing call to mxUnlock() is performed.

The mutex mechanism is traditionally used for implementing critical sections. Also see the section on thread synchronization for more information.

Input:

mx : MUTEX
The MUTEX to lock.

Returns: INT

0 Mutex is locked.
1 Mutex is not initialized.

Declaration:

```
FUNCTION mxLock : INT;
VAR_INPUT
   mx : MUTEX;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   mxCnt : MUTEX;
   Count : DINT := 0;
END_VAR;

THREAD_BLOCK Thread_A;
WHILE TRUE DO
   mxLock(mx:=mxCnt);
   Count := Count + 1;
   mxUnlock(mx:=mxCnt);
END_WHILE;
END_THREAD_BLOCK;

THREAD_BLOCK Thread_B;
WHILE TRUE DO
   mxLock(mx:=mxCnt);
   Count := Count + 5;
   mxUnlock(mx:=mxCnt);
END_WHILE;
END_THREAD_BLOCK;

PROGRAM test;
VAR
```

```
   TA     : Thread_A;
   TB     : Thread_B;
   i      : INT;
END_VAR;

mxCnt := mxInit();

TA();
TB();

BEGIN
   Sleep(delay:=1000);
   DebugFmt(message:="Count: \4",v4:=Count);
END;

END_PROGRAM;
```

4.1.11.6 mxUnlock (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

The mxUnlock() function will unlock a MUTEX variable previously locked by the mxLock() function.
A mutex can only be unlocked by the owner of the mutex.

The mutex mechanism is traditionally used for implementing critical sections. Also see the section on thread syncronization for more information.

Input:

mx : MUTEX
The MUTEX to unlock.

Returns: INT

0 Mutex is unlocked.
1 Mutex is not initialized.
2 Mutex is locked by another thread.

Declaration:

```
FUNCTION mxUnlock : INT;
VAR_INPUT
   mx : MUTEX;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   mxCnt : MUTEX;
   Count : DINT := 0;
END_VAR;

THREAD_BLOCK Thread_A;
WHILE TRUE DO
   mxLock(mx:=mxCnt);
   Count := Count + 1;
   mxUnlock(mx:=mxCnt);
```

```
END_WHILE;
END_THREAD_BLOCK;

THREAD_BLOCK Thread_B;
WHILE TRUE DO
   mxLock(mx:=mxCnt);
   Count := Count + 5;
   mxUnlock(mx:=mxCnt);
END_WHILE;
END_THREAD_BLOCK;

PROGRAM test;
VAR
   TA    : Thread_A;
   TB    : Thread_B;
   i     : INT;
END_VAR;

mxCnt := mxInit();

TA();
TB();

BEGIN
   Sleep(delay:=1000);
   DebugFmt(message:="Count: \4",v4:=Count);
END;

END_PROGRAM;
```

4.1.12 SFL: Persistent memory

4.1.12.1 SFL: Persistent memory

Persistent memory is storage that will keep its content even when the RTCU is powered down. By using the SaveString/LoadString and SaveData/LoadData, it is possible to save and load strings/data from a specified location in the persistent memory.

The persistent memory of the RTCU devices is implemented by using two different memory technologies. The largest part of persistent memory uses FLASH technology. The FLASH technology has a limitation in that it can only be written a finite number of times. If written more than that, the FLASH memory will no longer be stable, which then results in loss and corruption of data. To overcome this problem, the RTCU devices also contain FRAM-based memory, which does not have any limitations with respect to the number of times it can be written to.

The FLASH-based persistent memory is composed of entries with a maximum length of 255 bytes. Each entry can be rewritten a minimum of 100,000 times before flash wear-out will occur.

For FRAM-based persistent memory, there up to 100 separate entries available - each also with a maximum length of 255 bytes.

- SaveString — Saves a string to persistent FLASH memory.
- LoadString — Loads a string from persistent FLASH memory.
- SaveData — Saves a block of memory to persistent FLASH memory.
- LoadData — Loads a block of memory from persistent FLASH memory.
- SaveStringF — Saves a string to persistent FRAM memory.
- LoadStringF — Loads a string from persistent FRAM memory.
- SaveDataF — Saves a block of memory to persistent FRAM memory.
- LoadDataF — Loads a block of memory from persistent FRAM memory.
- SaveStringX — Saves a string to persistent extended FLASH memory.
- LoadStringX — Loads a string from persistent extended FLASH memory.
- SaveDataX — Saves a block of memory to persistent extended FLASH memory.
- LoadDataX — Loads a block of memory from persistent extended FLASH memory.
- GetFlashXSize — Gets the size of the persistent extended FLASH memory.

4.1.12.2 SaveString (Function)

```
Architecture:       X32 / NX32 / NX32L
Device support:     ALL
Firmware version:   1.00 / 1.00.00
```

SaveString will save a string to a specific location in the persistent FLASH memory.

Note: there is a limited number of writes to the Flash Memory that make use of this function. Please read the Persistent Memory introductory section.

Input:

index : INT (1..192)
Location number the string should be saved in.

str : STRING
String that is to be saved.

Returns: **INT**

(Only reported from firmware version 5.00 / R1.30.00)
0 - Success.
1 - Invalid index.
3 - Invalid length.

Declaration:

```
FUNCTION SaveString;
VAR_INPUT
   index : INT;
   str   : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   SaveString(index:=10, str:="hello world");
   // Persistent memory location number 10 will contain "hello world" after
the call
   ...
END;

END_PROGRAM;
```

4.1.12.3 SaveStringF (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

SaveStringF will save a string to a specific location in the persistent FRAM memory.

A total of 20 (X32) / 100 (NX32/NX32L) FRAM entries are available.

Input:

index : INT (1..20, 1..100)
Location number the string should be saved in.

str : STRING
String that is to be saved.

Returns: INT

(Only reported from firmware version 5.00 / R1.30.00)
0 - Success.
1 - Invalid index.
3 - Invalid length.

Declaration:

```
FUNCTION SaveStringF;
VAR_INPUT
   index : INT;
   str   : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   SaveStringF(index:=10, str:="hello world");
   // Persistent memory location number 10 will contain "hello world" after
the call
   ...
END;

END_PROGRAM;
```

4.1.12.4 SaveStringX (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

SaveStringX will save a string to a specific location in the persistent extended FLASH memory.

Use the GetFlashXSize function to determine the highest location it is possible to write to.

Note: there is a limited number of writes to the Flash Memory that make use of this function. Please read the Persistent Memory introductory section.

Input:

index : INT (1..max)
Location number the string should be saved in.

str : STRING
String that is to be saved.

Returns: INT

(Only reported from firmware version 5.00 / R1.30.00)

0 - Success.
1 - Invalid index.
3 - Invalid length.

Declaration:

```
FUNCTION SaveStringX;
VAR_INPUT
   index : INT;
   str   : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   SaveStringX(index:=10, str:="hello world");
   // Persistent memory location number 10 will contain "hello world" after
```

```
the call
   ...
END;

END_PROGRAM;
```

4.1.12.5 LoadString (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

LoadString will load a string from a specific location in the persistent FLASH memory.

Input:

index : INT (1..192)
Location number the string should be loaded from.

Returns: STRING

The string from the specified location.

Declaration:

```
FUNCTION LoadString : STRING;
VAR_INPUT
   index : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   result : STRING;
END_VAR;

BEGIN
   ...
   result:=LoadString(index:=10);
   // If persistent memory location number 10 contains "hello world", then
result
   // will contain the same text after the call
   ...
END;

END_PROGRAM;
```

4.1.12.6 LoadStringF (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

LoadStringF will load a string from a specific location in the persistent FRAM memory.

A total of 20 (X32) / 100 (NX32/NX32L) FRAM entries are available.

Input:

index : INT (1..20 or 1..100)
Location number the string should be loaded from.

Returns: STRING
The string from the specified location.

Declaration:
```
FUNCTION LoadStringF : STRING;
VAR_INPUT
   index : INT;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   result : STRING;
END_VAR;

BEGIN
   ...
   result:=LoadStringF(index:=10);
   // If persistent memory location number 10 contains "hello world", then
result
   // will contain the same text after the call
   ...
END;

END_PROGRAM;
```

4.1.12.7 LoadStringX (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

LoadStringX will load a string from a specific location in the persistent extended FLASH memory.

Use the GetFlashXSize function to determine the highest location it is possible to read from.

Note: not all RTCU devices support extended flash memory.

Input:
index : INT (1..max)
Location number the string should be loaded from.

Returns: STRING
The string from the specified location.

Declaration:
```
FUNCTION LoadStringX : STRING;
VAR_INPUT
   index : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   result : STRING;
END_VAR;

BEGIN
   ...
   result:=LoadStringX(index:=10);
   // If persistent memory location number 10 contains "hello world", then
result
   // will contain the same text after the call
   ...
END;

END_PROGRAM;
```

4.1.12.8 SaveData (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

SaveData will save a block of memory to a specific location in the persistent FLASH memory.

Note: there is a limited number of writes to the Flash Memory that make use of this function. Please read the Persistent Memory introductory section.

Input:

index : INT (1..192)
Location number the block of memory should be saved in.

data : ADR
Address of the memory block that should be saved.

datasize : INT (1..255)
Maximum size to save.

Returns: INT

(Only reported from firmware version 5.00 / R1.30.00)

0 - Success.
1 - Invalid index.
2 - Data is missing.
3 - Invalid size.

Declaration:

```
FUNCTION SaveData;
VAR_INPUT
   index    : INT;
   data     : PTR;
   datasize : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
```

```
PROGRAM test;

VAR
   xx    : ARRAY[1..100] OF SINT;
END_VAR;

BEGIN
   ...
   SaveData(index:=10, data:=ADDR(xx), datasize:=SIZEOF(xx));
   // Persistent memory location number 10 will contain the contents of 'xx'
after the call
   ...
END;

END_PROGRAM;
```

4.1.12.9 SaveDataF (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

SaveDataF will save a block of memory to a specific location in the persistent FRAM memory.

A total of 20 (X32) / 100 (NX32/NX32L) FRAM entries are available.

Input:

index : INT (1..20 or 1..100)
Location number the block of memory should be saved in.

data : ADR
Address of the memory block that should be saved.

datasize : INT (1..255)
Maximum size to save.

Returns: INT

(Only reported from firmware version 5.00 / R1.30.00)

0 - Success.
1 - Invalid index.
2 - Data is missing.
3 - Invalid size.

Declaration:

```
FUNCTION SaveDataF;
VAR_INPUT
   index    : INT;
   data     : PTR;
   datasize : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

VAR
   xx    : ARRAY[1..100] OF SINT;
END_VAR;
```

```
BEGIN
   ...
   SaveDataF(index:=10, data:=ADDR(xx), datasize:=SIZEOF(xx));
   // Persistent memory location number 10 will contain the contents of 'xx'
after the call
   ...
END;

END_PROGRAM;
```

4.1.12.10 SaveDataX (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

SaveDataX will save a block of memory to a specific location in the persistent extended FLASH memory.

Use the GetFlashXSize function to determine the highest location it is possible to write to.

Note: there is a limited number of writes to the Flash Memory that make use of this function. Please read the Persistent Memory introductory section.

Input:

index : INT (1..max)
Location number the block of memory should be saved in.

data : ADR
Address of the memory block that should be saved.

datasize : INT (1..255)
Maximum size to save.

Returns: INT

(Only reported from firmware version 5.00 / R1.30.00)

0 - Success.
1 - Invalid index.
2 - Data is missing.
3 - Invalid size.

Declaration:

```
FUNCTION SaveDataX;
VAR_INPUT
   index    : INT;
   data     : PTR;
   datasize : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

VAR
   xx    : ARRAY[1..100] OF SINT;
END_VAR;
```

```
BEGIN
   ...
   SaveDataX(index:=10, data:=ADDR(xx), datasize:=SIZEOF(xx));
   // Persistent memory location number 10 will contain the contents of 'xx'
after the call
   ...
END;

END_PROGRAM;
```

4.1.12.11 LoadData (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

LoadData will load a block of memory from a specific location in the persistent FLASH memory.

Input:

index : INT (1..192)
Location number the data should be loaded from.

data : ADR
Address of the memory block that receives the loaded data.

datasize : INT (1..255)
Maximum size to load into "ptr".

Returns: INT

The actual length loaded from the specified index (this will be 0 if there is no valid data at the index - i.e. a string is stored at this position).

Declaration:

```
FUNCTION LoadData : INT;
VAR_INPUT
   index    : INT;
   data     : PTR;
   datasize : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

VAR
   xx     : ARRAY[1..100] OF SINT;
   length : INT;
END_VAR;

BEGIN
   ...
   length:=LoadData(index:=10, data:=ADDR(xx), datasize:=SIZEOF(xx));
   // Persistent memory location number 10 will be loaded into 'xx'
   ...
END;
```

```
END_PROGRAM;
```

4.1.12.12 LoadDataF (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

LoadDataF will load a block of memory from a specific location in the persistent FRAM memory.

A total of 20 (X32) / 100 (NX32/NX32L) FRAM entries are available.

Input:

index : INT (1..20 or 1..100)
Location number the data should be loaded from.

data : ADR
Address of the memory block that receives the loaded data.

datasize : INT (1..255)
Maximum size to load into "ptr".

Returns: INT

The actual length loaded from the specified index (this will be 0 if there is no valid data at the index - i.e. a string is stored at this position).

Declaration:

```
FUNCTION LoadDataF : INT;
VAR_INPUT
   index    : INT;
   data     : PTR;
   datasize : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

VAR
   xx     : ARRAY[1..100] OF SINT;
   length : INT;
END_VAR;

BEGIN
   ...
   length:=LoadDataF(index:=10, data:=ADDR(xx), datasize:=SIZEOF(xx));
   // Persistent memory location number 10 will be loaded into 'xx'
   ...
END;

END_PROGRAM;
```

4.1.12.13 LoadDataX (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

LoadDataX will load a block of memory from a specific location in the persistent FRAM memory.

Use the GetFlashXSize function to determine the highest location it is possible to read from.

Note: not all RTCU devices support extended flash memory.

Input:

index : INT (1..max)
Location number the data should be loaded from.

data : ADR
Address of the memory block that receives the loaded data.

datasize : INT (1..255)
Maximum size to load into "ptr".

Returns: INT

The actual length loaded from the specified index (this will be 0 if there is no valid data at the index - i.e. a string is stored at this position).

Declaration:

```
FUNCTION LoadDataX : INT;
VAR_INPUT
   index    : INT;
   data     : PTR;
   datasize : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

VAR
   xx     : ARRAY[1..100] OF SINT;
   length : INT;
END_VAR;

BEGIN
   ...
   length:=LoadDataX(index:=10, data:=ADDR(xx), datasize:=SIZEOF(xx));
   // Persistent memory location number 10 will be loaded into 'xx'
   ...
END;

END_PROGRAM;
```

4.1.12.14 GetFlashXSize (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

GetFlashXSize will return the maximum index that can be used in the persistent extended FLASH.

Note:
Not all RTCU devices support extended flash memory. When not supported, this function will return 0 (zero).

Input:
None

Returns: INT
The highest location that can be used in the extended FLASH persistent memory.

Declaration:
```
FUNCTION GetFlashXSize : INT;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;

VAR
   max : INT;
END_VAR;

   max := GetFlashXSize();

BEGIN
   ...
END;

END_PROGRAM;
```

4.1.13 SFL: Real Time Clock functions

4.1.13.1 RTC: Functions for Real Time Clock

The clock functions allows the program to interact with the real time clock on the RTCU platform. By using these functions, it is possible to both set and read the built-in clock.

clockGet	Reads the time-of-day.
clockNow	Reads the time-of-day as a number of seconds since 1980-1-1 00:00:00 (LINSEC)
clockSet	Sets the time-of-day.
clockTimeToLinsec	Converts a time in yy,mm,dd,hh,mm,ss format to number of seconds since 1980-1-1 00:00:00.
clockLinsecToTime	Converts a time in number of seconds since 1980-1-1 00:00:00 to yy,mm,dd,hh,mm,ss format.
clockAlarm	Sets an alarm.
clockDayTimer	Sets a repeating start and stop time.
clockWeekTimer	Sets a repeating alarm on a specific weekday at a specific time.
clockSetWakeup	Configures the system to wake from pmSuspend at a specific time.

LINSEC
In many of the functions, the term LINSEC is used which stands for Linear Seconds.
In the RTCU concept, a LINSEC value is the number of seconds since 1980-1-1 00:00:00.

4.1.13.2 clockGet (Functionblock)

```
Architecture:       X32 / NX32 / NX32L
Device support:     ALL
Firmware version:   1.00 / 1.00.00
```

This function block can be used to read the real time clock on the RTCU device.

Input:
None

Output:
year : INT (1980 .. 2048)
The current year.

month : SINT (1 .. 12)
The current month.

day : SINT (1 .. 31)
The current date.

hour: SINT (0 .. 23)
The current hour.

minute : SINT (0 .. 59)
The current minute.

second : SINT (0 .. 59)
The current second.

linsec : DINT
Seconds since 1980-1-1 00:00:00.

Declaration:

```
FUNCTION_BLOCK clockGet;
VAR_OUTPUT
   Year   : INT;   // 1980..2048
   Month  : SINT;  // 01..12
   Day    : SINT;  // 01..31
   Hour   : SINT;  // 00..23
   Minute : SINT;  // 00..59
   Second : SINT;  // 00..59
   Linsec : DINT;  // Seconds since 1980-1-1 00:00:00
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   clock : clockGet; // Declare an instance of the clockGet functionblock
END_VAR;

PROGRAM test;

BEGIN
   clock();
   // Check if it is December
   IF clock.month = 12 THEN
      ...
   END_IF;
   ...
END;

END_PROGRAM;
```

4.1.13.3 clockNow (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

This function to read the current time - expressed as seconds since 1980-1-1 00:00:00 (Also called **LINSEC**).

Input:
None

Returns: DINT
The current time as seconds since 1980-1-1 00:00:00 (LINSEC).

Declaration:

```
FUNCTION clockNow : DINT;
```

Example:

```
INCLUDE rtcu.inc

VAR
```

```
   a : DINT;
END_VAR;

PROGRAM test;

BEGIN
   a := clockNow();
   // 'a' contains the current time as number of seconds since 1980-1-1
00:00:00
   ...
END;

END_PROGRAM;
```

4.1.13.4 clockTimeToLinsec (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

This function can be used to convert a time in year, month, day, hour, minute, and second format to number of seconds since 1980-1-1 00:00:00.

Input:

year : INT (-1, 1980 .. 2048) (*Default* -1)
The current year.

month : SINT (-1, 1 .. 12) (*Default* -1)
The current month.

day : SINT (-1, 1 .. 31) (*Default* -1)
The current date.

hour: SINT (-1 .. 23) (*Default* -1)
The current hour.

minute : SINT (-1 .. 59) (*Default* -1)
The current minute.

second : SINT (-1 .. 59) (*Default* -1)
The current second.

Returns: DINT

The specified time in seconds since 1980-1-1 00:00:00.

Declaration:

```
FUNCTION clockTimeToLinsec : DINT;
VAR_INPUT
   Year   : INT  := -1;  // 1980..2048
   Month  : SINT := -1;  // 01..12
   Day    : SINT := -1;  // 01..31
   Hour   : SINT := -1;  // 00..23
   Minute : SINT := -1;  // 00..59
   Second : SINT := -1;  // 00..59
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

VAR
   a : DINT;
END_VAR;

BEGIN
   ...
   // Convert 2001-12-24 12:00:30 to number of seconds since 1980-1-1 00:00:00
   a := clockTimeToLinsec(year := 2001, month := 12, day := 24, hour:=12,
minute:=0, second:=30);
   ...
END;

END_PROGRAM;
```

4.1.13.5 clockLinsecToTime (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

This function block can be used to convert number of seconds since 1980-1-1 00:00:00 to year, month, day, hour, minute, second, and weekday format

Input:

linsec : DINT
Number of seconds since 1980-1-1 00:00:00

Output:

year : INT (1980 .. 2048)
The current year.

month : SINT (1 .. 12)
The current month.

day : SINT (1 .. 31)
The current date.

hour: SINT (0 .. 23)
The current hour.

minute : SINT (0 .. 59)
The current minute.

second : SINT (0 .. 59)
The current second.

dayofweek : SINT (1 .. 7)
The current day of week, 1 = Monday, 2 = Tuesday... 7 = Sunday.

Declaration:

```
FUNCTION_BLOCK clockLinsecToTime;
VAR_INPUT
   Linsec    : DINT;   // Seconds since 1980-1-1 00:00:00
END_VAR;
```

```
VAR_OUTPUT
   Year      : INT;    // 1980..2048
   Month     : SINT;   // 01..12
   Day       : SINT;   // 01..31
   Hour      : SINT;   // 00..23
   Minute    : SINT;   // 00..59
   Second    : SINT;   // 00..59
   DayOfWeek : SINT;   // 1..7
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   clock : clockLinsecToTime; // Declare an instance of the clockLinsecToTime
functionblock
END_VAR

PROGRAM test;

BEGIN
   ...
   clock(Linsec:=30);
   // clock.year, clock.month, clock.day, clock.hour, clock.minute,
clock.second is now 1980-1-1 00:00:30
   ...
END;

END_PROGRAM;
```

4.1.13.6 clockSet (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

This function can be used to set the real time clock on the RTCU device. If any of the input parameters are not referenced in the call, they will not be modified in the real time clock.

Input:

year : INT (-1, 2000 .. 2048) (*Default* -1)
The current year - if "-1", then not updated.

month : SINT (-1, 1 .. 12) (*Default* -1)
The current month - if "-1", then not updated.

day : SINT (-1, 1 .. 31) (*Default* -1)
The current date - if "-1", then not updated.

hour: SINT (-1 .. 23) (*Default* -1)
The current hour - if "-1", then not updated.

minute : SINT (-1 .. 59) (*Default* -1)
The current minute - if "-1", then not updated.

second : SINT (-1 .. 59) (*Default* -1)
The current second - if "-1", then not updated.

linsec : DINT (-1 .. 2147483647) (*Default* -1)
Seconds since 1980-1-1 00:00:00. If all other input variables are set to their default state (-1), and Linsec is set to something different than -1, the new time will be calculated as Linsec numbers of seconds since 1980-1-1 00:00:00.

Returns:
None

Declaration:
```
FUNCTION clockSet;
VAR_INPUT
   Year   : INT  := -1;  // 2000..2048
   Month  : SINT := -1;  // 01..12
   Day    : SINT := -1;  // 01..31
   Hour   : SINT := -1;  // 00..23
   Minute : SINT := -1;  // 00..59
   Second : SINT := -1;  // 00..59
   Linsec : DINT := -1;  // Seconds since 1980-1-1 00:00:00
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;

   // Set the real time clock to 17th of May 2000. Please note that the hour,
minute, second and day are not assigned new values.
   clockSet(year := 2000, month := 5, day := 17);

BEGIN
   ...
END;

END_PROGRAM;
```

4.1.13.7 clockAlarm (Functionblock)

Architecture: **X32 / NX32 / NX32L**
Device support: **ALL**
Firmware version: **1.00 / 1.00.00**

---

clockAlarm is a convenient way of setting an alarm at a specific time. This function block lets you set alarms that will occur periodically. If one or more of the input parameters to the function block are omitted, they are not used in setting the alarm time.

Input:

enable : BOOL
Set to true to activate the timer.

month : INT (-1, 1 .. 12) (*Default: -1 (not used in comparison)*)
The month.

day : INT (-1, 1 .. 31) (*Default: -1 (not used in comparison)*)
The date.

hour: INT (-1 .. 23) (*Default: -1 (not used in comparison)*)
The hour.

minute : INT (-1 .. 59) (*Default: -1 (not used in comparison)*)
The minute.

second : INT (-1 .. 59) (*Default: -1 (not used in comparison)*)
The second.

Output:

q : BOOL
When the time is reached, q will become TRUE.

Declaration:

```
FUNCTION_BLOCK clockAlarm;
VAR_INPUT
   enable : BOOL := false;
   Month  : INT  := -1;   // 01..12
   Day    : INT  := -1;   // 01..31
   Hour   : INT  := -1;   // 00..23
   Minute : INT  := -1;   // 00..59
   Second : INT  := -1;   // 00..59
END_VAR;
VAR_OUTPUT
   q      : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   Alarm_1 : clockAlarm; // Declare an instance of the clockAlarm
functionblock
   Alarm_2 : clockAlarm; // Declare an instance of the clockAlarm
functionblock
   Alarm_3 : clockAlarm; // Declare an instance of the clockAlarm
functionblock
END_VAR;

// Set the alarm to December 24th. Please note that the hour, minute and
second are not assigned, and therefore not used in the alarm time.
Alarm_1(enable:=TRUE, month:=12, day:=24); // This will give an alarm at
midnight 00:00:00 the 24th of December
// Set the alarm to December 31th every year at 23:55. Please note that the
second are not assigned, and therefore not used in the alarm time.
Alarm_2(enable:=TRUE, month:=12, day:=31, hour:=23, minute:=55); // This will
give an alarm exactly at 23:55:00 the 31th of December
// Set the alarm to occur on each full hour.
Alarm_3(enable:=TRUE, minute:=0); // This will give an alarm at the beginning
of each hour
BEGIN
   Alarm_1();
   Alarm_2();
   Alarm_3();
   ...
   IF Alarm_1.q THEN
      DebugMsg(message:="Merry Christmas");
   END_IF;
   IF Alarm_2.q THEN
      DebugMsg(message:="New year in 5 minutes !");
   END_IF;
   IF Alarm_3.q THEN
```

```
      DebugMsg(message:="A new hour has begun");
   END_IF;
  ...
END;
END_PROGRAM;
```

4.1.13.8 clockDayTimer (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

---

clockDayTimer is a simple repeating timer. It has a start- and stop time. When the start time is reached, the q output will go true. Q will go false when the stop time is reached. This will repeat every day.

Input:

enable : BOOL (*Default: FALSE (Timer is NOT started)*)
Timer is enabled when enable is true.

start_hour: INT (-1 .. 23) *Default: -1 (not used in comparison)*
The starting hour.

start_minute : INT (-1 .. 59) (*Default: -1 (not used in comparison)*)
The starting minute.

start_second : INT (0 .. 59) (*Default: 0*)
The starting second.

stop_hour: INT (-1 .. 23) (*Default: -1 (not used in comparison)*)
The ending hour.

stop_minute : INT (-1 .. 59) (*Default: -1 (not used in comparison)*)
The ending minute.

stop_second : INT (0 .. 59) (*Default: 0*)
The ending second.

Output:

q : BOOL
When start time is reached, q will go true, and it will go false when the stop time is reached

Declaration:

```
FUNCTION_BLOCK clockDayTimer;
VAR_INPUT
   enable       : BOOL := FALSE;
   start_hour   : INT  := -1;   // 00..23
   start_minute : INT  := -1;   // 00..59
   start_second : INT  := 0;   // 00..59
   stop_hour    : INT  := -1;   // 00..23
   stop_minute  : INT  := -1;   // 00..59
   stop_second  : INT  := 0;   // 00..59
END_VAR;
VAR_OUTPUT
   q            : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_OUTPUT
   SignalLamp : BOOL; | Output that drives the signal lamp
END_VAR;

PROGRAM test;
VAR
   DayTimer : clockDayTimer; // Declare an instance of the clockDayTimer
functionblock
END_VAR;

// Set the timer to start at 12:00 and end at 13:30 (every day)
DayTimer(enable:=TRUE, start_hour:= 12, start_minute:=0, stop_hour:=13,
stop_minute:=30); // Start and stop time for the timer
BEGIN
   DayTimer();
   ...
   // SignalLamp will be on from 12:00 until 13:30 every day
   SignalLamp := DayTimer.q;
   ...
END;
END_PROGRAM;
```

4.1.13.9 clockWeekAlarm (Functionblock)

Architecture: **X32 / NX32 / NX32L**
Device support: **ALL**
Firmware version: **1.00 / 1.00.00**

clockWeekAlarm is a convenient way of setting an alarm at a specific time on a specific weekday.

Input:

enable : BOOL
The timer will be initialized and started on the leading edge of the enable input.

dayofweek : SINT (1 .. 7)
The day of the week the alarm should trigger. 1 is Monday, 7 is Sunday.

hour: SINT (0 .. 23)
The hour for the alarm.

minute : SINT (0 .. 59)
The minute for the alarm.

second : SINT (0 .. 59)
The second for the alarm

Output:

q : BOOL
When the alarm time is reached, q will be TRUE.

Declaration:

```
FUNCTION_BLOCK clockWeekAlarm;
VAR_INPUT
   enable    : BOOL R_EDGE;
   DayOfWeek : SINT;   // 01..31
```

```
   Hour      : SINT;   // 00..23
   Minute    : SINT;   // 00..59
   Second    : SINT;   // 00..59
END_VAR;
VAR_OUTPUT
   q         : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   Alarm_1 : clockWeekAlarm;
   Alarm_2 : clockWeekAlarm;
   Alarm_3 : clockWeekAlarm;
   Alarm_4 : clockWeekAlarm;
END_VAR;

PROGRAM test;

Alarm_1(enable:=TRUE, DayOfWeek:=1, Hour:=18);
Alarm_2(enable:=TRUE, DayOfWeek:=2, Hour:=18);
Alarm_3(enable:=TRUE, DayOfWeek:=4, Hour:=18);
Alarm_4(enable:=TRUE, DayOfWeek:=6, Hour:=18);

BEGIN
   Alarm_1();
   Alarm_2();
   Alarm_3();
   Alarm_4();

   IF Alarm_1.q THEN
      DebugMsg(message:="Alarm 1");
   END_IF;
   IF Alarm_2.q THEN
      DebugMsg(message:="Alarm 2");
   END_IF;
   IF Alarm_3.q THEN
      DebugMsg(message:="Alarm 3");
   END_IF;
   IF Alarm_4.q THEN
      DebugMsg(message:="Alarm 4");
   END_IF;
END;

END_PROGRAM;
```

4.1.13.10 clockSetWakeup (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.36.00

This function is used to configure the system to wake from suspend at a specific time.
It can e.g. be used to wake at midnight so the device can send a report.
The time can both be supplied as a linsec with an absolute time to wake at and the individual time components such as hour and minute.
If the time components are provided, the wake time will be set to an absolute time when the function is called, so it should be called just before calling pmSuspend.

When using time components, at least one part must be specified. All the parts that are larger than the specified parts will be ignored while all the parts that are smaller will be set to the current time.
The time will always be adjusted to be in the future, if e.g. a day is chosen that is smaller that the current day, it will trigger in the next month instead.
Examples:
Current time: 2019-04-02 15:44
Setting month = 5: 2019-05-02 15:44
Setting hour = 0: 2019-04-03 00:44
Setting minute = 30: 2019-04-02 16:30

Note that this can not be used while also using the time parameter on pmSuspend.

pmSuspend return codes for this wake-up source:

Error	Type	ID	Value	Description
True	1	1	-10	The time parameter on pmSuspend is in use.
True	1	1	-11	The wake up time is too soon.
False	1	1	1	Wake-up at the specific time.

Input:

Enable : BOOL (default: TRUE)
Set to true to enable the wake-up, set to false to disable it.

linsec : DINT (default: -1)
The linsec to wake at. If set to -1 it will not be used.

month : SINT (default: -1)
The month part of the wake-up time. If set to -1, the month will be ignored.

day : SINT (default: -1)
The day part of the wake-up time. If set to -1, the day will be ignored.

hour : SINT (default: -1)
The hour part of the wake-up time. If set to -1, the hour will be ignored.

minute : SINT (default: -1)
The minute part of the wake-up time. If set to -1, the minute will be ignored.

second: SINT (default: -1)
The second part of the wake-up time. If set to -1, the second will be ignored.

Returns:

1 - The system has been configured to wake at the specified time.
0 - This function is not supported.
-1 - The wake-up time could not be created.

Declaration:

```
FUNCTION ALIGN clockSetWakeup:INT;
VAR_INPUT
   enable: BOOL := TRUE;
   linsec: DINT := -1;
   month : SINT := -1;
   day   : SINT := -1;
   hour  : SINT := -1;
```

```
   minute: SINT := -1;
   second: SINT := -1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Enable wake-up when minute and second are 0, i.e. when the hour starts.
   rc := clockSetWakeup(minute := 0, second := 0);
   DebugFmt(message:="clockSetWakeup: \4", v1 := rc);
   ...
   // Enable wake-up at 2019-12-24 12:00:30
   rc := clockSetWakeup(linsec := clockTimeToLinsec(year := 2019, month := 12,
day := 24, hour:=12, minute:=0, second:=30));
   DebugFmt(message:="clockSetWakeup: \4", v1 := rc);
   ...
END;

END_PROGRAM;
```

4.1.14 SFL: Semaphore functions (multithreading)

4.1.14.1 sem: Semaphore functions

A SEMAPHORE is a protected variable (or *abstract data type*) and constitutes the classic method for restricting access to shared resources (e.g. storage) in a multi-processing environment.

Semaphores can only be accessed by using the following fundamental operations:

```
Wait(Semaphore s)
{
        while (value <= 0);        /* wait until value>0 */
        value = value-1;                    /* must not be interrupted */
}

Signal(Semaphore s)
{
        value = value+1;           /* must not be interrupted */
}

Init(Semaphore s, Integer v)
{
        value = v;
}
```

The *value* of a semaphore is the number of units of the resource which are free (if there is only one resource, a "binary semaphore" with values 0 or 1 is used). The **Wait** operation waits until a resource is available, whereupon it immediately claims one. **Signal** is the inverse; it simply makes a resource available again after the thread has finished using it. **Init** is only used to initialize the semaphore before any requests are made. The **Wait** and **Signal** operations must be indivisible, which means that each of the operations may not be executed multiple times concurrently.

As the semaphore mechanism is a very general mechanism, it can be used for a broad range of synchronization purposes. It can be used to implement critical sections, like the Mutex, but a Mutex is preferred as it is easier, safer and more efficient to use.

Also see the section on thread synchronization for more information.

The following semaphore functions exists:

- semInit — Initializes a semaphore.
- semDestroy — Destroys a semaphore.
- semWait — Waits on a semaphore.
- semSignal — Signals a semaphore.
- semValue — Queries a semaphore for its value.

4.1.14.2 semInit (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

The semInit() function will initialize a SEMAPHORE variable with the specified initial value. The semInit() function must be called before any other operations can be performed on the semaphore.

Input:
initval : INT
The initial value for the semaphore.

Returns: SEMAPHORE
The initialized semaphore.

Declaration:
```
FUNCTION semInit : SEMAPHORE;
VAR_INPUT
   initval : INT := 1;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

VAR
   sem : SEMAPHORE;
END_VAR;

PROGRAM test;
   ...
   // Initialize Semaphore
   sem := semInit(initval:=3);
   IF semValue(sem:=sem) = -1 THEN DebugMsg(message:="sem failed to init!");
END_IF;
   ...
BEGIN
   ...
   // Wait until resource is free or timeout
   IF semWait(sem:=sem,timeout:=1000) = 0 THEN
      // Only do these actions if we have access to the resource
      ...
      // Free the resource after use
      semSignal(sem:=sem);
   END_IF;
   ...
END;

END_PROGRAM;
```

4.1.14.3 semDestroy (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

The semDestory() function will destroy a SEMAPHORE variable.
After the call to semDestroy(), the semaphore variable cannot be used in further operations.

Input:
sem : SEMAPHORE
The semaphore to destroy.

Returns: INT

0 Semaphore is destroyed.
1 Semaphore is not initialized.
2 Semaphore is busy.

Declaration:

```
FUNCTION semDestroy : INT;
VAR_INPUT
   sem : SEMAPHORE;
END_VAR;
```

4.1.14.4 semWait (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

The semWait() function will perform a traditional WAIT operation on the specified semaphore.
The semWait() mechanism will:

- When the semaphore value>0, it will decrement the semaphore value and the calling thread will continue to execute.
- When the semaphore value=0, it will block the calling thread on a FIFO queue waiting for another thread to call semSignal().

The semaphore mechanism can be used for a big range of different synchronization tasks. It can also be used to implement critical sections, like the Mutex mechanism, but for that purpose the Mutex is better suited.
The semWait() operation also offers a timeout facility to reduce the time waiting for the semaphore.

Also see the section on thread synchronization for more information.

Input:

sem : SEMAPHORE
The semaphore to wait for.

timeout : INT *(default: -1 forever)*
The time, in ms., to wait for the semaphore to be signaled.
Use **-1** to wait forever. This is the default.

Returns: INT

0 Success.
1 Semaphore is not initialized.
2 Timeout occurred. (operation aborted).

Declaration:

```
FUNCTION semWait : INT;
VAR_INPUT
   sem     : SEMAPHORE;
   timeout : INT := -1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
```

```
VAR
   sem : SEMAPHORE;
END_VAR;

PROGRAM test;
   ...
   // Initialize Semaphore
   sem := semInit(initval:=3);
   IF semValue(sem:=sem) = -1 THEN DebugMsg(message:="sem failed to init!");
END_IF;
   ...
BEGIN
   ...
   // Wait until resource is free or timeout
   IF semWait(sem:=sem,timeout:=1000) = 0 THEN
      // Only do these actions if we have access to the resource
      ...
      // Free the resource after use
      semSignal(sem:=sem);
   END_IF;
   ...
END;

END_PROGRAM;
```

4.1.14.5 semSignal (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

The semSignal() function will perform a traditional SIGNAL operation on the specified semaphore. The semSignal() mechanism will:

- If there are threads blocked in the semWait() waiting queue for the semaphore, the first thread will be released to execute.
- Otherwise the semaphore value will be incremented by 1.

Also see the function semWait() and the section on thread synchronization for more information.

Input:

sem : SEMAPHORE
The semaphore to Signal.

Returns: INT

0 Semaphore is signaled.
1 Semaphore is not initialized.

Declaration:

```
FUNCTION semSignal : INT;
VAR_INPUT
   sem : SEMAPHORE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   sem : SEMAPHORE;
```

```
END_VAR;

PROGRAM test;
   ...
   // Initialize Semaphore
   sem := semInit(initval:=3);
   IF semValue(sem:=sem) = -1 THEN DebugMsg(message:="sem failed to init!");
END_IF;
   ...
BEGIN
   ...
   // Wait until resource is free or timeout
   IF semWait(sem:=sem,timeout:=1000) = 0 THEN
      // Only do these actions if we have access to the resource
      ...
      // Free the resource after use
      semSignal(sem:=sem);
   END_IF;
   ...
END;

END_PROGRAM;
```

4.1.14.6 semValue (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

The semValue() function will return the current value of the specified semaphore.

Input:

sem : SEMAPHORE
The semaphore to query.

Returns: INT

-1 Semaphore not initialized.
>=0 Value of the semaphore.

Declaration:

```
FUNCTION semValue : INT;
VAR_INPUT
   sem : SEMAPHORE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   sem : SEMAPHORE;
END_VAR;

PROGRAM test;
   ...
   // Initialize Semaphore
   sem := semInit(initval:=3);
   IF semValue(sem:=sem) = -1 THEN DebugMsg(message:="sem failed to init!");
END_IF;
   ...
```

```
BEGIN
   ...
   // Wait until resource is free or timeout
   IF semWait(sem:=sem,timeout:=1000) = 0 THEN
      // Only do these actions if we have access to the resource
      ...
      // Free the resource after use
      semSignal(sem:=sem);
   END_IF;
   ...
END;

END_PROGRAM;
```

4.1.15 SFL: String handling functions

4.1.15.1 SFL: String handling functions

The string handling functions are used when you need to manipulate strings in a program. A number of functions exists, and they are each explained individually on the following pages.

String template
The string template functions are helpful when handling large strings that are used multiple times but must be slightly different every time, such as strings that contain sensor data to send to an external device. See strTemplateCreate for more details.

- strFormat — Formats a string.
- strGetValues — Extracts numerical values from a string.
- strGetStrings — Extracts string values from a string.
- strToInt — Converts a string to a number (INT).
- hexToInt — Converts a hex string to a number (INT).
- strToDint — Converts a string to a number (DINT).
- hexToDint — Converts a hex string to a number (DINT).
- strToSint — Converts a string to a number (SINT).
- hexToSint — Converts a hex string to a number (SINT).
- strCompare — Compare two strings.
- strConcat — Concatenate two strings.
- strLeft — Extracts the left part of a string.
- strRight — Extracts the right part of a string.
- strMid — Extracts part of a string.
- strLen — Finds the length (number of characters) of a string.
- strLookup — Find a string in a list of strings.
- strFind — Checks if a string is contained within another string.
- strToken — Extracts a string from a delimited string.
- strRemoveSpaces — Removes leading/trailing spaces in a string.
- sintToStr — Converts a number (SINT) to a string.
- sintToHex — Converts a number (SINT) to a hex string.
- intToStr — Converts a number (INT) to a string.
- intToHex — Converts a number (INT) to a hex string.
- dintToStr — Converts a number (DINT) to a string.
- dintToHex — Converts a number (DINT) to a hex string.
- strToMemory — Copies contents of a string to memory.
- strFromMemory — Copies contents of memory to a string.
- strEncode64 — Encodes a string with base 64.
- strDecode64 — Decodes a base 64 encoded string.
- strMemoryUsage — Get information about memory usage of dynamic strings.
- strTemplateCreate — Create a string template to perform fast replacement of sub-strings.
- strTemplateFree — Release the string template,
- strTemplateVarInfo — Get information about the variables found in the string template.
- strTemplateSetVar — Update the value of a variable in the string template.
- strTemplateGenerateString — Generate a string from the string template.

For floating-point string conversion, please refer to the Floating-Point Math section.

4.1.15.2 strFormat (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

---

strFormat is used to format a string. By supplying the function with a format string, a string, and up to 4 numbers, it is possible to format a string.
The format string can be specified such that up to 4 numbers can be inserted in a string. Please look at the examples below.

Please note that the 4th value is a DINT type as this allows you to convert large numbers into a string.

Note that the returned string is a dynamic string.

Input:

format : STRING
This format string can contain up to 4 parameter references written as
\1, \2, \3 and \4. Each of these will insert the value in v1, v2, v3, and v4.
Please note that a specific number can be referenced more than one time in the format string.

v1 : INT
Value for {1} in the format string.

v2 : INT
Value for {2} in the format string.

v3 : INT
Value for {3} in the format string.

v4 : DINT
Value for {4} in the format string.

Returns: STRING

A string containing the resulting string from the formatting

Declaration:

```
FUNCTION strFormat : STRING;
VAR_INPUT
   format    : STRING;
   v1,v2,v3  : INT;
   v4        : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   number1 : INT;
   number2 : INT;
   str     : STRING;
END_VAR;

PROGRAM test;

BEGIN
```

```
    ...
    str := strFormat(format:="The temperature at sensor \1 is \2 degrees",
v1:=12, v2:=32);
    // str will now contain: "The temperature at sensor 12 is 32 degrees"
    ...
    str := strFormat(format:="\1", v1 := 4711);
    // str will now contain: "4711"
    ...
END;

END_PROGRAM;
```

4.1.15.3 strGetValues (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

---

strGetValues will try to extract up to 4 values from a string. It uses the same syntax for the format string as strFormat. strGetValues is not case-sensitive.

Input:

str : STRING
String that strGetValues will extract values from.

format : STRING
The format string can contain up to 4 parameter references written as
\1, \2, \3 and \4. Each of these will extract the value at that position and place it in v1, v2, v3, or v4.
Please note that a specific number can only be referenced one time in the format string.

Please note that the 4th value is a DINT type, this allows you to convert large numbers.

Output:

match: BOOL
This will be TRUE if a match was found, and the number of values was extracted successfully

v1: INT
The extracted value at {1} from str

v2: INT
The extracted value at {2} from str

v3: INT
The extracted value at {3} from str

v4 : DINT
The extracted value at {4} from str

Declaration:

```
FUNCTION_BLOCK strGetValues;
VAR_INPUT
    str    : STRING;
    format : STRING;
END_VAR;
VAR_OUTPUT
    match  : BOOL;
    v1     : INT;
    v2     : INT;
    v3     : INT;
```

```
   v4     : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   extract     : strGetValues;
   inputstring : STRING;
END_VAR;

PROGRAM test;

BEGIN
   ...
   inputstring:="The temperature at sensor 12 is 32 degrees";
   extract(str:=inputstring, format:="The temperature at sensor \1 is \2
degrees");
   IF extract.match THEN
      // extract.v1 contains 12, extract.v2 contains 32
      ...
   END_IF;
END;

END_PROGRAM;
```

4.1.15.4 strGetStrings (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

strGetStrings is similar to strGetValues. It will try to extract up to 4 strings from a string. It uses the same syntax for the format string as strFormat. strGetStrings is not case sensitive !

Input:

str : STRING
String that strGetValues will extract values from

format : STRING
Format string, can contain up to 4 parameter references written as
\1, \2, \3, and \4. Each of these will extract the value at that position and place it in v1, v2, v3, or v4.
Please note that a specific number can only be referenced one time in the format string.

Output:

match: BOOL
This will be TRUE if a match was found and the number of strings was extracted successfully.

v1: STRING
The extracted string at {1} from str.

v2: STRING
The extracted string at {2} from str.

v3: STRING
The extracted string at {3} from str.

v4: STRING
The extracted string at {4} from str.

Declaration:

```
FUNCTION_BLOCK strGetStrings;
VAR_INPUT
   str    : STRING;
   format : STRING;
END_VAR;
VAR_OUTPUT
   match  : BOOL;
   v1     : STRING;
   v2     : STRING;
   v3     : STRING;
   v4     : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   extract     : strGetStrings;
   inputstring : STRING;
END_VAR;

PROGRAM test;

BEGIN
   ...
   inputstring:="The north valve is closed";
   extract(str:=inputstring, format:="The \1 valve is \2");
   IF extract.match THEN
      // extract.v1 contains "north", extract.v2 contains "closed"
      ...
   END_IF;
END;

END_PROGRAM;
```

4.1.15.5 strToInt (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

---

strToInt will convert a string to a number (INT). If the conversion is not possible, strToInt will return 0.

Input:

str : STRING
String to convert.

Returns: INT

The value of the string.

Declaration:

```
FUNCTION strToInt : INT;
VAR_INPUT
   str : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   result : INT;
END_VAR;

PROGRAM test;

BEGIN
   ...
   result := strToInt(str:="4711");
   // result will contain 4711 after the call
   ...
END;

END_PROGRAM;
```

4.1.15.6 hexToInt (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 5.08 / 1.52.00

hexToInt will convert a hex string to a number (INT). If the conversion is not possible, hexToInt will return 0.

Input:

hex : STRING
String to convert.

Returns: INT

The value of the string.

Declaration:

```
FUNCTION hexToInt : INT;
VAR_INPUT
   hex : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   result : INT;
END_VAR;

PROGRAM test;

BEGIN
   ...
   result := hexToInt(hex:="100");
   // result will contain 256 after the call
   ...
END;

END_PROGRAM;
```

4.1.15.7 strToDint (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

strToDint will convert a string to a number (DINT). If the conversion is not possible, strToDint will return 0.

Input:
str : STRING
String to convert.

Returns: **DINT**
The value of the string.

Declaration:
```
FUNCTION strToDint : DINT;
VAR_INPUT
   str : STRING;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

VAR
   result : DINT;
END_VAR;

PROGRAM test;

BEGIN
   ...
   result := strToDint(str:="4711");
   // result will contain 4711 after the call
   ...
END;

END_PROGRAM;
```

4.1.15.8 hexToDint (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 5.08 / 1.52.00

hexToDint will convert a hex string to a number (DINT). If the conversion is not possible, hexToDint will return 0.

Input:
hex : STRING
String to convert.

Returns: **DINT**
The value of the string.

Declaration:

```
FUNCTION hexToDint : DINT;
VAR_INPUT
   hex : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   result : DINT;
END_VAR;

PROGRAM test;

BEGIN
   ...
   result := hexToDint(hex:="10000");
   // result will contain 65536 after the call
   ...
END;

END_PROGRAM;
```

4.1.15.9 strToSint (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

---

strToSint will convert a string to a number (SINT). If the conversion is not possible, strToSint will return 0.

Input:

str : STRING
String to convert.

Returns: SINT

The value of the string.

Declaration:

```
FUNCTION strToSint : SINT;
VAR_INPUT
   str : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   result : SINT;
END_VAR;

PROGRAM test;

BEGIN
   ...
   result := strToSint(str:="47");
   // result will contain 47 after the call
```

```
   ...
END;

END_PROGRAM;
```

4.1.15.10 hexToSint (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 5.08 / 1.52.00

hexToSint will convert a string to a number (SINT). If the conversion is not possible, hexToSint will return 0.

Input:

hex : STRING
String to convert.

Returns: SINT

The value of the string.

Declaration:

```
FUNCTION hexToSint : SINT;
VAR_INPUT
   hex : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   result : SINT;
END_VAR;

PROGRAM test;

BEGIN
   ...
   result := hexToSint(hex:="80");
   // result will contain -128 after the call
   ...
END;

END_PROGRAM;
```

4.1.15.11 strCompare (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

strCompare compares two strings and returns a number that specifies which one of the strings is greater than the other or if they are equal. It is possible to test strings without case sensitivity - that is to say that "abc" will be the same as "ABC". If no case-checking is done (see input parameter CheckCase).

Input:

str1 : STRING
String 1 for the compare function.

str2 : STRING
String 2 for the compare function.

CheckCase : BOOL *Default: FALSE*
TRUE if the search should be case-sensitive.

Returns: INT(-1,0,1)

-1 if str1 < str2
0 if str1 = str2
1 if str1 > str2

Declaration:

```
FUNCTION strCompare : INT;
VAR_INPUT
   str1, str2 : STRING;
   CheckCase  : BOOL := FALSE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   compareresult : INT;
END_VAR;

PROGRAM test;

BEGIN
   ...
   compareresult := strCompare(str1:="hello", str2:="HeLLo");
   // Compareresult will be 0 because the two strings a equal (because of
CheckCase is default FALSE)
   ...
   compareresult := strCompare(str1:="hello", str2:="HeLLo", CheckCase:=TRUE);
   // Compareresult will be 1 because str1 < str2
   ...
   compareresult := strCompare(str1:="ABC", str2:="def");
   // Compareresult will be -1 because str2 > str1, and CheckCase is default
equal to FALSE
   ...
END;

END_PROGRAM;
```

4.1.15.12 strConcat (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

---

strConcat concatenates ("adds" together) two strings and returns a string with the result in it. Also see the '+' operator for an alternative way of concatenating strings.

Note that the returned string is a dynamic string.

Input:

str1 : STRING
Left string.

str2 : STRING
Right string.

Returns: STRING

The resulting string from concatenating str1 and str2.

Declaration:

```
FUNCTION strConcat : STRING;
VAR_INPUT
   str1, str2 : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   result : STRING;
END_VAR;

PROGRAM test;

BEGIN
   ...
   result := strConcat(str1:="hello ", str2:="world");
   // result will contain "Hello world" after the call
   ...
END;

END_PROGRAM;
```

4.1.15.13 strLeft (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

---

strLeft will return a specified number of characters from a string. It starts from the left and moves right.

Note that the returned string is a dynamic string.

Input:

str : STRING
String that is to be extracted from.

length : INT
Number of characters to be returned.

Returns: STRING

The extracted string.

Declaration:

```
FUNCTION strLeft : STRING;
VAR_INPUT
   str    : STRING;
   length : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   result : STRING;
END_VAR;

PROGRAM test;

BEGIN
   ...
   result := strLeft(str:="hello world", length:=3);
   // result will contain "Hel" after the call
   ...
END;

END_PROGRAM;
```

4.1.15.14 strRight (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

---

strRight will return a specified number of characters from a string. It starts from the right and moves left.
Note that the returned string is a dynamic string.

Input:

str : STRING
String that is to be extracted from.

length : INT
Number of characters to be returned.

Returns: STRING

The extracted string.

Declaration:

```
FUNCTION strRight : STRING;
VAR_INPUT
   str    : STRING;
   length : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   result : STRING;
END_VAR;
```

```
PROGRAM test;

BEGIN
   ...
   result := strRight(str:="hello world", length:=3);
   // result will contain "rld" after the call
   ...
END;

END_PROGRAM;
```

4.1.15.15 strMid (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

---

strMid will return a specified sub-string from a string.

Note that the returned string is a dynamic string.

Input:

str : STRING
String that is to be extracted from.

start : INT
Start position from the left for extracted string.

length : INT ***Default 999***
Number of characters to return from the "start" position. If not assigned, then the rest of 'str' is returned.

Returns: STRING

The extracted string.

Declaration:

```
FUNCTION strMid : STRING;
VAR_INPUT
   str    : STRING;
   start  : INT;
   length : INT := 999;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   result : STRING;
END_VAR;

PROGRAM test;

BEGIN
   ...
   result := strMid(str:="hello world", start:=4, length:=2);
   // result will contain "lo" after the call
   ...
END;
```

```
END_PROGRAM;
```

4.1.15.16 strLen (Function)

```
Architecture:       X32 / NX32 / NX32L
Device support:     ALL
Firmware version:   1.00 / 1.00.00
```

strLen will return the length (number of characters) of a string.

Input:

str : STRING
String that is to be extracted from.

Returns: INT

The number of characters in the str string.

Declaration:

```
FUNCTION strLen : INT;
VAR_INPUT
   str : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   result : INT;
END_VAR;

PROGRAM test;

BEGIN
   ...
   result := strLen(str:="hello world");
   // result will contain 11 after the call
   ...
END;

END_PROGRAM;
```

4.1.15.17 strLookup (Functionblock)

```
Architecture:       X32 / NX32 / NX32L
Device support:     ALL
Firmware version:   1.00 / 1.00.00
```

strLookup will compare a string with up to 4 different strings and report if there is a match. The function is not case sensitive - so "Fox" and "fox" will be equal.

Input:

str : STRING
String that the function will try to match with str1, str2, str3, or str4.

str1 : STRING
String 1 that the function will compare with str.

str2 : STRING
String 2 that the function will compare with str.

str3 : STRING
String 3 that the function will compare with str.

str4 : STRING
String 4 that the function will compare with str.

Output:

match : INT (0..4)

0	if str matched none of str1, str2, str3, or str4.
1	if str is identical to str1.
2	if str is identical to str2.
3	if str is identical to str3.
4	if str is identical to str4.

Declaration:

```
FUNCTION_BLOCK strLookup;
VAR_INPUT
   str   : STRING;
   str1  : STRING;
   str2  : STRING;
   str3  : STRING;
   str4  : STRING;
END_VAR;
VAR_OUTPUT
   match : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   result : INT;
   search : strLookup;
END_VAR;

PROGRAM test;

BEGIN
   ...
   search(str:="fox", str1:="The", str2:="Quick", str3:="Brown", str4:="Fox");
   result := search.match;
   // result will contain 4 as the str matched the contents of str4 (Fox)
   ...
END;

END_PROGRAM;
```

4.1.15.18 strFind (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 (4.60 : for the 'start' parameter) / 1.00.00

---

strFind will search for a string within another string. It will return the start position of "str2" in "str1" if str2 was found in str1 - otherwise it will return 0. It is possible to search for the string without case sensitivity - that is to say that "abc" will be the same as "ABC" if no case checking is done (see input parameter CheckCase).

Please note that if either str1 or str2 is empty, this function will return 0 (not found).

Input:

str1 : STRING
String to search in.

str2 : STRING
String to search for.

start : int *Default:1*
Position in "str1" where the search for "str2" starts.
Default is to start at the beginning of 'str1' which equals 'start=1'.

CheckCase : BOOL *Default: FALSE*
TRUE if the search should be case-sensitive.

Returns: INT

The start position of "str2" in "str1", 0 if "str2" was not found.
The first character of "str1" is defined as position 1.

Declaration:

```
FUNCTION strFind : INT;
VAR_INPUT
   str1      : STRING;
   str2      : STRING;
   start     : INT := 1;
   CheckCase : BOOL := FALSE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   result : INT;
END_VAR;

PROGRAM test;

BEGIN
   ...
   result := strFind(str1:="hello world", str2:="World");
   // result will contain 7 after the call
   result := strFind(str1:="Hard and harder", str2:="hard", start:=5);
   // result will contain 10 after the call
   ...
END;

END_PROGRAM;
```

4.1.15.19 strToken (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

---

strToken will extract a sub-string within another string that contains a number of sub-strings - each separated by a "delimiter". The input number is the sub-string number to return.
Any leading or trailing spaces will be ignored.

Input:

str : STRING
String to search in.

delimiter : STRING
The delimiting string that separates the sub-strings in "str".

index : INT
The index number of the sub-string that should be returned.

Returns: STRING

The extracted sub-string from "str".

Declaration:

```
FUNCTION strToken : STRING;
VAR_INPUT
   str       : STRING;
   delimiter : STRING;
   index     : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   result : STRING;
END_VAR;

PROGRAM test;

BEGIN
   ...
   result := strToken(str:="4711,4712,4713,4714", delimiter:=",", index:=2);
   // result will contain "4712" after the call
   ...
END;

END_PROGRAM;
```

4.1.15.20 strRemoveSpaces (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

---

strRemoveSpaces will remove leading and trailing spaces in a string. It is optional if the function should remove the leading, trailing, or both types of spaces in a string.

Note that the returned string is a dynamic string.

Input:

str : STRING
String that is to be extracted from.

leading : BOOL *Default TRUE*
TRUE if leading spaces should be removed.

trailing : BOOL *Default TRUE*
TRUE if trailing spaces should be removed.

Returns: STRING

The extracted string.

Declaration:

```
FUNCTION strRemoveSpaces : STRING;
VAR_INPUT
   str      : STRING;
   leading  : BOOL := TRUE;
   trailing : BOOL := TRUE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   result : STRING;
END_VAR;

PROGRAM test;

BEGIN
   ...
   result := strRemoveSpaces(str:="   hello world  ", trailing:=FALSE);
   // result will contain "hello world  " after the call, as only leading
spaces are removed
   ...
END;

END_PROGRAM;
```

4.1.15.21 sintToStr (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

sintToStr will convert a number (SINT) to a string.

Input:

v : SINT
Number to convert.

Returns: STRING

A string representing the value of v.

Declaration:

```
FUNCTION sintToStr : STRING;
VAR_INPUT
   v : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   result : STRING;
END_VAR;

PROGRAM test;

BEGIN
   ...
   result := sintToStr(v:=47);
   // result will contain "47" after the call
   ...
END;

END_PROGRAM;
```

4.1.15.22 sintToHex (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 5.08 / 1.52.00

sintToHex will convert a number (SINT) to a string.

Input:

v : SINT
Number to convert.

Returns: STRING

A string containing the hexadecimal representation of v.

Declaration:

```
FUNCTION sintToHex : STRING;
VAR_INPUT
   v : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   result : STRING;
END_VAR;

PROGRAM test;

BEGIN
   ...
```

```
   result := sintToHex(v:=-128);
   // result will contain "80" after the call
   ...
END;

END_PROGRAM;
```

4.1.15.23 intToStr (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

---

intToStr will convert a number (INT) to a string.

Input:

v : INT
Number to convert.

Returns: STRING

A string representing the value of v.

Declaration:

```
FUNCTION intToStr : STRING;
VAR_INPUT
   v : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   result : STRING;
END_VAR;

PROGRAM test;

BEGIN
   ...
   result := intToStr(v:=4711);
   // result will contain "4711" after the call
   ...
END;

END_PROGRAM;
```

4.1.15.24 intToHex (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 5.08 / 1.52.00

---

intToHex will convert a number (INT) to a hex-string.

Input:

v : INT
Number to convert.

Returns: STRING
A string containing the hexadecimal representation of v.

Declaration:
```
FUNCTION intToHex : STRING;
VAR_INPUT
   v : INT;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

VAR
   result : STRING;
END_VAR;

PROGRAM test;

BEGIN
   ...
   result := intToHex(v:=256);
   // result will contain "0100" after the call
   ...
END;

END_PROGRAM;
```

4.1.15.25 dintToStr (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

---

dintToStr will convert a number (DINT) to a string.

Input:
v : DINT
Number to convert.

Returns: STRING
A string representing the value of v.

Declaration:
```
FUNCTION dintToStr : STRING;
VAR_INPUT
   v : DINT;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

VAR
   result : STRING;
END_VAR;

PROGRAM test;
```

```
BEGIN
   ...
   result := dintToStr(v:=12345678);
   // result will contain "12345678" after the call
   ...
END;

END_PROGRAM;
```

4.1.15.26 dintToHex (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 5.08 / 1.52.00

---

dintToHex will convert a number (DINT) to a hex-string.

Input:

v : DINT
Number to convert.

Returns: STRING

A string containing the hexadecimal representation of v.

Declaration:

```
FUNCTION dintToHex : STRING;
VAR_INPUT
   v : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   result : STRING;
END_VAR;

PROGRAM test;

BEGIN
   ...
   result := dintToHex(v:=20000000);
   // result will contain "01312D00" after the call
   ...
END;

END_PROGRAM;
```

4.1.15.27 strToMemory (function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

---

strToMemory will copy the contents of a string to memory.

Input:

dst : PTR
Pointer to the memory where the string should be copied to.

str : STRING
The string that is to be copied.

len : INT
Number of bytes to be copied to memory.

Returns:
None

Declaration:
```
FUNCTION strToMemory;
VAR_INPUT
   dst : PTR;
   str : STRING;
   len : INT;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

VAR
   data : ARRAY[0..100] OF SINT;
   s    : STRING;
END_VAR;

PROGRAM test;

BEGIN
   ...
   s:="hello world";
   strToMemory(dst:=ADDR(data), str:=s, len:=strlen(str:=s));
   // The array 'data' will now contain the contents of the string 's'
   ...
END;

END_PROGRAM;
```

4.1.15.28 strFromMemory (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

---

strFromMemory will copy the contents of a memory area to string.

Note that the returned string is a dynamic string.

Input:
src : PTR
Pointer to the memory where the string should be copied from.

len : INT
Number of bytes to be copied from memory.

Returns: STRING

The string copied from the memory area.

Declaration:

```
FUNCTION strFromMemory : STRING;
VAR_INPUT
   src : PTR;
   len : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   data : ARRAY[0..100] OF SINT;
   s    : STRING;
END_VAR;

PROGRAM test;

BEGIN
   ...
   s:=strFromMemory(src:=ADDR(data), len:=SIZEOF(data));
   // The string 's' will now contain the data (string) from the array 'data'
   ...
END;

END_PROGRAM;
```

4.1.15.29 strEncode64 (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

strEncode64 will encode a string by using base 64.

Note that the returned string is a dynamic string.

Input:

str : STRING
The string to encode.

Returns: STRING

The string encoded in base 64.

Declaration:

```
FUNCTION strEncode64 : STRING;
VAR_INPUT
   str : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   str_src : STRING;
   str_enc : STRING;
   str_dec : STRING;
```

```
END_VAR;

PROGRAM test;

BEGIN
   ...
   str_src := "Aladdin:open sesame";
   str_enc := strEncode64(str:=str_src);
   str_dec := strDecode64(str:=str_enc);
   // srt_enc = "QWxhZGRpbjvcGVuIHNlc2FtZQ=="
   // str_dec = "Aladdin:open sesame"
   ...
END;

END_PROGRAM;
```

4.1.15.30 strDecode64 (Function)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

---

strDecode64 will decode a string that is encoded in base64.

Note that the returned string is a dynamic string.

Input:

str : STRING
A base64-encoded string.

Returns: STRING

The decoded string.

Declaration:

```
FUNCTION strDecode64 : STRING;
VAR_INPUT
   str : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   str_src : STRING;
   str_enc : STRING;
   str_dec : STRING;
END_VAR;

PROGRAM test;

BEGIN
   ...
   str_src := "Aladdin:open sesame";
   str_enc := strEncode64(str:=str_src);
   str_dec := strDecode64(str:=str_enc);
   // srt_enc = "QWxhZGRpbjvcGVuIHNlc2FtZQ=="
   // str_dec = "Aladdin:open sesame"
   ...
END;
```

```
END_PROGRAM;
```

4.1.15.31 strMemoryUsage (Function)

```
Architecture:       X32 / NX32 / NX32L
Device support:     ALL
Firmware version:   3.12 / 1.00.00
```

strMemoryUsage returns information on the dynamic string memory allocated by the application. The currently allocated number of dynamic strings and the memory used is returned by this function. Additionally the peak for both values are returned. The strMemoryUsage function can assist in locating string allocation faults.

Please also see the STRING datatype.

Input:

None.

Output:

cnt_cur : DINT
The current number of dynamic strings allocated.

cnt_peak : DINT
The maximum number of simultaneously allocated dynamic strings since device start-up.

mem_cur : DINT
The number of bytes currently used by dynamic strings.

mem_peak : DINT
The maximum number of bytes used by dynamic strings since device start-up.

Returns:

None.

Declaration:

```
FUNCTION strMemoryUsage;
VAR_INPUT
   cnt_cur  : ACCESS DINT;
   cnt_peak : ACCESS DINT;
   mem_cur  : ACCESS DINT;
   mem_peak : ACCESS DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   cnt_cur  : DINT;
   cnt_peak : DINT;
   mem_cur  : DINT;
   mem_peak : DINT;
END_VAR;

PROGRAM test;

BEGIN
```

```
    ...
    strMemoryUsage(cnt_cur := cnt_cur, cnt_peak := cnt_peak, mem_cur :=
mem_cur, mem_peak := mem_peak);
    debugFmt(message := "String count, cur = \4", v4 := cnt_cur);
    debugFmt(message := "String count, peak = \4", v4 := cnt_peak);
    debugFmt(message := "String memory, cur = \4", v4 := mem_cur);
    debugFmt(message := "String memory, peak = \4", v4 := mem_peak);
    ...
END;

END_PROGRAM;
```

4.1.15.32 strTemplateCreate (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

---

strTemplateCreate parses the given string and generates a string template based on it.
String templates can be used to quickly replace specific sub-strings in a large string with another string.

It detects names surrounded by specific characters, which can then be replaced when generating new strings.
Using the default sov of `"${"` and eov of `"}"`, it matches e.g. `"${variable}"`.

The string template functions are only available in NX32L compilation mode.

String templates use the following steps:
1. Call strTemplateCreate with the source string to generate the template
2. Optional: Use strTemplateVarInfo to examine the found variables.
3. Use strTemplateSetVar to set the value to use for the variables.
4. Call strTemplateGenerateString to generate a string with the new variables.
5. Repeat step 3 and 4 as many times as needed.
6. Call strTemplateFree to release the string template.

Note: To avoid having to escape special characters in the source string, the string can be defined as a raw string, see STRING_BEGIN for more details.

Input:

str : STRING
The string to use as template.

sov : STRING ***(Default "$${")***
The start-of-variable string.

eov : STRING ***(Default "}")***
The end-of-variable string.

Output:

st: SYSHANDLE
A handle to the string template.

Returns: INT

1	- Success
0	- Function is not supported.
-1	- Failed to create handle.
-2	- Failed to parse string.

-3 - Not enough resources to parse string.
-4 - sov or eov is invalid.

Declaration:

```
FUNCTION strTemplateCreate;
VAR_INPUT
   st       : ACCESS SYSHANDLE;
   str      : STRING;
   sov      : STRING := "$${";
   eov      : STRING := "}";
END_VAR;
```

Example:

```
INCLUDE rtcu.inc


PROGRAM test;
VAR
   st       : SYSHANDLE;
   rc       : INT;
   str_in   : STRING;
   str_out  : STRING;
END_VAR;

   str_in := "Time since reset: $${time}, Temperature: $${temp}, Values:
$"$${time}, $${temp}, $${value}$"";

   // Parse string and create template.
   rc := strTemplateCreate(st:=st, str:=str_in, sov := "$${", eov := "}");
   DebugFmt(message:="strTemplateCreate: \1", v1:=rc);

   // Set the value for the fixed variable
   rc := strTemplateSetVar(st:=st, name:="value", value := "value");
   DebugFmt(message:="strTemplateSetVar: \1", v1:=rc);

BEGIN
   // Set the value for the dynamic variables
   rc := strTemplateSetVar(st:=st, name:="time",
value:=dintToStr(v:=boardTimeSinceReset()/1000));
   DebugFmt(message:="strTemplateSetVar: \1", v1:=rc);

   rc := strTemplateSetVar(st:=st, name:="temp",
value:=floatToStr(v:=FLOAT(boardTemperature())/100.0));
   DebugFmt(message:="strTemplateSetVar: \1", v1:=rc);

   // Generate string
   rc := strTemplateGenerateString(st:=st, str:=str_out);
   DebugFmt(message:="strTemplateGenerateString: \1", v1:=rc);

   // Print string to device output
   DebugMsg(message:=str_out);

   Sleep(delay:=10000);
END;

END_PROGRAM;
```

4.1.15.33 strTemplateFree (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

---

strTemplateFree releases the string template.

Input:

st: SYSHANDLE
A handle to the string template to release.

Returns: INT

1 - Success
0 - Function is not supported.
-1 - Invalid handle.

Declaration:

```
FUNCTION strTemplateFree;
VAR_INPUT
   st       : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc


PROGRAM test;
VAR
   st       : SYSHANDLE;
   rc       : INT;
   str_in   : STRING;
   str_out  : STRING;
END_VAR;

   str_in := "Time since reset: $${time}, Temperature: $${temp}, Values:
$"$${time}, $${temp}, $${value}$"";

   // Parse string and create template.
   rc := strTemplateCreate(st:=st, str:=str_in, sov := "$${", eov := "}");
   DebugFmt(message:="strTemplateCreate: \1", v1:=rc);

   // Set the value for the fixed variable
   rc := strTemplateSetVar(st:=st, name:="value", value := "value");
   DebugFmt(message:="strTemplateSetVar: \1", v1:=rc);

BEGIN
   // Set the value for the dynamic variables
   rc := strTemplateSetVar(st:=st, name:="time",
value:=dintToStr(v:=boardTimeSinceReset()/1000));
   DebugFmt(message:="strTemplateSetVar: \1", v1:=rc);

   rc := strTemplateSetVar(st:=st, name:="temp",
value:=floatToStr(v:=FLOAT(boardTemperature())/100.0));
   DebugFmt(message:="strTemplateSetVar: \1", v1:=rc);
```

```
    // Generate string
    rc := strTemplateGenerateString(st:=st, str:=str_out);
    DebugFmt(message:="strTemplateGenerateString: \1", v1:=rc);

    // Print string to device output
    DebugMsg(message:=str_out);

    Sleep(delay:=10000);
END;

END_PROGRAM;
```

4.1.15.34 strTemplateVarInfo (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

---

strTemplateVarInfo is used to enumerate over the variables found in the string template.

Input:

st: SYSHANDLE
A handle to the string template.

idx : INT*(Default 0)*
The index of the variable to get information about.

Output:

name : STRING
The name of the variable.

Returns: INT

>0	- The number of times the variable is used in the template.
0	- Function is not supported.
-1	- Invalid handle.
-2	- Template is not ready.
-4	- Variable was not found.

Declaration:

```
FUNCTION strTemplateVarInfo;
VAR_INPUT
    st        : SYSHANDLE;
    idx       : INT;
    name      : ACCESS STRING;
END_VAR;
```

Example:

```
// Function to list the variables in a template
FUNCTION dump;
VAR_INPUT
    st    : SYSHANDLE;
END_VAR;
VAR
```

```
   i    : INT;
   rc   : INT;
   name : STRING;
END_VAR;
   i := 0;
   DebugMsg(message:="Variables:");
   rc := strTemplateVarInfo(st := st, idx := i, name := name);
   DebugMsg(message:=" Idx Usage Name");
   WHILE rc > 0 DO
      DebugFmt(message := " [\1]   \2   " + name, v1 := i, v2 := rc);
      i := i + 1;
      rc := strTemplateVarInfo(st := st, idx := i, name := name);
   END_WHILE;
END_FUNCTION;
```

4.1.15.35 strTemplateSetVar (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

---

strTemplateSetVar updates the value of a variable in the template.
If a variable is not set with this function it will by default just use an empty string.

Input:

st: SYSHANDLE
A handle to the string template.

name : STRING
The name of the variable to update.

value : STRING
The new value of the variable.

Returns: INT

1	- Success
0	- Function is not supported.
-1	- Invalid handle.
-2	- Template is not ready.
-4	- Variable was not found.

Declaration:

```
FUNCTION strTemplateSetVar;
VAR_INPUT
   st       : SYSHANDLE;
   name     : STRING;
   value    : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
```

```
PROGRAM test;
VAR
   st       : SYSHANDLE;
   rc       : INT;
   str_in   : STRING;
   str_out  : STRING;
END_VAR;

   str_in := "Time since reset: $${time}, Temperature: $${temp}, Values:
$"$${time}, $${temp}, $${value}$"";

   // Parse string and create template.
   rc := strTemplateCreate(st:=st, str:=str_in, sov := "$${", eov := "}");
   DebugFmt(message:="strTemplateCreate: \1", v1:=rc);

   // Set the value for the fixed variable
   rc := strTemplateSetVar(st:=st, name:="value", value := "value");
   DebugFmt(message:="strTemplateSetVar: \1", v1:=rc);

BEGIN
   // Set the value for the dynamic variables
   rc := strTemplateSetVar(st:=st, name:="time",
value:=dintToStr(v:=boardTimeSinceReset()/1000));
   DebugFmt(message:="strTemplateSetVar: \1", v1:=rc);

   rc := strTemplateSetVar(st:=st, name:="temp",
value:=floatToStr(v:=FLOAT(boardTemperature())/100.0));
   DebugFmt(message:="strTemplateSetVar: \1", v1:=rc);

   // Generate string
   rc := strTemplateGenerateString(st:=st, str:=str_out);
   DebugFmt(message:="strTemplateGenerateString: \1", v1:=rc);

   // Print string to device output
   DebugMsg(message:=str_out);

   Sleep(delay:=10000);
END;

END_PROGRAM;
```

4.1.15.36 strTemplateGenerateString (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

---

strTemplateGenerateString generates a new dynamic string from the template, using the current value of the variables.

Input:

st: SYSHANDLE
A handle to the string template.

Output:

str : STRING
The generated string.

Returns: INT

1 - Success
0 - Function is not supported.
-1 - Invalid handle.
-2 - Template is not ready.
-3 - Not enough resources to create string.

Declaration:

```
FUNCTION strTemplateGenerateString;
VAR_INPUT
   st       : SYSHANDLE;
   str      : ACCESS STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc


PROGRAM test;
VAR
   st       : SYSHANDLE;
   rc       : INT;
   str_in   : STRING;
   str_out  : STRING;
END_VAR;

   str_in := "Time since reset: $${time}, Temperature: $${temp}, Values:
$"$${time}, $${temp}, $${value}$"";

   // Parse string and create template.
   rc := strTemplateCreate(st:=st, str:=str_in, sov := "$${", eov := "}");
   DebugFmt(message:="strTemplateCreate: \1", v1:=rc);

   // Set the value for the fixed variable
   rc := strTemplateSetVar(st:=st, name:="value", value := "value");
   DebugFmt(message:="strTemplateSetVar: \1", v1:=rc);

BEGIN
   // Set the value for the dynamic variables
   rc := strTemplateSetVar(st:=st, name:="time",
value:=dintToStr(v:=boardTimeSinceReset()/1000));
   DebugFmt(message:="strTemplateSetVar: \1", v1:=rc);

   rc := strTemplateSetVar(st:=st, name:="temp",
value:=floatToStr(v:=FLOAT(boardTemperature())/100.0));
   DebugFmt(message:="strTemplateSetVar: \1", v1:=rc);

   // Generate string
   rc := strTemplateGenerateString(st:=st, str:=str_out);
   DebugFmt(message:="strTemplateGenerateString: \1", v1:=rc);

   // Print string to device output
   DebugMsg(message:=str_out);

   Sleep(delay:=10000);
END;

END_PROGRAM;
```

4.1.16 SFL: Timer functions

4.1.16.1 SFL: Timer functions

The VPL environment contains a number of timer function blocks:

- TOF — Off-delay timer.
- TON — On-delay timer.
- TP — Pulse timer.
- TPERIOD — Programmable ON/OFF timer.

The resolution of the timers are 10 milliseconds.
For NX32L based devices there is a total of 200 timers and under X32/NX32 there is maximum of 50 timers.

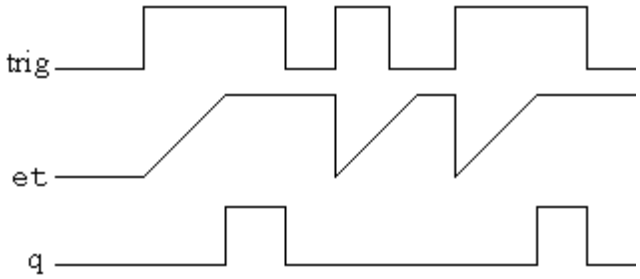
4.1.16.2 TON (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

The TON function block is an ON delay timer that enables the "q" output a specific number of milliseconds after the trig input is enabled.

When the trig input is set TRUE, the elapsed time counter (et) is reset to zero and then incremented until the preset timer value (pt) is reached, after which the "q" output is set TRUE.
When the trig input is switched to FALSE, the "q" output is set FALSE.
If the trig input is set to FALSE before the et counter reaches the pt value, the timer keeps running but the "q" output is not set TRUE when the et counter reaches the pt value.

This image shows the TON function block timing:



Input:

trig : BOOL (true/false)
On the leading edge of this input, the output "q" will go high and the timer will start. When this input goes low, the "q" output will also go low.

pt : DINT (0..2147483648)
Specifies the delay in ms. from "trig" going high to "q" going high.

Output:

et : DINT (0..2147483648)
The current value of the timer.

q : BOOL (true/false)
Output from the timer. See description for "trig".

Declaration:

```
FUNCTION_BLOCK TON;
VAR_INPUT
   trig : BOOL; | Signal that starts timer, and sets 'q' high
   pt   : DINT; | Delay before 'q' goes high after 'trig' signal
END_VAR;
VAR_OUTPUT
   q    : BOOL; | Output from timer
   et   : DINT; | Current timer value
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   input1 : BOOL; | Input that will enable the timer
END_VAR;

VAR_OUTPUT
   signal : BOOL; | Output that will follow the q output from the timer.
END_VAR;

VAR
   timer : TON; // Declare an instance of the TON functionblock
END_VAR;

PROGRAM test;

BEGIN
   timer();
   ...
   timer.pt := 100; // number of milliseconds the q output will be delayed
from 'trig'
   timer.trig := input1; // The timer will start when input1 goes high
   signal := timer.q; // output will follow the q output from the timer
   ...
END;
END_PROGRAM;
```

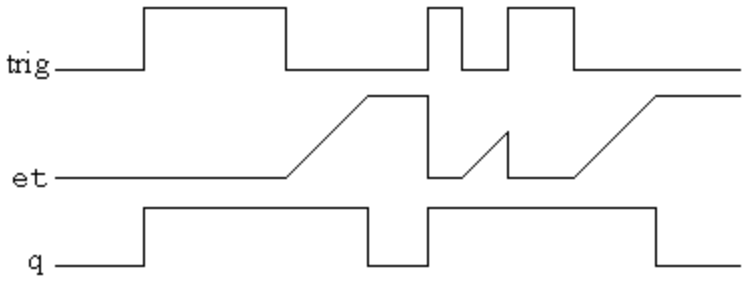
4.1.16.3 TOF (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

The TOF function block is an OFF delay timer that disables the "q" output a specified number of milliseconds after the trig input is disabled.

When the trig input is set TRUE, the elapsed time counter (et) is reset to zero and the "q" output is set TRUE.
When the trig input is switched to FALSE, the "et" counter is incremented until it reaches the preset timer value (pt), after which the "q" output is set FALSE.
If the trig input is set TRUE before the et counter reaches the pt value, the timer is stopped, and the et counter is reset to zero.

This image shows the TON function block timing:



Input:

trig : BOOL (true/false)
On the leading edge of this input, the output "q" will go high. On the falling edge, the timer will start. When the timer reaches "pt", the "q" output will go low.

pt : DINT (0..2147483648)
Specifies the delay in ms. from "trig" going low to "q" going low.

Output:

et : DINT (0..2147483648)
The current value of the timer.

q : BOOL (true/false)
Output from the timer. See description for "trig".

Declaration:

```
FUNCTION_BLOCK TOF;
VAR_INPUT
   trig : BOOL; | Signal that sets 'q' high, starts timer on low signal
   pt   : DINT; | Delay before 'q' goes low after 'trig' signal
END_VAR;
VAR_OUTPUT
   q    : BOOL; | Output from timer
   et   : DINT; | Current timer value
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   input1 : BOOL; | Input that will enable the timer
END_VAR;

VAR_OUTPUT
   signal : BOOL; | Output that will follow the q output from the timer.
END_VAR;

VAR
   timer  : TOF; // Declare an instance of the TOF functionblock
END_VAR;

PROGRAM test;

timer.pt := 100; // number of milliseconds the q output will be delayed from
'trig'
BEGIN
   ...
   timer(trig := input1); // The timer will start when input1 goes low
```

```
   signal := timer.q; // output will follow the q output from the timer
   ...
END;
END_PROGRAM;
```

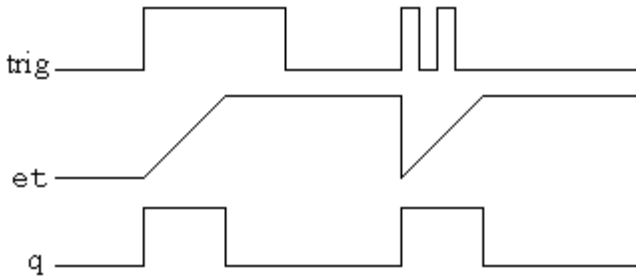
4.1.16.4 TP (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: ALL
Firmware version: 1.00 / 1.00.00

The TP function block is a pulse timer that enables "q" (the output) for a preset amount of time.

When trig is set TRUE, the "q" output is set TRUE and the elapsed time counter (et) is reset to zero and incremented until the preset timer value is reached, after which the "q" output is set FALSE.
Only during this period of time is "q" set to TRUE.
Once the timer is running, any changes to the trig input are ignored until after the preset value is reached.

This image shows the TP function block timing:



Input:

trig : BOOL (true/false)
On the leading edge of this input, the timer will start and set the "q" output true for the specified "pt" time

pt : DINT (0..2147483648)
Specifies how long the "q" output should be true - specified in ms.

Output:

et : DINT (0..2147483648)
The current value of the timer.

q : BOOL (true/false)
Output from the timer. This will go high until "et" equals "pt"

Declaration:

```
FUNCTION_BLOCK TP;
VAR_INPUT
   trig : BOOL; | Leading edge on this signal will start the timer
   pt   : DINT; | Time where 'q' will be high after 'trig' signal
END_VAR;
VAR_OUTPUT
   q    : BOOL; | Output from timer
   et   : DINT; | Current timer value
END_VAR;
```

Example:

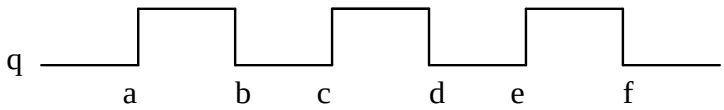
```
INCLUDE rtcu.inc

VAR_INPUT
   input1 : BOOL; | Input that will enable the timer
END_VAR;

VAR_OUTPUT
   signal : BOOL; | Output that will follow the q output from the timer.
END_VAR;

VAR
   timer : TP; // Declare an instance of the TP functionblock
END_VAR;

PROGRAM test;

BEGIN
   timer();
   ...
   timer.pt := 100; // number of milliseconds the q output should be high
   timer.trig := input1; // The timer will start on a leading edge on input1
   signal := timer.q; // output will follow the q output from the timer
   ...
END;
END_PROGRAM;
```

4.1.16.5 TPERIOD (Functionblock)

Architecture: **X32 / NX32 / NX32L**
Device support: **ALL**
Firmware version: **1.00 / 1.00.00**

TPERIOD is a periodic timer. This function block will activate its "q" output when it is enabled by using the enable input. The high_period specifies the period where the "q" output should be true, and the low_period specifies the period where the "q" output should be false. The periods are specified in milliseconds.

Timing:



b-a, d-c, f-e = high_period
c-b, e-d = low_period

Notes:

- This functionblock uses the TP function block.

Input:

enable : BOOL (true/false) ***Default FALSE***
If this input is true, the timer is enabled.

high_period : INT (0..32767)
Specifies how many milliseconds the "q" output should be true.

low_period : INT (0..32767)
Specifies how many milliseconds the "q" output should be false.

Output:

q : BOOL (true/false)
Output from the timer.

Declaration:

```
FUNCTION_BLOCK TPERIOD;
VAR_INPUT
   enable      : BOOL := FALSE; |'TPERIOD' operation enabled.
   low_period  : INT;           | Period for low-signal
   high_period : INT;           | Period for high-signal
END_VAR;
VAR_OUTPUT
   q           : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   input1 : BOOL; | Input that will enable the periodic timer
END_VAR;

VAR_OUTPUT
   signal : BOOL; | Output that will follow the q output from the periodic
timer.
END_VAR;

VAR
   tper : TPERIOD; // Declare an instance of the TPERIOD functionblock
END_VAR;

PROGRAM test;

BEGIN
   tper();
   ...
   tper.high_period := 20; // number of ticks the q output should be high
   tper.low_period := 20; // number of ticks the q output should be low
   tper.enable := input1; // The timer is enabled when input1 is true
   signal := tper.q; // output will follow the q output from the timer
   ...
END;
END_PROGRAM;
```

4.2 SFL: Platform Support Functions

4.2.1 SFL: Platform Support Functions

To facilitate easy access to the many different features offered by the RTCU platform, a number of functions are available to the programmer.
Please consult the technical documentation for the actual RTCU device for detailed information on supported functionality.

The complete list of platform functions are:

- acc — 3D accelerometer functions.
- audio — Audio functions.
- bat — Battery charger functions.
- board — Specific functions for the RTCU core platform.
- ble — Bluetooth communication(LX).
- bt — Bluetooth communication(NX).
- cam — Control of a camera module.
- can — CAN communications.
- display — Control of an LCD Display.
- dtmf — DTMF user interaction (voice response system).
- emu — Emulator functions.
- ext — Platform Extension functions.
- fs — File system functions.
- ftp — FTP functions.
- gnss — GNSS Receiver.
- gprs — Cellular network control functions.
- gps — GPS receiver.
- gsm — GSM module interaction.
- gui — GUI functions.
- gw — RTCU Communcation Hub / Gateway functions.
- hsio — High-Speed IO functions.
- json — JSON functions.
- jwt — JSON Web Tokens
- mbus — M-Bus communication.
- mdt — MDT interaction.
- misc — Other functions.
- modbus — MODBUS communication.
- mqtt — MQTT communication.
- ms — Motion Sensor
- nav — Navigation interface.
- net — Network.
- nmp — Navigation and Messaging Platform.
- ow — OneWire device support.
- pm — Power management functions.
- rest — REST and HTTP functions.
- rf — Wireless communication.
- rfbc — Wireless bi-directional communication.
- sec — Certificate management functions.
- ser — Serial (RS232/RS485) communication.
- smtp — SMTP interface.
- so — Socket interface
- sock — TCP/IP Socket interface.
- sos — System Object Storage.

• udp	UDP/IP interface.
• usb	USB host functions.
• ver	Application version functions.
• voice	Voice messages (voice response system).
• zp	Zero power functions.

4.2.2 acc: 3D Accelerometer

4.2.2.1 acc: 3D Accelerometer

The 3D accelerometer API supports applications for such things as collision detection, monitoring driving experience (vibration and harshness), inertial navigation, etc.

It is possible to start a logging feature which will automatically collect current acceleration with a minimum of resource requirement - thus allowing for a continuous collection of data while the device is performing other tasks. This functionality will allow implementation of a "black box" that collects various information such as G-forces, GPS positions, and CAN bus information at the time of a given incident or event.

The acceleration values are returned as **mg units** (SI standard: 0.001 m/s2 [meters per second squared]).
The maximum acceleration that can be detected is 16000 mg.
The axis of the acceleration is represented relative to the device (please see the technical manual for details).

Architecture of the Accelerometer API

The Accelerometer API is based on an asynchronous event-based model where solicited as well as unsolicited events from the accelerometer are automatically queued and delivered to the user application. The function accWaitEvent is used to return information about the various events arriving from the accelerometer. This model eliminates the requirement for the application to continuously poll for messages, and it also encourage use of multithreading with the added benefits that comes with this.
Events that are triggered by the application requesting various information from the accelerometer are therefore called *solicited* events, or, alternatively, they can be triggered by the accelerometer itself and are therefore instead called *unsolicited* events. A solicited event is, for example, when the application calls accLoggerStop and then accWaitEvent returns with event# 4. An unsolicited event works exactly the same way as a solicited event, except for the fact that accLoggerStop is no longer called by the application.
Please consult accWaitEvent for detailed information about the various events and how they are handled.

The following functions/function blocks are available:

• accOpen	Opens the interface.
• accClose	Closes the interface.
• accVector	Reads an acceleration vector.
• accVectorToForce	Calculates the total force of acceleration.
• accVectorToAngles	Calculates axis angles relative to acceleration.
• accWaitEvent	Waits for an event from the acceleration interface.
• accSetAccelerationEvent	Control monitoring of acceleration.
• accAccelerationEvent	Reads acceleration event.
• accSetShockEvent	Controls monitoring of shocks.
• accShockEvent	Reads shock event.
• accLoggerSetConfig	Configures logging feature.
• accLoggerGetConfig	Reads logging configuration and status.
• accLoggerStart	Starts logging.
• accLoggerStop	Stops logging.
• accLoggerLevel	Returns the amount of buffer used by the logger.
• accLoggerRead	Reads value from logger
• accLoggerToMem	Moves the contents of the logger to memory.

- accVibrationSetWakeup — Enable or disable the ability to wake from low power modes with device vibration. See pmSuspend.

4.2.2.2 accOpen (Function)

```
Architecture:       X32 / NX32 / NX32L
Device support:     CX1, SX1, MX2 turbo/encore/warp/warp, NX-200, NX-
                    900, LX2, LX5
Firmware version:   2.40 / 1.40.00
```

This function will open the accelerometer interface. Calling this function is required before the accelerometer interface can be used.
This interface and the Motion Sensor (MS) interface are mutually exclusive, thus only one of them can be used at a time. See also msOpen.

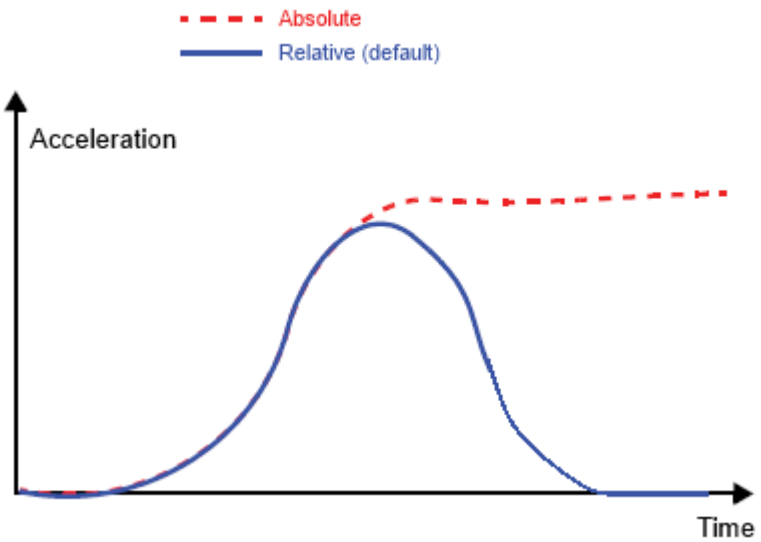
The device is unable to detect vibration (using pmVibration or pmWaitEvent) when the acceleration interface is open.

Modes
The mode parameter will determine how if acceleration event and readings are relative to motion or gravity of earth.

1: RELATIVE.
The changes in g-forces are measured as yielding the acceleration relative to the actual motion. With a steady acceleration, all axes will read zero.

2: ABSOLUTE.
All g-forces acting on the device are measured - therefore giving the absolute acceleration. This includes the force of earth gravity, which is to say that when it is not in motion and placed on at flat surface, the Z-axis will read -1000.

3: MIXED.
All g-forces acting on the device are measured as in ABSOLUTE mode, while events are triggered relative to motion as in RELATIVE mode.



Acceleration measured as it reaches a constant acceleration in both ABSOLUTE and RELATIVE mode.

Input:

Mode : SINT (1..3, Default 1)
Determine how accelerations and events are measured and triggered, for details see description of the modes above.

Returns: INT

0- Success.
- 1- General error.
- 2- Interface already open.
- 3- Invalid mode.
- 4- Not enough memory.
-5- Motion Sensor interface is open. Use msClose to close interface.

Declaration:

```
FUNCTION accOpen : INT;
VAR_INPUT
   Mode : SINT := 1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
   accOpen();
   ...
   accClose();
END_PROGRAM;
```

4.2.2.3 accClose (Function)

Architecture: X32 / NX32 / NX32L
Device support: CX1, SX1, MX2 turbo/encore/warp, NX-200, NX-900, LX2, LX5
Firmware version: 2.40 / 1.40.00

This function will close the accelerometer interface. After the accelerometer interface is closed, it cannot be used until opened again.

Input:

None.

Returns: INT

0- Success.
- 1- General error.
- 2- Interface not open (see accOpen).

Declaration:

```
FUNCTION accClose : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
   accOpen();
   ...
   accClose();
END_PROGRAM;
```

4.2.2.4 accVector (Function)

Architecture: X32 / NX32 / NX32L
Device support: CX1, SX1, MX2 turbo/encore/warp, NX-200, NX-900, LX2, LX5
Firmware version: 2.40 / 1.40.00

This function will retrieve the current acceleration vector measured by the accelerometer. Returned values are in mg.

Input:
None

Output:
x : DINT
The acceleration force measured on the X-axis (in mg).

y : DINT
The acceleration force measured on the Y-axis (in mg).

z : DINT
The acceleration force measured on the Z-axis (in mg).

Returns: INT
0- Success.
-1- General error.
-2- Interface not open (see accOpen)

Declaration:
```
FUNCTION accVector : INT;
VAR_INPUT
   X : ACCESS DINT;
   Y : ACCESS DINT;
   Z : ACCESS DINT;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   x,y,z : DINT;
END_VAR;
   accOpen();
BEGIN
   accVector(x := x, y := y, z := z);
   ...
END;
END_PROGRAM;
```

4.2.2.5 accVectorToForce (Function)

Architecture: X32 / NX32 / NX32L
Device support: CX1, SX1, MX2 turbo/encore/warp, NX-200, NX-900, LX2, LX5
Firmware version: 2.40 / 1.40.00

This function will take the applied values and calculate the total force.

Returned values are in mg.

Input:
x : DINT
The acceleration force measured on the X-axis (mg).

y : DINT
The acceleration force measured on the Y-axis (mg).

z : DINT
The acceleration force measured on the Z-axis (mg).

Output:
None

Returns: DINT
>=0 - The total acceleration force (in mg).

Declaration:
```
FUNCTION accVectorToForce : DINT;
VAR_INPUT
   X : DINT;
   Y : DINT;
   Z : DINT;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   x,y,z : DINT;
   force : DINT;
END_VAR;
accOpen();
BEGIN
  accVector(x := x, y := y, z := z);
  force := accVectorToForce(x := x, y := y, z := z);
  ...
END;
END_PROGRAM;
```

4.2.2.6 accVectorToAngles (Function)

Architecture: X32 / NX32 / NX32L
Device support: CX1, SX1, MX2 turbo/encore/warp, NX-200, NX-900, LX2, LX5
Firmware version: 2.40 / 1.40.00

This function will take the applied values and calculate the angles of the axis relative to acceleration forces.

When mode is ABSOLUTE (see accOpen) and not moving, the angles returned will represent the tilt of each axis relative to the earth surface.

Input:
inX : DINT
The acceleration force (mg) measured on the X-axis.

inY : DINT
The acceleration force (mg) measured on the Y-axis.

inZ : DINT
The acceleration force (mg) measured on the Z-axis.

Output:
x : INT
Angle of the X-axis relative to acceleration forces in 0.01 degrees.

y : INT
Angle of the Y-axis relative to acceleration forces in 0.01 degrees.

z : INT
Angle of the Z-axis relative to acceleration forces in 0.01 degrees.

Returns: INT
0 -Ok.

Declaration:
```
FUNCTION accVectorToAngles : INT;
VAR_INPUT
   inX   : DINT;
   inY   : DINT;
   inZ   : DINT;

   x     : ACCESS INT;
   y     : ACCESS INT;
   z     : ACCESS INT;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   x,y,z : DINT;
   pitch : INT;
   roll  : INT;
   yaw   : INT;
END_VAR;
accOpen(mode := 2);
BEGIN
  accVector(x := x, y := y, z := z);
  accVectorToAngles(inX := x, inY := y, inZ := z, x := pitch, y := roll, z :=
yaw);
  ...
END;
END_PROGRAM;
```

4.2.2.7 accWaitEvent (Function)

Architecture: X32 / NX32 / NX32L
Device support: CX1, SX1, MX2 turbo/encore/warp, NX-200, NX-900, LX2, LX5
Firmware version: 2.40 / 1.40.00

This function will wait until an event is raised with an optional timeout.

An event is a notification that something has happened in the accelerometer interface.
There are two types of events:

1. Events triggered by the application (*solicited event*).
2. Events triggered by the Accelerometer (*unsolicited event*).

The events are queued until appropriate action is taken as described in this table:

Event#	Event	Solicited	Action
1	Events has been lost due to too many events in queue.	No	
2	Logging is stopped due to acceleration.	No	
3	Logging is stopped due to shock.	No	
4	Logging is stopped due to buffer is full or stopped by application	Yes	
5	Logging has reached warning level.	No	*Optional:* Save logging buffer.
6	Logging buffer is full.	No	*Optional:* Save logging buffer.
7	Acceleration surpassing requested limits detected.	No	Call accAccelerationEvent.
8	Shock surpassing the requested limits detected.	No	Call accShockEvent.

accWaitEvent notifies the application of the oldest queued event.
This means that if the appropriate action is not taken, accWaitEvent will continue to report the same event.

Note: waiting for an event by using this function will block the calling thread.

Input:

Timeout : INT (-1,0..32000, default -1)
Timeout period in milliseconds to wait.

- 0 - Disable timeout. The function will return immediately.
- -1 - Wait forever. The function will only return when data is received.

Output:

None.

Returns: INT

- 8 - Shock surpassing the requested limits detected.
- 7 - Acceleration surpassing requested limits detected.
- 6 - Logging buffer is full.
- 5 - Logging has reached warning level.
- 4 - Logging is stopped due to buffer is full or stopped by application.
- 3 - Logging is stopped due to shock.
- 2 - Logging is stopped due to acceleration.
- 1 - Events have been lost due to too many events in queue.
- 0 - Timeout.
- -1 - General error.
- -2 - Accelerometer interface is not open.

-3 - Invalid timeout.

Declaration:

```
FUNCTION accWaitEvent : INT;
VAR_INPUT
   timeout : INT;
END_VAR;
```

Example:

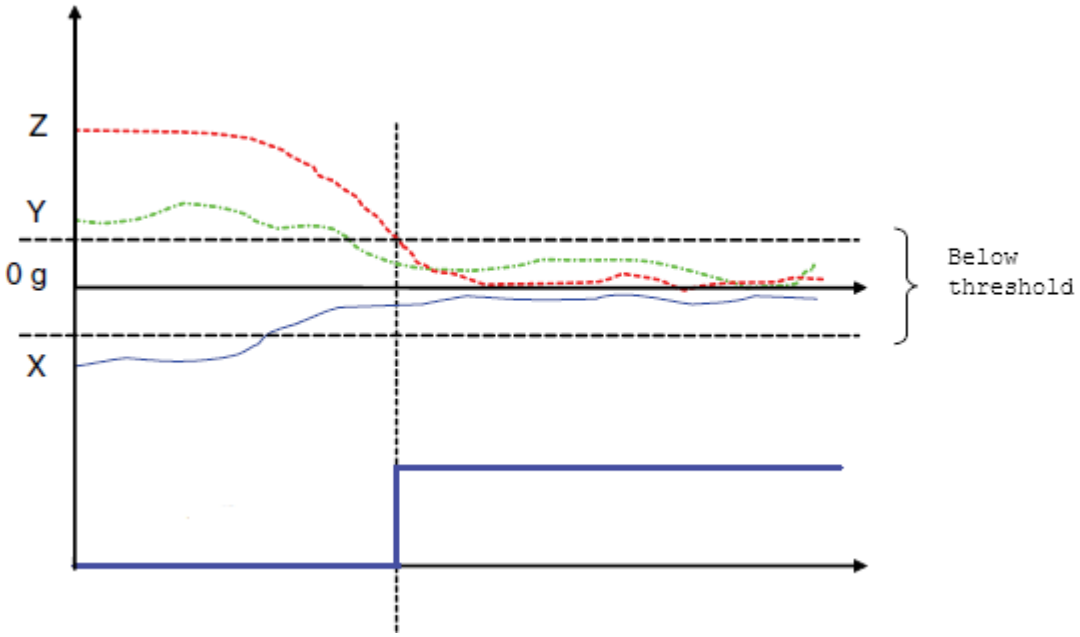
```
INCLUDE rtcu.inc

THREAD_BLOCK accMonitor;
VAR
   event : INT := 0;
END_VAR;
WHILE event <> -1 DO
   event := accWaitEvent(timeout := -1);
   CASE event OF
      1: DebugMsg(message := "Events has been lost due to too many events in
queue");
      2: DebugMsg(message := "Logging is stopped due to acceleration event");
      3: DebugMsg(message := "Logging is stopped due to shock event");
      4: SendAccLog();
      5: DumpAccLog();
      6: DumpAccLog();
         DebugMsg(message := "Logging buffer is full, data could be lost");
      7: ReadAccEvent();
      8: ReadShockEvent();
   ELSE
      DebugFmt(message := "accWaitEvent - event=\1", v1 := event);
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;
```

4.2.2.8 accSetAccelerationEvent (Function)

Architecture: X32 / NX32 / NX32L
Device support: CX1, SX1, MX2 turbo/encore/warp, NX-200, NX-900, LX2, LX5
Firmware version: 2.40 / 1.40.00

This function will enable or disable monitoring of the acceleration.

When monitoring is enabled, the accelerometer will compare the acceleration of an axis with a threshold value.
If the acceleration is above (or below) the threshold for the selected duration, an "Acceleration" event is raised. See also accWaitEvent for event # 7.
The accelerometer will continue to raise the event after each duration interval until the acceleration moves below (or above) the threshold again.



If combined is true, then all axis conditions must be true for the event to be raised as illustrated above where all axes are required to be below threshold (also know as free fall).

Input:

Enable : BOOL (default FALSE)
Is event generation enabled.

Once : BOOL (default FALSE)
Event must be enabled again after detection.

Threshold: DINT (0..16000, default 0)
The acceleration which each axis is compared against (in mg).

Duration : DINT (0..5120, default 0)
Number of milliseconds the axis must be high/low before the axis condition is true.

Combined : BOOL (default FALSE)
Whether all axis conditions should be true or just a single one before the event is raised.

XAbove : BOOL (default FALSE)
Fire event when x-axis is above threshold for the duration.

XBelow : BOOL (default FALSE)
Fire event when x-axis is below threshold for the duration.

YAbove : BOOL (default FALSE)
Fire event when y-axis is above threshold for the duration.

YBelow : BOOL (default FALSE)
Fire event when y-axis is below threshold for the duration.

ZAbove : BOOL (default FALSE)
Fire event when z-axis is above threshold for the duration.

ZBelow : BOOL (default FALSE)
Fire event when z-axis is below threshold for the duration.

Output:

None.

Returns: INT

0- OK
-1- General error.
-2- Interface not open (see accOpen).
-3- Invalid threshold.
-4- Invalid duration.
-5- Conflict in combined axis.

Declaration:

```
FUNCTION accSetAccelerationEvent : INT;
VAR_INPUT
   Threshold   : DINT := 0;
   Duration    : DINT := 0;
   Once        : BOOL := FALSE;
   Combined    : BOOL := FALSE;
   XAbove      : BOOL := FALSE;
   XBelow      : BOOL := FALSE;
   YAbove      : BOOL := FALSE;
   YBelow      : BOOL := FALSE;
   ZAbove      : BOOL := FALSE;
   ZBelow      : BOOL := FALSE;
   Enable      : BOOL := FALSE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
   accOpen();
   accSetAccelerationEvent(Threshold := 500, Duration := 10, XAbove := TRUE,
Enable := TRUE);
BEGIN
  e := accWaitEvent(timeout := 60);
  ...
END;
END_PROGRAM;
```

4.2.2.9 accAccelerationEvent (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: CX1, SX1, MX2 turbo/encore/warp, NX-200, NX-900, LX2, LX5
Firmware version: 2.40 / 1.40.00

accAccelerationEvent returns the information of an acceleration event.
Note that the information is only valid after an "Acceleration" event is raised; the event will be removed when this function block is called.
An "Acceleration" event is raised when the acceleration matches the parameters set when monitoring was enabled.

Input:

None.

Output:

XAbove : BOOL
Was x-axis above threshold for the duration.

XBelow : BOOL
Was x-axis below threshold for the duration.

YAbove : BOOL
Was y-axis above threshold for the duration.

YBelow : BOOL
Was y-axis below threshold for the duration.

ZAbove : BOOL
Was z-axis above threshold for the duration.

ZBelow : BOOL
was z-axis below threshold for the duration.

X : DINT
Last x-axis reading before event (mg).

Y : DINT
Last y-axis reading before event (mg).

Z : DINT
Last z-axis reading before event (mg).

Ready : BOOL
Indicates whether event was removed and data updated.

Declaration:

```
FUNCTION_BLOCK accAccelerationEvent;
VAR_OUTPUT
   X           : DINT;
   Y           : DINT;
   Z           : DINT;
   XAbove      : BOOL;
   XBelow      : BOOL;
   YAbove      : BOOL;
   YBelow      : BOOL;
   ZAbove      : BOOL;
   ZBelow      : BOOL;
   Ready       : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   acc : accAccelerationEvent;
END_VAR;

FUNCTION ReadAccEvent;
   acc();
   IF acc.ready THEN
      DebugMsg(message := "*** acceleration event received ***");
      DebugFmt(message := "-x=\4", v4 := acc.x);
      DebugFmt(message := "-y=\4", v4 := acc.y);
      DebugFmt(message := "-z=\4", v4 := acc.z);
   END_IF;
END_FUNCTION;
```

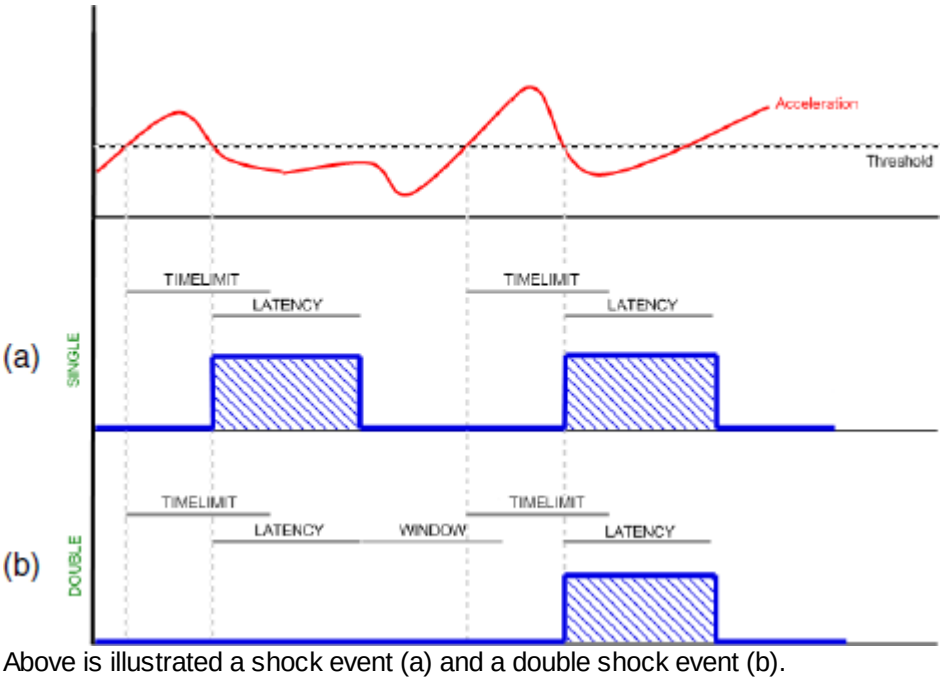
```
THREAD_BLOCK accMonitor;
VAR
   event : INT := 0;
END_VAR;

WHILE event <> -1 DO
   event := accWaitEvent(timeout := -1);
   CASE event OF
      ...
      7: ReadAccEvent();
      ...
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;
```

4.2.2.10 accSetShockEvent (Function)

Architecture: X32 / NX32 / NX32L
Device support: CX1, SX1, MX2 turbo/encore/warp, NX-200, NX-900, LX2, LX5
Firmware version: 2.40 / 1.40.00

This function will enable or disable monitoring of shocks.
A shock is defined as the measured acceleration exceeding a threshold and returning below the threshold before the duration limit expires.
A double shock is defined as two shocks detected after each other where the second shock must start within a time window after the first has ended.

If a shock (or double shock) is detected, a "Shock" event is raised. See also accWaitEvent for event # 8.



Above is illustrated a shock event (a) and a double shock event (b).

Input:

Enable : BOOL (*default FALSE*)
Is event generation enabled. If disabled, all other parameters are ignored.

Once : BOOL (*default FALSE*)
Event must be enabled again after event has been detected

Threshold : DINT (0..16000, *default 0*)
The acceleration which each axis is compared against to detect the start and end of a shock (in mg).

Duration : DINT (1..5120)
The time in milliseconds before which the end of a shock must occur.

Latency : DINT (0..10240, *default 0*)
Number of milliseconds where shocks are ignored.

Window : DINT (0..10240, *default 0*)
Number of milliseconds before the second shock must occur.
0 : single shock detection
>0 : double shock detection

X : BOOL (*default TRUE*)
Detects shocks on x-axis

Y : BOOL (*default TRUE*)
Detects shocks on y-axis

Z : BOOL (*default TRUE*)
Detects shocks on z-axis

Returns: INT

0 - OK
-1 - General error.
-2 - Interface not open (see accOpen).
-3 - Invalid argument.

Declaration:

```
FUNCTION accSetShockEvent : INT;
VAR_INPUT
   Threshold : DINT;
   Duration  : DINT;
   Latency   : DINT;
   Window    : DINT;
   Once      : BOOL;
   X         : BOOL;
   Y         : BOOL;
   Z         : BOOL;
   Enable    : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   e : INT;
END_VAR;
accOpen();
accSetShockEvent(Threshold := 500, Duration := 10, Z := TRUE, Enable := TRUE);
BEGIN
  e := accWaitEvent(timeout := 60);
  ...
```

```
END;
END_PROGRAM;
```

4.2.2.11 accShockEvent (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: CX1, SX1, MX2 turbo/encore/warp, NX-200, NX-900, LX2, LX5
Firmware version: 2.40 / 1.40.00

---

accShockEvent retirnes the information of a shock event.
Note that the information is only valid after a "Shock" event is raised; the event will be removed when this function block is called.
A "Shock" event is raised when a shock is experienced that matches the parameters set when monitoring was enabled.

Input:

None.

Output:

X : BOOL
Detected shock on x-axis.

Y : BOOL
Detected shock on y-axis.

Z : BOOL
Detected shock on z-axis.

Ready : BOOL
Is event removed and data updated.

Declaration:

```
FUNCTION_BLOCK accShockEvent;
VAR_OUTPUT
   X     : BOOL;
   Y     : BOOL;
   Z     : BOOL;
   Ready : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   shock : accShockEvent;
END_VAR;

FUNCTION ReadShockEvent;
   shock();
   IF shock.ready THEN
      DebugMsg(message := "*** acceleration event received ***");
      DebugFmt(message := "-x=\1", v1 := INT(shock.x));
      DebugFmt(message := "-y=\1", v1 := INT(shock.y));
      DebugFmt(message := "-z=\1", v1 := INT(shock.z));
   END_IF;
END_FUNCTION;

THREAD_BLOCK accMonitor;
```

```
VAR
   event : INT := 0;
END_VAR;

WHILE event <> -1 DO
   event := accWaitEvent(timeout := -1);
   CASE event OF
      ...
      8: ReadShockEvent();
      ...
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;
```

4.2.2.12 accLoggerSetConfig (Function)

Architecture: X32 / NX32 / NX32L
Device support: CX1, SX1, MX2 turbo/encore/warp, NX-200, NX-900, LX2, LX5
Firmware version: 2.40 / 1.40.00

This function will initialize a buffer where acceleration vectors are logged.
The logging to the buffer is started with accLoggerStart.

Note:
The logging will be suspended when calling the pmWaitEvent function and will resume once the function returns.

Input:

DownSample : SINT (0..127, default 0)
Only collects each X sample generation - allowing logging for a longer period at the risk of losing some measurements.

Buffer : PTR
Address of buffer that is used to store the logged values.

Size : DINT (0,4..2147483647)
Size of buffer area in bytes.

WarningLevel : INT (0..999, default 0)
At what level a "Logger level warning" event will be generated in 0.1 percent, if 0 no warning will be generated.

WarnIfFull : BOOL (default TRUE)
Generates a "Logger is full" event when the data area is full and data begins to be overwritten.

StopOnFull : BOOL (default FALSE)
Stops logger when data area if full.

StopOnShock : BOOL (default FALSE)
Stops logging when shock event is detected.

StopOnAcc : BOOL (default FALSE)
Stops logging when acceleration event is detected.

Returns: **INT**

0- OK
-1- General error.
-2- Interface not open (see accOpen).

-3- Invalid argument.
-4- Logger is running.

Declaration

```
FUNCTION accLoggerSetConfig : INT;
VAR_INPUT
   Buffer      : PTR;
   Size        : DINT := 0;
   WarningLevel: INT  := 0;
   DownSample  : SINT := 0;
   WarnIfFull  : BOOL := TRUE;
   StopOnFull  : BOOL := FALSE;
   StopOnShock : BOOL := FALSE;
   StopOnAcc   : BOOL := FALSE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   buf : ARRAY [1..500] OF DINT;
   cfg : accLoggerGetConfig;
END_VAR;
accOpen();
accLoggerSetConfig(buffer := ADDR(buf), size := SIZEOF(buf));
cfg();
...
BEGIN
  ...
END;
END_PROGRAM;
```

4.2.2.13 accLoggerGetConfig (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: CX1, SX1, MX2 turbo/encore/warp, NX-200, NX-900, LX2, LX5
Firmware version: 2.40 / 1.40.00

This function will return the current configuration and status of the logging
This configuration must first be set with accLoggerSetConfig.

Outputs:

Size : INT
The maximum number of samples the buffer can hold.

Samples : INT
The current number of samples in the buffer.

DownSample : SINT
Skip X number of samples when collecting.

Level : INT
How full the buffer is in 0.1 percent.

WarningLevel : INT
At what level a "Logger level warning" event will be generated in 0.1 percent. If 0 no warning will be generated.

WarnIfFull : BOOL
Generates a "Logger is full" event when the data area is full and data begins to be overwritten.

StopOnFull : BOOL
Stops logger when data area if full.

StopOnShock : BOOL
Stops logging when shock event is detected.

StopOnAcc : BOOL
Stops logging when acceleration event is detected.

Running : BOOL
True if logging is started, False if logging is stopped.

Declaration:

```
FUNCTION_BLOCK accLoggerGetConfig;
VAR_OUTPUT
   Size           : INT;
   Samples        : INT;
   Level          : INT;
   WarningLevel   : INT;
   DownSample     : SINT;
   WarnIfFull     : BOOL;
   StopOnFull     : BOOL;
   StopOnShock    : BOOL;
   StopOnAcc      : BOOL;
   Running        : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   buf : ARRAY [1..500] OF DINT;
   cfg : accLoggerGetConfig;
END_VAR;
accOpen();
accLoggerSetConfig(buffer := ADDR(buf), size := SIZEOF(buf));
cfg();
...
BEGIN
  ...
END;
END_PROGRAM;
```

4.2.2.14 accLoggerStart (Function)

Architecture: X32 / NX32 / NX32L
Device support: CX1, SX1, MX2 turbo/encore/warp, NX-200, NX-900, LX2, LX5
Firmware version: 2.40 / 1.40.00

This function will start (or resume) logging.
The logger must first have been configured with accLoggerSetConfig.

Inputs:

Reset : BOOL (Default FALSE)
If TRUE, then all previous stored values will be removed before start.

Returns: INT

0- Success.
- 1- General error.
- 2- Interface not open (see accOpen).
- 3- Logger not configured.
- 4- Logger already started.

Declaration:

```
FUNCTION accLoggerStart : INT;
VAR_INPUT
   Reset : BOOL := FALSE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   buf : ARRAY [1..500] OF DINT;
END_VAR;
accOpen();
accLoggerSetConfig(buffer := ADDR(buf), size := SIZEOF(buf));
...
BEGIN
  accLoggerStart(reset := TRUE);
  ...
  accLoggerStop();
END;
END_PROGRAM;
```

4.2.2.15 accLoggerStop (Function)

Architecture: X32 / NX32 / NX32L
Device support: CX1, SX1, MX2 turbo/encore/warp, NX-200, NX-900, LX2, LX5
Firmware version: 2.40 / 1.40.00

This function will stop logging.
The logger must first have been started with accLoggerStart.

Inputs:

Event : BOOL (Default FALSE)
If TRUE, then a "Logging stopped" event will be raised.
Note: this will overrule the configuration setting for raising a "Logging stopped" event.

Returns: INT

0- OK.
-1- General error.
-2- Interface not open (see accOpen).
-3- Logger not running.

Declaration:

```
FUNCTION accLoggerStop : INT;
VAR_INPUT
```

```
   Event : BOOL := FALSE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   buf : ARRAY [1..500] OF DINT;
END_VAR;
accOpen();
accLoggerSetConfig(buffer := ADDR(buf), size := SIZEOF(buf));
...
BEGIN
  accLoggerStart(reset := TRUE);
  ...
  accLoggerStop();
END;
END_PROGRAM;
```

4.2.2.16 accLoggerLevel (Function)

Architecture: X32 / NX32 / NX32L
Device support: CX1, SX1, MX2 turbo/encore/warp, NX-200, NX-900, LX2, LX5
Firmware version: 2.40 / 1.40.00

This will return the amount of space used by the logger.
The logger must first have been started with accLoggerStart for buffer to be filled.

Returns: INT

0-1000- Logger level in 0.1 percent.

Declaration:

```
FUNCTION accLoggerLevel : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   buf : ARRAY [1..500] OF DINT;
END_VAR;
accOpen();
accLoggerSetConfig(buffer := ADDR(buf), size := SIZEOF(buf));
accLoggerStart();
BEGIN
  IF accLoggerLevel() > 800 THEN
     ...
  END_IF;
END;
END_PROGRAM;
```

4.2.2.17 accLoggerRead (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: CX1, SX1, MX2 turbo/encore/warp, NX-200, NX-900, LX2, LX5
Firmware version: 2.40 / 1.40.00

This function will read the oldest value stored in the logger.

Input:

None.

Output:

X : DINT
Buffered x-axis reading.

Y : DINT
Buffered y-axis reading.

Z : DINT
Buffered z-axis reading.

Ready : BOOL
Has data been collected

Declaration:

```
FUNCTION_BLOCK accLoggerRead;
VAR_OUTPUT
   X     : DINT;
   Y     : DINT;
   Z     : DINT;
   Ready : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   buf    : ARRAY [1..500] OF DINT;
   reader : accLoggerRead;
END_VAR;
accOpen();
accLoggerSetConfig(buffer := ADDR(buf), size := SIZEOF(buf));
accLoggerStart();
BEGIN
  IF accLoggerLevel() > 800 THEN
     REPEAT
       reader();
       IF reader.ready THEN
          ...
       END_IF;
     UNTIL NOT reader.ready
     END_REPEAT;
  END_IF;
END;
END_PROGRAM;
```

4.2.2.18 accLoggerToMem (Function)

Architecture: X32 / NX32 / NX32L
Device support: CX1, SX1, MX2 turbo/encore/warp, NX-200, NX-900, LX2, LX5
Firmware version: 2.40 / 1.40.00

This function will move the contents of the logger to the memory.
The acceleration vectors will be stored in memory as 3 DINTs, where the first dint contains the x-axis, the second contains the y-axis, and the third contains the z-axis.

Input:

mem :PTR;
Address of destination data area.

size :DINT;
The size of the memory in bytes.

Output:

None.

Returns: INT

>0- Number of bytes written.
-1- General error.
-3- Destination size too small to hold a single reading.

Declaration:

```
FUNCTION accLoggerToMem : INT;
VAR_INPUT
   mem   : PTR;
   size  : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   buf      : ARRAY [1..3000] OF DINT;
   readings : ARRAY [1..100] OF DINT;
   i        : INT;
END_VAR;

PROGRAM example;
accOpen();
accLoggerSetConfig(buffer := ADDR(buf), size := SIZEOF(buf));
accLoggerStart();
BEGIN
   IF accLoggerLevel() > 800 THEN
      i := accLoggerToMem(mem := ADDR(readings), size := SIZEOF(readings));
      IF i > 0 THEN
          ...
      END_IF;
   END_IF;
END;
END_PROGRAM;
```

4.2.2.19 accVibrationSetWakeup (Function)

Architecture: NX32L
Device support: NX-200, NX-900, LX2, LX5
Firmware version: 2.20.00

Configure the system to wake from suspend on detection of vibration. See pmSuspend.

pmSuspend return codes for this wake-up source (shared with msVibrationSetWakeup):

Error	Type	Value	Description
False	4	1	Vibration has woken up the system.
True	4	-10	Sensitivity is 0.
True	4	-11	Accelerometer is open, preventing vibration from working. Call accClose to close it.
True	4	-12	MS is open and accelerometer, gyroscope or magnetometer is active.

Input:

Enable : BOOL
Controls if enabling or disabling wake-up on vibration.

Sensitivity : SINT (-1..100, default -1)
The sensitivity level of the vibration detection. 0 is no vibration detection, 100 is most sensitive. -1 will use the sensitivity used under normal operation. See pmSetVibrationSensivity

To use vibration as a wake-up source, the sensitivity must not be 0.

Returns: INT

0- Success or this function is not supported.
-1- Sensitivity is outside the valid range.

Declaration

```
FUNCTION accVibrationSetWakeup : INT;
VAR_INPUT
   Enable      : BOOL;
   Sensitivity : SINT := -1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
// These are the local variables of the program block
VAR
   wk    : DINT;
END_VAR;
// The next code will only be executed once after the program starts
   ...
   // Enable wake on vibration with high sensitivity
   accVibrationSetWakeup(sensitivity := 80, enable := ON);
   ...
   // Sleep 600 seconds or wake on vibration detection
   wk := pmSuspend(time:=600, mode := 0);
   ...
BEGIN
```

```
// Code from this point until END will be executed repeatedly
END;
END_PROGRAM;
```

4.2.3 audio: Audio Functions

4.2.3.1 audio: Audio Functions

The Audio API is used to control the routing and volume of the audio nodes and devices.
The audio nodes represent the endpoints for the audio, e.g. the speaker, line-out and microphone jacks on an audio device.

There are two ways of connecting audio to the audio jacks:
- gsmHeadset can be used to route the audio between the nodes and the GSM modem.
- voiceSetChannel can be used to route the voice output to the nodes. The table below shows the available nodes on the channels.

Available audio nodes:

Node	Name	Channel	Muted by default
0	Microphone	1	No
1	Line-out	1	No
2	Speaker	1	Yes

The following audio functions are available:

- audioSetVolume — Controls the volume level of an audio node.
- audioGetVolume — Retrieves the volume level from an audio node.
- audioRoute — Controls the routing between audio devices.

4.2.3.2 audioSetVolume (Function)

```
Architecture:       NX32L
Device support:     NX-200, NX-400
Firmware version:   1.00.00
```

The audioSetVolume function is used to control both the volume level used by an audio node and also for muting the node.
The volume level does not refer to an absolute level and may vary between different device types.
Muting can be done separately from changing the volume, to make it easier to manage custom volume levels.

Input:

node: SINT ***Default 0***
The audio node to change the volume for.
Available nodes:

Node	Name	Muted by default
0	Microphone	No
1	Line-out	No
2	Speaker	Yes

volume: SINT (-1,0..100) ***Default -1***
Volume level, where 0 is the lowest and 100 is the highest volume. Setting it to `-1` leaves it at its current value.

mute: BOOL ***Default OFF***
Controls whether the node should be muted or not.

Returns: **INT**

0	- Operation was successful.
-1	- Invalid volume
-2	- Unknown node
-3	- Invalid configuration (microphone can not be enabled without either Line-out or Speaker enabled).

Declaration:

```
FUNCTION audioSetVolume : INT;
VAR_INPUT
   node   : SINT := 0;
   volume : SINT := -1;
   mute   : BOOL := OFF;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

...
BEGIN
   ...
   // Set the Line-out volume to 90%
   audioSetVolume(node:=1, volume := 90);
   // Mute the speaker and set its volume to 10%
   audioSetVolume(node:=2, volume := 10, mute := ON);
   // Route audio between GSM modem and audio jacks
   gsmHeadset(enable:=TRUE);
   ...
END;

END_PROGRAM;
```

4.2.3.3 audioGetVolume (Function)

Architecture: NX32L
Device support: NX-200, NX-400
Firmware version: 1.00.00

Retrieves the volume level used by an audio node.
The volume level does not refer to an absolute level and may vary between different device types.

Input:

node: SINT *Default 0*
The audio node to get the volume for.
Available nodes:

Node	Name
0	Microphone
1	Line-out
2	Speaker

Output:

volume: SINT
The volume of the node, where 0 is the lowest and 100 is the highest volume.

mute: BOOL
Tells if the node is muted or not.

Returns: INT

0 - Operation was successful.
-2 - Unknown node

Declaration:

```
FUNCTION audioGetVolume : INT;
VAR_INPUT
   node   : SINT := 0;
   volume : ACCESS SINT;
   mute   : ACCESS BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   vol   : SINT;
   muted : BOOL;
END_VAR;
...
BEGIN
   ...
   // Get the volume for Line-out
   IF audioGetVolume(node:=1, volume := vol, mute := muted) = 0 THEN
      IF muted THEN
         DebugFmt(message := "Muted. Volume: \1", v1:= vol);
      ELSE
         DebugFmt(message := "Unmuted. Volume: \1", v1:= vol);
      END_IF;
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.3.4 audioRoute (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900, NX-910
Firmware version: 1.10.00

Controls the routing of the audio between capture devices and playback devices.

Instead of using gsmHeadset to route audio between GSM and the analog audio in- and outputs, this function can be used for more advanced routing.
It controls a single direction at a time, so to have audio running in both directions, it must be called twice.
It is currently possible to have 3 routes active at the same time, each using a unique route ID.
Note: gsmHeadset uses two routes when enabled, reducing the total number of available routes.

The possible devices are:

Device ID	Name	Capture	Playback
0	Speaker/Line out	No	Yes
1	GSM output (Audio sent to the GSM device)	No	Yes
2	GSM input (Audio from the GSM device)	Yes	No
3	Microphone	Yes	No
10..19	Extension module audio streams	Yes*	Yes*

*: Extension module streams may support Capture, Playback or both. See the RTCU M2M Platform SDK Reference Manual for more details.

Input:

id: INT (1..3) ***Default 1***
The ID of the route to control. This is not related to the Device ID.

src: INT
The capture device to read from .
See Device ID table.

dest: INT
The playback device to write to.
See Device ID table.

enable: BOOL ***default TRUE***
Set to TRUE to enable the route, set to FALSE to disable the route.

Returns: INT

0	- Operation was successful.
-1	- Invalid route ID
-2	- Route is in use.
-3	- Invalid device.
-4	- Device is already in use.
-5	- No available route could be found.
-6	- Failed to start route.
-7	- Device can not be used in this direction.

Declaration:

```
FUNCTION audioRoute : INT;
VAR_INPUT
   id     : INT := 1;
   src    : INT;
   dest   : INT;
   enable : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc    : INT;
END_VAR;
...
// Create route from GSM to Speaker
rc := audioRoute(id:=1, src := 2, dest := 0);
DebugFmt(message := "GSM => SPK: \1", v1:= rc);
```

```
// Create route from Microphone to GSM
rc := audioRoute(id:=2, src := 3, dest := 1);
DebugFmt(message := "MIC => GSM: \1", v1:= rc);
...
BEGIN
   ...
END;

END_PROGRAM;
```

4.2.4 bat: Battery

4.2.4.1 Bat: Battery Functions

The battery functions are used for monitoring the on-board battery and control of the battery charger.

- batChargerEnable — Enables/disables the charging off the internal battery pack.
- batIsCharging — Queries the status on fast (rapid) charging.
- batVoltageIsLow — Queries the status on the battery voltage.
- batModuleMounted — Queries whether a battery module is mounted.
- batPowerLevel — Returns the power level of the internal battery pack.

The battery consumption functions allow the program to interact with the battery power consumption circuit found on the RTCU SX1 series.
The power consumption circuit measures the current consumed from the battery with the following functions:

- batConsumption — Queries the current battery consumption.
- batConsumptionAvg — Queries the average battery consumption.
- batConsumptionClear — Resets the average and total battery consumption.
- batConsumptionEnable — Controls the battery current consumption measurement module.
- batConsumptionTotal — Queries the total battery consumption.

4.2.4.2 batChargerEnable (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

---

batChargerEnable enables/disables the charging of the on-board battery.
The charger enable/disable state is used until the device is restarted or batChargerEnable is called again.
The battery charger is enabled by default.

Input:

enable : BOOL (false/true)
TRUE: Enables the charging.
FALSE: Disables the charging.

Returns: INT

0 Denotes successful operation.
-1 Denotes that the function is not supported on the specific device or that the battery is missing.

Declaration:

```
FUNCTION batChargerEnable : INT;
VAR_INPUT
   enable : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   doCharge : BOOL; | enable/disable charger
END_VAR;

VAR_OUTPUT
   isEnabled : BOOL; | Charger is enabled
END_VAR;

PROGRAM test;
BEGIN
   IF (doCharge XOR isEnabled) THEN
      // Enable/Disable battery charging
      batChargerEnable(enable:=doCharge);
      isEnabled := doCharge;
   END_IF;
END;
END_PROGRAM;
```

4.2.4.3 batIsCharging (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

batIsCharging returns information about whether the on-board battery is being charged or not.

Input:
None

Returns: BOOL
True if the battery is being charged. False if not.

Declaration:
```
FUNCTION batIsCharging : BOOL;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   IF batModuleMounted() THEN
      IF batVoltageIsLow() THEN
         gsmSendSMS(phonenumber:="12345678", message:="Battery Voltage is
low");
      END_IF;
      IF NOT batIsCharging() THEN
         PowerDown(seconds:=600);
      END_IF;
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.4.4 batVoltageIsLow (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

batVoltageIsLow queries the status on the battery voltage. If the voltage is below a hardware-set threshold, a Low Battery indication will occur.

Input:
None

Returns: **BOOL**
True if the battery voltage is low. False if not.

Declaration:
```
FUNCTION batVoltageIsLow : BOOL;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   IF batModuleMounted() THEN
      IF batVoltageIsLow() THEN
         gsmSendSMS(phonenumber:="12345678", message:="Battery Voltage is
low");
      END_IF;
      IF NOT batIsCharging() THEN
         PowerDown(seconds:=600);
      END_IF;
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.4.5 batModuleMounted (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

batModuleMounted queries if a battery module is mounted on the RTCU device.

Input:
None

Returns: **BOOL**
True if a battery module is mounted. False if not.

Declaration:
```
FUNCTION batModuleMounted : BOOL;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   IF batModuleMounted() THEN
      IF batVoltageIsLow() THEN
         gsmSendSMS(phonenumber:="12345678", message:="Battery Voltage is
low");
      END_IF;
      IF NOT batIsCharging() THEN
         PowerDown(seconds:=600);
      END_IF;
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.4.6 batPowerLevel (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

batPowerLevel will return the power level of the backup battery.

Input:
None

Returns: SINT
-1 - Error.
0-5 - Power level, where 5 is maximum power.

Declaration:
```
FUNCTION batPowerLevel : SINT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   batlvl : SINT;
END_VAR;

BEGIN
   ...
   IF batModuleMounted() THEN
      batlvl := batPowerLevel();
      IF batlvl > -1 AND batlvl < 3 THEN
         DebugFmt(message:="Battery level=\1",v1:=batlvl);
      END_IF;
   END_IF;
   ...
END;
```

```
END_PROGRAM;
```

4.2.4.7 batConsumption (function)

Architecture: X32
Device support: SX1
Firmware version: 2.70

This function will return the battery current consumption measured in mA.
The battery current consumption is only measured when the current consumption measurement circuit is enabled by using the batConsumptionEnable function.

Input:
None.

Returns: DINT
The current that is being consumed from the battery in mA.

Declaration:
```
FUNCTION batConsumption : DINT;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   val : DINT;
END_VAR;

   // Start the current consumption measurement
   batConsumptionEnable(enable := ON);

BEGIN
   ...
   // Get the current consumption
   val := batConsumption();
   ...
END;
END_PROGRAM;
```

4.2.4.8 batConsumptionAvg (function)

Architecture: X32
Device support: SX1
Firmware version: 2.70

This function will return the average battery current consumption in mA measured since the last call to batConsumptionClear.

The average battery current consumption is only measured when the current consumption measurement circuit is enabled by using the batConsumptionEnable function.
The current consumption average is persistent even through system restarts and may be reset by calling batConsumptionClear,

Input:
None.

Returns: DINT
The average current that is being consumed from the battery in mA.

Declaration:
```
FUNCTION batConsumptionAvg : DINT;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   val : DINT;
END_VAR;

   // Start the current consumption measurement
   batConsumptionEnable(enable := ON);

BEGIN
   ...
   // Get the current consumption
   val := batConsumptionAvg();
   ...
END;
END_PROGRAM;
```

4.2.4.9 batConsumptionClear (function)

Architecture: X32
Device support: SX1
Firmware version: 2.70

This function will reset the average current consumption (see batConsumptionAvg) and the total current consumption (see batConsumptionTotal), so that both values will be set to zero.

Input:
None.

Returns:
None.

Declaration:
```
FUNCTION batConsumptionClear;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM example;

BEGIN
   ...
   // Clear the current consumption
   batConsumptionClear();
   ...
END;
END_PROGRAM;
```

4.2.4.10 batConsumptionEnable (function)

Architecture: X32
Device support: SX1
Firmware version: 2.70

This function controls whether the battery current consumption measurement circuit is enabled or disabled.
As the power consumption circuit itself uses power, it is recommended to disable it when not used. Default state is disabled.

The battery current consumption measurement is automatically suspended while in power saving modes such as pmWaitEvent but not while running at lower speeds.
The power consumption used when entering power saving modes such as pmWaitEvent are not included in the measurement.

Input:

enable : BOOL (false/true)
TRUE: Enables the current consumption measurement.
FALSE: Disables the current consumption measurement.

Returns:

None.

Declaration:

```
FUNCTION batConsumptionEnable;
VAR_INPUT
   enable : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;

   // Start the current consumption measurement
   batConsumptionEnable(enable := ON);

BEGIN
   ...
END;
END_PROGRAM;
```

4.2.4.11 batConsumptionTotal (function)

Architecture: X32
Device support: SX1
Firmware version: 2.70

This function will return the total battery current consumption in mAh accumulated since the last call to batConsumptionClear.

The total battery current consumption is only measured when the current consumption measurement circuit is enabled by using the batConsumptionEnable function.
The current consumption total is persistent even through system restarts and may be reset by calling batConsumptionClear,

Input:
None.

Returns: DINT
The total current that has been consumed from the battery in mAh.

Declaration:
```
FUNCTION batConsumptionTotal : DINT;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   val : DINT;
END_VAR;

   // Start the current consumption measurement
   batConsumptionEnable(enable := ON);

BEGIN
   ...
   // Get the current consumption
   val := batConsumptionTotal();
   ...
END;
END_PROGRAM;
```

4.2.5 board: Board Related Functions

4.2.5.1 Board: Board related functions

The board-related functions give access to a number of features that are offered by the RTCU platform. Some of these features might not be available on all RTCU devices. Please consult the technical manual of the actual RTCU device.

- boardSupplyVoltage — Returns the current supply voltage.
- boardSupplyType — Returns the type of supply the RTCU device is powered with.
- boardTemperature — Returns the current temperature.
- boardVersion — Returns the version of firmware.
- boardVersionX — Returns detailed information on the firmware versions in an NX32L based device.
- boardType — Returns the type of RTCU device.
- boardSerialNumber — Returns the serial number of the RTCU device.
- boardReset — Resets the RTCU device.
- boardGetProfile — Returns information about features that are available on the RTCU device.
- boardGetProfileX — Returns extended information about features that are available..
- boardWatchdog — Software watchdog.
- boardSetPassword — Sets the password for accessing the RTCU device.
- boardSetPasswordAlt — Sets the alternative password of the device.
- boardClearPassword — Clears the password of the device.
- boardSetFaultReset — Sets a timer for resetting a device after a fault.
- boardDCOut — Enable/disable external DC-power.
- boardDCOut2 — Enable/disable external DC-power 2.
- boardFaultLogGet — Gets fault log entries.
- boardFaultLogGetDebug — Read the debug information of a fault log entry.
- boardFaultLogClear — Clears the fault log.
- boardEnableS0 — Controls the S0 interface.
- boardSetOption — Enables options using an option key.
- boardRequestOption — Requests on-demand options from the Logic IO option server.
- boardGetStatistics — Query system statistics from the device.
- boardTimeSinceReset — Returns number of milliseconds since device start.
- boardSetAIMode — Switch the analog input mode between synchronous and asynchronous.
- boardSetAIResolution — Selects the resolution for the analog input to use.
- boardSetAOResolution — Selects the resolution for the analog output to use.
- boardSysSwitchGet — Query the state of a System Switch.
- boardSysSwitchSet — Control a System Switch.
- boardSetApplication — Select application file from project drive to run,
- boardSupplySetWakeup — Configures the system to wake from pmSuspend when the external power is applied/removed.
- boardDinSetWakeup — Configures the digital inputs as wake-up sources for pmSuspend.

4.2.5.2 boardSupplyVoltage (Function)

```
Architecture:       X32 / NX32 / NX32L
Device support:     All
Firmware version:   1.00 / 1.00.00
```

---

boardSupplyVoltage returns the measured voltage the RTCU device is supplied with.
NOTE: The actual supply measurement is done asynchronously, so it may take some seconds for a physical change to be reflected.

Input:
None

Returns: INT (0..400)
Supply voltage in 0.1 Volt increments. 12 V supply voltage will be returned as 120.

Declaration:
```
FUNCTION boardSupplyVoltage : INT;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   IF boardSupplyVoltage() > 120 THEN
      // Supply voltage is above 12 Volt
      ...
   END_IF;
END;

END_PROGRAM;
```

4.2.5.3 boardSupplyType (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

boardSupplyType returns information about the type of power (battery, DC, or AC) the RTCU is currently operating on.

Input:
None

Returns: INT (1..3)
Type of supply:
1: Operating on internal battery.
2: Operating on external DC power.
3: Operating on external AC power.

Declaration:
```
FUNCTION boardSupplyType : INT;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
```

```
   ...
   IF boardSupplyType() = 1 THEN
      // Supply voltage is battery
      ...
   END_IF;
   IF boardSupplyType() = 2 THEN
      // Supply voltage is External DC
      ...
   END_IF;
END;

END_PROGRAM;
```

4.2.5.4 boardTemperature (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

boardTemperature returns the measured temperature inside the RTCU device in degrees Celsius.

Input:

None

Returns: INT

-2000..12500	Temperature in 0.01 deg Celsius. 22.50 degrees will be returned as 2250, and -17.00 degrees will be returned as -1700.
-9999	Error - the temperature read failed.

Declaration:

```
FUNCTION boardTemperature : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   IF boardTemperature() > 3500 THEN
      // Internal temperature of the RTCU device is above 35 degrees
      ...
   ELSIF boardTemperature() < -1000 THEN
      // Internal temperature of the RTCU device is below -10 degrees
      ...
   END_IF;
END;

END_PROGRAM;
```

4.2.5.5 boardVersion (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

boardVersion returns the version number of the firmware running on the RTCU device.

Input:
None

Returns: INT (0..32767)
Version number of the firmware on the RTCU device. Version will be scaled by 100 so that version 3.00 will be returned as 300.

Declaration:
```
FUNCTION boardVersion : INT;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Set the current write position to 1,2 (leftmost, second row)
   displayXY(x := 1, y := 2);
   // Print the version number of the firmware at the current write position
   displayNumber(number := boardVersion());
   ...
END;

END_PROGRAM;
```

4.2.5.6 boardVersionX (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.12.00

The boardVersionX function returns detailed information about all three types of firmware in an NX32L based device.

Input:
type : SINT;
The firmware type to return the version of.
_BV_SYSTEM System-firmware version
_BV_RUNTIME Runtime-firmware version
_BV_MONITOR Monitor-firmware version
_BV_CO_P_BOOT Co-processor bootloader version (LX only).
_BV_CO_P_MAIN Co-processor firmware version (LX only).

Output:
major : INT;
The major part of the selected firmware version.

minor : INT;
The minor part of the selected firmware version.

build : INT;
The build part of the selected firmware version.

Returns: INT

1 - Success
0 - Not supported.
-1 - Illegal firmware type

Declaration:

```
FUNCTION ALIGN boardVersionX : INT;
VAR_INPUT
   type    : SINT;
   major   : ACCESS INT;
   minor   : ACCESS INT;
   build   : ACCESS INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   major   : INT;
   minor   : INT;
   build   : INT;
END_VAR;

boardVersionX(type := _BV_SYSTEM, major := major, minor := minor, build :=
build);
DebugFmt(message := "System:  \1.\2.\3", v1 := major, v2 := minor, v3 :=
build);
boardVersionX(type := _BV_RUNTIME, major := major, minor := minor, build :=
build);
DebugFmt(message := "Runtime: \1.\2.\3", v1 := major, v2 := minor, v3 :=
build);

BEGIN
END;
END_PROGRAM;
```

4.2.5.7 boardType (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

---

boardType returns the type of RTCU device.

Input:

None

Returns: INT

Type of device the program is executing on:

9: RTCU A9/A9i.
10: RTCU M10.
11: RTCU M11/M11i.
102: RTCU MX2/MX2i.
103: RTCU DX4i mk2.
104: RTCU DX4/DX4i.

105: RTCU CX1/CX1i/CX1 mk2.
106: RTCU SX1.
109: RTCU AX9/AX9i.
122: RTCU MX2 turbo / MX2 encore / MX2 warp.
129: RTCU AX9 turbo / AX9 encore.
192: RTCU MX2 SOM module.
202: RTCU NX-200.
204: RTCU NX-400.
209: RTCU NX-900 / NX-910
302: RTCU LX2 series
304: RTCU LX4 series
305: RTCU LX5 series

Declaration:

```
FUNCTION boardType : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Set the current write position to 1,2 (leftmost, second row)
   displayXY(x := 1, y := 2);
   // Print the type of board at the current write position
   displayNumber(number := boardType());
   ...
END;

END_PROGRAM;
```

4.2.5.8 boardSerialNumber (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

boardSerialNumber returns the serial number of the RTCU device.

Input:
None

Returns: **DINT**
Serial number of the RTCU device.

Declaration:

```
FUNCTION boardSerialNumber : DINT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
```

```
VAR
   sernum : DINT;
END_VAR;

BEGIN
   ...
   // Fetch the serial number of the RTCU device
   sernum := boardSerialNumber();
   ...
END;

END_PROGRAM;
```

4.2.5.9 boardReset (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

---

boardReset will reset the RTCU device.

Input:

level : SINT *(default 0)*

NX32L only: When this parameter is 1 a complete Linux level reboot of the system will be performed.

Returns:

None

Declaration:

```
FUNCTION boardReset;
VAR_INPUT
   level : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Make the RTCU device reset
   boardReset();
   ...
END;

END_PROGRAM;
```

4.2.5.10 boardGetProfile (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

---

boardGetProfile returns information about the the facilities available on the RTCU device.

Input:

index : SINT (0..18)
This identifies which parameter is being requested
Possible values are:

0	- Number of indexes present in profile (currently 18).
1	- Number of digital inputs.
2	- Number of digital outputs.
3	- Number of analog inputs.
4	- Number of analog outputs.
5	- Number of dip-switches.
6	- Number of user-LEDs.
7	- LCD mounted?
8	- RTC mounted?
9	- GSM mounted?
10	- Programmable?
11	- Built-in GPS receiver?
12	- TCP/IP enabled?
13	- Number of serial ports (1/2/3).
14	- On-board temperature sensor present?
15	- Extended FLASH persistent memory present?
16	- Is the device an 'i'-revision?
17	- Execution architecture. 1=X32 / 2=NX32 / 3=NX32L
18	- Number of flex inputs.

Returns: DINT

The requested parameter
In case of TRUE/FALSE conditions, 0 (zero) will indicate FALSE and 1 (one) will indicate TRUE.

Declaration:

```
FUNCTION boardGetProfile : DINT;
VAR_INPUT
   index : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   num_do : DINT;
END_VAR;

BEGIN
   ...
   // Ask how many digital outputs are present
   num_do := boardGetProfile(index:=2);
   ...
END;

END_PROGRAM;
```

4.2.5.11 boardGetProfileX (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

boardGetProfileX returns extended device information.
Also see boardGetProfile() for basic device information.

Input:

index : SINT (0..27)
This identifies which parameter is being requested
Possible values are:

0 - Number of indexes present in profile (currently 26).
1 - Cellular type:
  0: None.
  1: GSM Dual Band.
  2: GSM Quad Band.
  3: GSM Quad Band / Edge.
  4: UMTS EMEA.
  5: UMTS America.
  6: UMTS Worldwide.
  7: LTE Cat. M1/NB1.
  8: LTE Cat. 4 Regional.
  9: LTE Cat.4 Worldwide.
2 - GPS type (0=None, 1=GPS, 2=Super GPS, 3=GNSS, 4=PPP).
3 - Battery type (0=None, 1=Standard capacity, 2=High capacity).
4 - Non-volatile memory type (1=NVRAM, 2=VFRAM)?
5 - Headset present?
6 - DTMF/Voice present?
7 - SD-CARD present?
8 - Number of CAN buses.
9 - Not used.
10 - Buzzer present?
11 - Onewire bus present?
12 - Optional RS485 present?
13 - Navigation present?
14 - Full execution speed?
15 - X32 enhanced memory?
16 - RF present?
17 - Standard RS485 present?
18 - External voice playback supported?
19 - Number of FLASH entries.
20 - Number of FRAM entries.
21 - Number of extended FLASH entries.
22 - Execution Environment (0=RTCU Device, 2=RTCU Emulator)
23 - Number of USB host ports.
24 - Bluetooth present? (only NX32L)
25 - Motion Sensors present? (only NX32L)
  Bitwise OR of the following sensors:
    1 : Accelerometer
    2 : Gyroscope
    4 : Magnetometer
26 - NX-400 - optionally mounted Radiocrafts module:
  0: Not present.
  1: RC1140-MBUS3
  2: RC1160-MBUS3
  3: RC1170-MBUS3
  4: RC1180-MBUS3
  5: RC1682-SIG
  6: RC1682-SSM
  7: RC1692HP-SIG
  8: RC1692HP-SSM
  9: RC1180-KNX2
27 - GNSS type (only NX32L):

0: None.
1: Quectel L26
2: uBlox NEO-M8L
3: Cellular GNSS
4: Quectel LC29H.
5: uBlox NEO-M9V

Returns: **DINT**
The requested parameter.
In case of TRUE/FALSE conditions, 0 (zero) will indicate FALSE and 1 (one) will indicate TRUE.

Declaration:
```
FUNCTION boardGetProfileX : DINT;
VAR_INPUT
   index : SINT;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   num_do : DINT;
END_VAR;

BEGIN
   ...
   // Ask how many digital outputs are present
   num_do := boardGetProfileX(index:=2);
   ...
END;

END_PROGRAM;
```

4.2.5.12 boardWatchdog (Function)

Architecture: **X32 / NX32 / NX32L**
Device support: **All**
Firmware version: **1.00 / 1.00.00**

---

boardWatchdog offers a software watchdog that can be used for detecting and recovering from problems in the running application.
The watchdog will monitor that it has been called within a certain time frame, and if this does not occur, the device will be restarted.

To ensure recovery in case of any problems in the application and/or the firmware, it is recommended to integrate the use of boardWatchdog() into any application. This will ensure that the device will reset if any problems should occur.

Input:
timeout : INT

The use of the timeout parameter is:
timeout > 0:
Start watchdog with the timeout specified in number of seconds.
timeout = 0:
Stop watchdog.

timeout = -1:

Reset watchdog.
Must be called with this parameter periodically and within the timeout period, or the device will restart.

Returns:

None

Declaration:

```
FUNCTION boardWatchdog;
VAR_INPUT
   timeout : INT := -1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

   // Start Watchdog. This will restart the device if boardWatchdog() is not
called within 10 seconds.
   boardWatchdog(timeout:=10);

BEGIN
   ...
   // Reset Watchdog
   boardWatchdog();
   ...
END;

END_PROGRAM;
```

4.2.5.13 boardSetPassword (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

---

boardSetPassword sets the primary password that is used by the RTCU IDE (or any other RACP1/RACP2) client that will authenticate itself to the device.
If the primary password is empty, the RTCU device will not require a password for authentication.

It is also possible to use an alternative password in addition to the primary password with the boardSetPasswordAlt function
The primary password can be cleared by calling boardClearPassword.

Input:

curpsw : STRING (*Max 20 characters*)
The current password. Must be correct to set new password.

newpsw : STRING (*Max 20 characters*)
The new password.
An empty string will disable the password of the device.

Returns: BOOL

TRUE: New password has been set.
FALSE: Password is not set. Wrong or invalid password specified.

Declaration:

```
FUNCTION boardSetPassword : BOOL;
VAR_INPUT
   curpsw : STRING;
   newpsw : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Change password
boardSetPassword(curpsw:="12345",newpsw:="54321");

END_PROGRAM;
```

4.2.5.14 boardSetPasswordAlt (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.05 / 1.00.00

boardSetPasswordAlt sets the alternative password that is used by the RTCU IDE (or any other RACP1/RACP2) client that will authenticate itself to the device.

The alternative password is only enabled when a primary password is set, see boardSetPassword for more information.

Input:

curpsw : STRING (*Max 20 characters*)
The current password. Must be correct to set new password.

newpsw : STRING (*Max 20 characters*)
The new password.

Returns: BOOL

TRUE: New password has been set.
FALSE: Password is not set. Wrong or invalid password specified.

Declaration:

```
FUNCTION boardSetPasswordAlt : BOOL;
VAR_INPUT
   curpsw : STRING;
   newpsw : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Change password
boardSetPasswordAlt(curpsw:="12345",newpsw:="54321");

END_PROGRAM;
```

4.2.5.15 boardClearPassword (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.05 / 1.00.00

Clears the primary password that is used by the RTCU IDE (or any other RACP1/RACP2) client that will authenticate itself to the device.

See boardSetPassword for more information.

Input:
None

Returns: BOOL
TRUE: Password has been cleared.
FALSE: Password is not cleared.

Declaration:
```
FUNCTION boardClearPassword : BOOL;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;

// Clear password
boardClearPassword();

END_PROGRAM;
```

4.2.5.16 boardSetFaultReset (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

boardSetFaultReset will set the length of time (in seconds) to wait after a fault before resetting the device. A time of 0 (zero) disables reset after fault.
Note: the minimum delay that is supported is 10 seconds.

Caution should be exercised when using this function as the devices will continue to reset when a fault occurs. This can make it difficult to connect to the device remotely for diagnostic etc.

When a PIN code fault (incorrect PIN code) occurs, the auto-reset will not occur.

Input:
delay : INT
The time the device waits before resetting when a fault is experienced.

Returns:
None

Declaration:

```
FUNCTION boardSetFaultReset;
VAR_INPUT
   delay : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Enable reset after a Fault
boardSetFaultReset(delay:=10);

END_PROGRAM;
```

4.2.5.17 boardDCOut (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, DX4, AX9 pro, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, NX-910, LX2
Firmware version: 1.00 / 1.00.00

This function is used to enable/disable the DC-OUT external power supply that is available on some devices.
For technical specification on the DC-OUT external power supply, please consult the technical manual for the device in question.

Input:

Enable : BOOL
Enables/disables DC-OUT. Default state is: *disabled.*

Returns:

None

Declaration:

```
FUNCTION boardDCOut;
VAR_INPUT
   enable : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Enable DC-OUT (external power-supply)
   boardDCout(enable:=ON);
   ...
END;

END_PROGRAM;
```

4.2.5.18 boardDCOut2 (Function)

Architecture: X32 / NX32 / NX32L
Device support: AX9 pro, AX9 turbo/encore, NX-900, NX-910
Firmware version: 2.10 / 1.51.00

This function is used to enable/disable the DC-OUT 2 external power supply that is available on some devices.
For technical specification on the DC-OUT 2 external power supply, please consult the technical manual of the respective device.

Input:

Enable : BOOL
Enables/disables DC-OUT 2. Default state is: *disabled.*

Returns:

None

Declaration:

```
FUNCTION boardDCOut2;
VAR_INPUT
   enable : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Enable DC-OUT 2
   boardDCout2(enable:=ON);
   ...
END;

END_PROGRAM;
```

4.2.5.19 boardFaultLogGet (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.05 / 1.00.00

The boardFaultLogGet function block is used to read the fault log from the VPL application.
The faults returned in this function block can also be read from the Fault log dialog.

The fault entries are sorted with the last fault first.

Because the fault code is a value between 1 and 255, and the SINT data type can only hold values between -128 and 127, the fault codes over 127 are returned as a negative number.
To convert a fault code returned as a negative number back to the correct value, you should use the following method:
<int> := INT(fault[n] AND 16#FF);

A debug feature allows information about the specific location of a fault to be retrieved. Please refer to boardFaultLogGetDebug for additional information.

See the Project settings dialog in the RTCU IDE for more information on the debug feature.

For a list and explanation of the fault codes, please refer to the Troubleshooting section, Fault codes.

Input:
None

Output:
count : INT;
Number of fault log entries (0..32).

time[n] : DINT;
Linsec for fault timestamp.

fault[n] : SINT;
Fault code.

Declaration:
```
FUNCTION_BLOCK boardFaultLogGet;
VAR_OUTPUT
   count : INT;
   time  : ARRAY[1..32] OF DINT;
   fault : ARRAY[1..32] OF SINT;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   read  : boardFaultLogGet;
   clock : clockLinsecToTime;
   str   : STRING;
   i     : INT;
END_VAR;

// Get list of faults
read();
// Iterate any faults
FOR i := 1 TO read.count DO
   clock(linsec := read.time[i]);
   DebugFmt(message := "fault= \1", v1 := INT(read.fault[i] AND 16#FF));
   str := strFormat(format := "\1.\2.\3", v1 := clock.day, v2 := clock.month,
v3 := clock.year);
   DebugFmt(message := "time= \1:\2:\3, " + str, v1 := clock.hour, v2 :=
clock.minute, v3 := clock.second);
END_FOR;

BEGIN
  ...
END;
END_PROGRAM;
```

4.2.5.20 boardFaultLogGetDebug (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 3.10 / 1.00.00

The boardFaultLogGetDebug is used to read the debug information of a fault log entry from the VPL application.
The function is designed to be used together with the boardFaultLogGet function block. Where boardFaultLogGet reads the fault log and boardFaultLogGetDebug reads the additional debug information if present.

The debug information returned in this function can also be read from the RTCU IDE.

Input:

index : INT;
The index of the fault entry to read.

Output:

filename : STRING;
The file name of the file where the fault occurred.

line : DINT;
The approximate line number in the file where the fault occurred.

threadid : INT
The ID of thread which executed the last know line of code.
This ID corresponds to the ID returned from thGetID().

Returns: BOOL

TRUE: debug information present for entry
FALSE: no debug infromation present for entry

Declaration:

```
FUNCTION boardFaultLogGetDebug;
VAR_INPUT
   index    : INT;
   filename : ACCESS STRING;
   line     : ACCESS DINT;
   id       : ACCESS INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   read  : boardFaultLogGet;
   clock : clockLinsecToTime;
   str   : STRING;
   i     : INT;
   name  : STRING;
   line  : DINT;
   thread: INT;
END_VAR;

// Get list of faults
read();
// Iterate any faults
FOR i := 1 TO read.count DO
   clock(linsec := read.time[i]);
   DebugFmt(message := "fault= \1", v1 := INT(read.fault[i] AND 16#FF));
   str := strFormat(format := "\1.\2.\3", v1 := clock.day, v2 := clock.month,
v3 := clock.year);
```

```
   DebugFmt(message := "time= \1:\2:\3, " + str, v1 := clock.hour, v2 :=
clock.minute, v3 := clock.second);
   // Get the debug information from the fault log
   IF
boardFaultLogGetDebug(index:=i,filename:=name,line:=line,threadid:=thread)
THEN
      DebugFmt(message := "file= " + name);
      DebugFmt(message := "line= \4", v4 := line);
      DebugFmt(message := "thread= \1", v1 := thread);
   END_IF;
END_FOR;

BEGIN
  ...
END;
END_PROGRAM;
```

4.2.5.21 boardFaultLogClear (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.05 / 1.00.00

The boardFaultLogClear is used to clear the Fault log in the device.

The fault log can also be cleared from the RTCU IDE.

Input:

None

Returns:

None

Declaration:

```
FUNCTION boardFaultLogClear;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
   ...
   // Clear the Fault log
   boardFaultLogClear();
   ...
BEGIN
END;

END_PROGRAM;
```

4.2.5.22 boardEnableS0 (Function)

Architecture: X32 / NX32 / NX32L
Device support: DX4, AX9, NX-400, NX-900, NX-910, LX4
Firmware version: 1.30 / 1.00.00

The boardEnableS0 function is used to control the IEC62053-31 compliant S0 interface that is available on some RTCU devices. Please refer to the Technical Manual for information about the necessary settings of the hardware jumpers to use the S0 interface.

By enabling the S0 interface, the power will be supplied to the necessary electronic circuits which will increase the power consumption of the device.

The S0 interface is disabled by default and must specifically be enabled by the application.

Input:

enable : BOOL
Enables or disables the S0 interface.

Returns:

0 - Success.
1 - Not supported.

Declaration:

```
FUNCTION boardEnableS0 : INT;
VAR_INPUT
   enable : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

   // Enable S0 interface
   boardEnableS0(enable := ON);

BEGIN
   ...
END;
END_PROGRAM;
```

4.2.5.23 boardSetOption (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 2.50 / 1.00.00

The boardSetOption function makes it possible to enable certain options in the RTCU device.

Note that the device must be reset before the changes take effect (for example by using boardReset).

Input:

optionkey : STRING
The option key that is used to enable the options.
The option key is an 11 character long text that can contain both upper and lower case characters.

Returns:

0 - Success.

2 - Illegal option key.
3 - Unknown option.

Declaration:

```
FUNCTION boardSetOption : INT;
VAR_INPUT
   optionkey : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

   boardSetOption(optionkey := "5UtiUEhx30i");

BEGIN
   ...
END;
END_PROGRAM;
```

4.2.5.24 boardRequestOption (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 2.64 / 1.00.00

The boardRequestOption function makes it possible to request options from the Option Server directly from the RTCU device. The functionality is equal to the RTCU IDE Request Option functionality.

When an option is requested, the option server at Logic IO is contacted through the internet by using the supplied account credentials. If there are sufficient credits for the requested option(s), the transaction will be made and the options activated on the device.
To use the request option functionality, it is therefore necessary to have an account with a positive credit established at Logic IO with a given user name/password.

Note that the device must be reset before the changes take effect (for example by using boardReset).

Note: using this function will deduct the credit on the account used and constitutes a non-refundable purchase.

Input:

user : STRING
The user name used to connect to the option server. Case sensitive.

pass : STRING
The password used to connect to the option server. Case sensitive.

options : STRING
The list of options to apply. The options are case sensitive and multiple options must be separated with spaces.
To find an up-to-date list of the possible options and prices for a specific target, use the IDs from Request Options in the RTCU IDE.

The following devices currently supports the options listed below:

CX1 series

CX1 flex / CX1i flex

COM	RS232, FMI, 1-wire (CX1i flex only).
IO	I/O option. See technical manual for details.

CX1 warp

COM	RS232, FMI, 1-wire
IO	I/O option. See technical manual for details.
RF	RF interface.
FLASH	3MB (XF3) extended flash, SD-CARD

CX1 warp-c

COM	CAN
IO	I/O option. See technical manual for details.
RF	RF interface.
FLASH	3MB (XF3) extended flash, SD Card

DX4 Series

DX4 flex / DX4i warp

COM	RS485, MODBUS and 1-wire option.
DIO	Digital I/O option. See technical manual for details.
AIO	Analog I/O option. See technical manual for details.

MX2 Series

MX2 warp

CAN	CAN option.
COM	RS485 and 2nd RS232 (RJ45).
HS	Headset.
ISIM	Internal SIM card reader

MX2 encore

CAN	CAN option.
RS485	RS485 option.
ISIM	Internal SIM card reader

MX2 eco+

FMI	Fleet Management Interface

AX9 Series

AX9 pro / AX9 pro-x / AX9i pro / AX9 turbo

FMI	Fleet Management Interface

AX9 encore

RS485	RS485 option.

Returns:

>=0 - The total cost of the options.
-1 - Error connecting to server.
-2 - Illegal argument.
-3 - Server uses an unsupported protocol version.
-4 - Failed to log in.
-5 - Not enough credits to complete order.
-6 - Invalid option.

-7 - Unspecified error when trying to order.
-8 - Error applying options.
-9 - One or more options are restricted and must be approved by Logic IO.

Declaration:

```
FUNCTION boardRequestOption : INT;
VAR_INPUT
   user    : STRING;
   pass    : STRING;
   options : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc:INT;
END_VAR;
   ...
   ...
   IF boardGetProfileX(index := 13) = 0 THEN
      gsmPower(power:=ON);
      gprsOpen();
      WHILE NOT gprsConnected() DO
         Sleep(delay:=2500);
      END_WHILE;
      rc:= boardRequestOption(user:="user", pass:="pass",options:="FMI");
      IF rc >= 0 THEN
         DebugMsg(message := "Options enabled");
         // Make the RTCU device reset
         boardReset();
      END_IF;
   END_IF;
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.5.25 boardGetStatistics (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 2.70 / 1.00.00

boardGetStatistics is used to query the system statistics available in the RTCU device.
The device statistics contains lifetime information initialized during the production of the device and cannot be reset.
It has the same effect as reading the statistics via Device: Information: Statistics.

Input:

index : INT (0..17)
This identifies which parameter are being requested.
Possible values are:

0 - Number of indexes present in statistics (currently 17).
1 - Number of times the device has rebooted.
2 - Highest temperature measured in the device - represented in 0.01 deg Celsius. 22.50 degrees will be returned as 2250, and -17.00 degrees will be returned as -1700.

3 - Average temperature measured in the device - represented in 0.01 deg Celsius.
4 - Lowest temperature measured in the device - represented in 0.01 deg Celsius.
5 - Number of minutes the device has been operating.
6 - Number of minutes the device has been operating from backup battery.
7 - Number of minutes the device has been charging the backup battery.
8 - Number of times the device has tried to connect to RTCU Communication Hub.
9 - Number of incoming transactions from the RTCU Communication Hub.
10 - Number of outgoing transactions to the RTCU Communication Hub.
11 - Number of keep-alive transactions the device has sent to the RTCU Communication Hub.
12 - Number of times the device successfully connected to a mobile network.
13 - Number of bytes sent through the mobile network connection.
14 - Number of bytes received from the mobile network connection.
15 - Number of times the GPS module has reported "No fix".
16 - Number of times the GPS module has reported "2D fix".
17 - Number of times the GPS module has reported "3D fix".

Returns: **DINT**
The requested parameter.

Declaration:

```
FUNCTION boardGetStatistics : DINT;
VAR_INPUT
   index : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   reboots : DINT;
END_VAR;

BEGIN
   ...
   // Ask how many times the device have rebooted
   reboots := boardGetStatistics(index := 1);
   ...
END;

END_PROGRAM;
```

4.2.5.26 boardTimeSinceReset (Function)

Architecture: **X32 / NX32 / NX32L**
Device support: **All**
Firmware version: **3.10 / 1.00.00**

---

boardTimeSinceReset will return the number of milliseconds since the RTCU device was reset.

Please note that the value will wrap around, going negative after approximately 25 days.

Input:
None

Returns:

The time since RTCU device was reset in milliseconds.

Declaration:

```
FUNCTION boardTimeSinceReset : DINT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   a : DINT;
END_VAR;

BEGIN
   ...
   // Get time since the RTCU device was reset
   a := boardTimeSinceReset();
   ...
END;

END_PROGRAM;
```

4.2.5.27 boardSetAIMode (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 3.15 / 1.00.00

The analog input sampling logic can work in either asynchronous or synchronous mode.

In **asynchronous** mode the sampling occurs in the background, so that execution of BEGIN/END/UPDATEIO is not blocked waiting for a new sample. This mode means that a new sample is not always present in each BEGIN/END iteration.

In **synchronous** mode the sampling occurs at the time of the call to BEGIN/END/UPDATEIO, and means that in each new iteration of a new sample is guaranteed.

With the synchronous mode a higher sampling rate and better control of the sampling can be achieved.
The default mode is asynchronous, which for most applications will work as expected.

Input:

sync : bool *(default false)*
Selects synchronous or asynchronous mode of analog input sampling.

Returns: INT

0 Successful.
1 Not supported.

Declaration:

```
FUNCTION boardSetAIMode : INT;
VAR_INPUT
   sync : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
// Selects asynchronous mode for high speed sampling:
boardSetAIMode(sync := TRUE);

BEGIN
END;
END_PROGRAM;
```

4.2.5.28 boardSetAIResolution (Function)

Architecture: X32/ NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9i series, DX4i pro, AX9 turbo/encore, NX-200, NX-400, NX-900, NX-910, LX2, LX4, LX5
Firmware version: 4.00 / 1.00.00

The analog input sampling logic can work at multiple resolutions to fit different needs.
The default resolution is 10 bit.
When increasing the resolution from 10 to 12 bit, the range of the analog input values change from 0..1023 to 0..4095.

Input:

res : int (10,12)
Selects the resolution for the analog input sampling.

Returns: INT

0 Successful.
1 Not supported.

Declaration:

```
FUNCTION boardSetAIResolution : INT;
VAR_INPUT
   res : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
// Selects 12 bit sampling:
boardSetAIResolution(res := 12);

BEGIN
END;
END_PROGRAM;
```

4.2.5.29 boardSetAOResolution (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

The analog output logic can work at multiple resolutions to fit different needs.

The default resolution is 10 bit.
When increasing the resolution from 10 to 12 bit, the range of the analog output values change from 0..1023 to 0..4095.

Input:

res : int (10,12)
Selects the resolution for the analog output.

Returns: INT

0 Successful.
1 Not supported.

Declaration:

```
FUNCTION boardSetAOResolution : INT;
VAR_INPUT
   res : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
// Selects 12 bit output:
boardSetAOResolution(res := 12);

BEGIN
END;
END_PROGRAM;
```

4.2.5.30 boardSysSwitchGet (Function)

Architecture: NX32L
Device support: NX-200, NX-400
Firmware version: 1.00.00

Query the state of a system switch.

A system switch is a software controlled switch, which is an alternative to hardware a jumper. The state of the switch is persistent.

The state of the system switch can be changed with the boardSysSwitchSet function.

Input:

id : DINT;
The id of the system switch to query.
_SSW_C CAN Termination
AN_1_TE
RM
_SSW_C CAN Write enable
AN_1_W
R
_SSW_C CAN2 Termination
AN_2_TE
RM
_SSW_C CAN2 Write enable
AN_2_W

R
_SSW_R RS485_1 Termination
S485_1_
TERM
_SSW_R RS485_2 Termination
S485_2_
TERM

Output:

state : SINT;
The state of the system switch.
1 On
0 Off

Returns: BOOL

1 Successful.
0 Not supported.
-1 The system switch do not exist

Declaration:

```
FUNCTION boardSysSwitchGet : INT;
VAR_INPUT
   id    : DINT;
   state : ACCESS SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
PROGRAM example;
VAR
   state : SINT;
END_VAR;
// Check CAN write support
boardSysSwitchGet(id:=_SSW_CAN_1_WR, state:=state);
IF state = 1 THEN
   DebugMsg(message := "CAN write is supported");
ELSE
   DebugMsg(message := "CAN write is not supported");
END_IF;
BEGIN
   . . .
END;
END_PROGRAM;
```

4.2.5.31 boardSysSwitchSet (Function)

Architecture: NX32L
Device support: NX-200, NX-400
Firmware version: 1.00.00

Set the state of a system switch.
The state of the system switch can be queried with the boardSysSwitchGet function.

The state is persistent and will be kept over reset.

Input:

id : DINT;

The id of the system switch to set.
_SSW_C CAN Termination (NX-200 / NX-400)
AN_1_TE
RM
_SSW_C CAN Write enable (NX-400)
AN_1_W
R
_SSW_C CAN2 Termination (NX-200)
AN_2_TE
RM
_SSW_R RS485_1 Termination (NX-200 / NX-400)
S485_1_
TERM
_SSW_R RS485_2 Termination (NX-400)
S485_2_
TERM

state : SINT;
The state of the system switch.
1 On
0 Off

Returns: BOOL
1 Successful.
0 Not supported.
-1 The system switch do not exist
-2 Illegal state

Declaration:
```
FUNCTION boardSysSwitchSet : INT;
VAR_INPUT
   id    : DINT;
   state : SINT;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc
PROGRAM example;
// Enable CAN write support
boardSysSwitchSet(id:=_SSW_CAN_1_WR, state:=1);
BEGIN
   . . .
END;
END_PROGRAM;
```

4.2.5.32 boardSetApplication (Function)

Architecture: NX32 / NX32L
Device support: All
Firmware version: 4.60 / 1.10.00

This function sets an alternative application to run when the device boots.
The changes will take effect at device reset. (e.g. by using boardReset).

The application must be a valid VSX file and can be placed on the Project Drive (P:). On LX/NX devices the VSX file can also be placed on the internal drive (B:) or on the SD-CARD drive (A:).

The VSX file must be accessible and valid at the time of the call to boardSetApplication.

When the device boots it will look for the alternative application file and if found valid it will be executed instead of the default application on the Project Drive.
Calling boardSetApplication with an empty filename or transferring a new project to the device will reset to use the default application.

Input:

filename : STRING
The application file name. This can be a fully qualified path such as "A:\myapp\test.vsx". When no drive specification is present, it will implicitly refer to the Project Drive.
If the filename is empty, the default application will be selected.

Returns: INT

1	Successful.
0	Not supported.
-1	Not a valid application filename.
-2	Application could not be found or is invalid.
-3	Invalid media.

Declaration:

```
FUNCTION boardSetApplication : INT;
VAR_INPUT
   filename : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Change application
boardSetApplication(filename := "example.vsx");
boardReset();

END_PROGRAM;
```

4.2.5.33 boardSupplySetWakeup (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.36.00

This function is used to configure the system to wake from suspend when the external power is applied or removed.
It is similar to using the PowerFail and PowerApply parameters on pmWaitEvent.

pmSuspend return codes for this wake-up source:

Error	Type	ID	Value	Description
True	1	3	-11	External supply is disabled using pmExternalPower.
False	1	3	1	Wake-up on power apply
False	1	3	0	Wake-up on power fail.

Input:

Enable : BOOL (default: TRUE)
Set to true to enable the wake-up, set to false to disable it.

Fail : BOOL (default: TRUE)
If set to true, the device will wake when the external power is removed.

Apply : BOOL (default: FALSE)
If set to true, the device will wake when the external power is connected.

Returns:

1 - The system has been configured to wake.
0 - This function is not supported.
-2 - Neither fail nor apply are enabled.

Declaration:

```
FUNCTION ALIGN boardSupplySetWakeup:INT;
VAR_INPUT
   enable: BOOL := TRUE;
   fail  : BOOL := TRUE;
   apply : BOOL := FALSE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Enable wake on power apply
   boardSupplySetWakeup(apply := TRUE);
   ...
END;

END_PROGRAM;
```

4.2.5.34 boardDinSetWakeup (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.36.00

This function is used to configure the system to wake from suspend when a digital input changes state.
It is similar to using the DI1-4 and DI1-5Falling parameters on pmWaitEvent.

There may be situations where it is possible to enable both the rising and falling edges, triggering a wake-up on any change. The return code can be used to determine if this is the case.
Under some circumstances, the return code from pmSuspend may show the edge that triggered the wake-up.

pmSuspend return codes for this wake-up source:

Error	Type	Value	Description
True	2	-10	S0 is enabled but there is no external power.
False	2	1	Wake-up on a rising edge

False	2	0	Wake-up on a falling edge.
False	2	-1	Wake-up on an edge.

The ID variable is set to the number of the digital input(1..8).

Input:

Enable : BOOL (default: TRUE)
Set to true to enable the wake-up, set to false to disable it.

Pin : SINT (1..8)
The digital input to enable wake up on.

Rising : BOOL (default: TRUE)
If set to true, the device will wake on the rising edge of the input.

Falling : BOOL (default: FALSE)
If set to true, the device will wake on the falling edge of the input.

Returns:

1 - The system has been configured to wake.
0 - This function is not supported.
-1 - The input is not present
-2 - The input does not support wake on both edges
-3 - The input is not a supported wake source.

Declaration:

```
FUNCTION ALIGN boardDinSetWakeup:INT;
VAR_INPUT
   enable  : BOOL := TRUE;
   pin     : SINT;
   rising  : BOOL := TRUE;
   falling : BOOL := FALSE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Enable wake on DIN 1 rising edge
   boardDinSetWakeup(pin := 1, rising := TRUE);
   ...
END;

END_PROGRAM;
```

4.2.6 ble: Bluetooth Low Energy (LX)

4.2.6.1 ble: Bluetooth Low Energy (LX)

This Bluetooth library is used to communicate with Bluetooth Low Energy devices.
This API is only supported on LX devices with BLE support. For BLE support on NX devices, see bt: Bluetooth functions (NX32L).

The examples BLE Scanner and BLE Beacon show how this API can be used.

.

Common Bluetooth Low Energy management

- blePower — Controls power to the Bluetooth interface.
- bleObserverStart — Starts listening for advertising data.
- bleObserverStop — Stops listening for advertising data.
- bleRegisterAdvData — Register a callback for handling advertising data.
- bleRegisterAdvRaw — Register a callback for handling raw advertising data.
- bleAcceptFilterAdd — Adds a device to the accept filter.
- bleAcceptFilterRemove — Removes a device from the accept filter.
- bleAcceptFilterClear — Removes all devices from the accept filter.

Device connection handling

- bleConnect — Begins connecting to a device.
- bleDisconnect — Disconnect from a device
- bleDeviceStatus — Get the status of the connection to a device.
- bleDeviceMacGet — Get the MAC address of a device.

GATT Service and characteristics

- bleServiceCacheUpdate — Updates the cache for a device.
- bleDeviceCacheDelete — Deletes the cache for one or all devices.
- bleServiceGet — Get information about a service on a device.
- bleServiceFind — Find a specific service on a device.
- bleCharGet — Get information about a characteristic on a device.
- bleCharFind — Find a specific characteristic on a device.
- bleWriteVal — Writes a value to a characteristic
- bleReadVal — Reads the value of a characteristic
- bleNotifyRegister — Registers a notification callback on a characteristic
- bleNotifyRelease — Releases a notification callback.
- bleIndicationRegister — Registers an indication callback on a characteristic
- bleIndicationRelease — Releases an indication callback.

Helper functions

- bleMacCreate — Combines a simple MAC address with the address type.
- bleTimeFromMs — Converts a time interval from ms to the number of 0.625ms ticks.

MAC address format

BLE uses different kinds of MAC addresses.
A public MAC address is a standard 48-bit MAC address as assigned by the IEEE Registration Authority.

A random device address is generated on the device and might change on periodically.
To differentiate between these addresses, this API uses strings of the following format to represent MAC addresses:

```
"yXX:XX:XX:XX:XX:XX"
```

y is the address type and is 'p' for public addresses and 'r' for random addresses.
X is a hexadecimal value from 0-F, and is part of the normal address.
The MAC address string can be created manually or using the bleMacCreate function.

Time units

This API uses the number of 0.625ms ticks for some time intervals. To convert a time in milliseconds to this format, use the bleTimeFromMs function

Error codes

The BLE API uses a number of common error codes across the functions.
The following table lists the common error codes. The meaning may change slightly from function to function.

1	- _BLE_OK	Success
0	- _BLE_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BLE_ERR_NOT_OPEN	The interface is not powered(see blePower).
-2	- _BLE_ERR_ALREADY_OPEN	The interface is already turned on.
-4	- _BLE_ERR_NOT_FOUND	Could not find device.
-5	- _BLE_ERR_NO_ADAP	Could not find interface.
-7	- _BLE_ERR_NO_RES	Could not allocate resources.
-8	- _BLE_ERR_INVAL	Invalid argument.
-9	- _BLE_ERR_NODATA	No valid data found.
-13	- _BLE_ERR_STATE	Invalid state.
-17	- _BLE_ERR_TIMEOUT	Timeout
-19	- _BLE_ERR_NOT_CONNECTED	Device not connected.
-20	- _BLE_ERR_INVAL_HANDLE	Invalid handle.
-21	- _BLE_ERR_NO_CACHE	No cache found.
-99	- _BLE_ERR_GENERIC	Unknown error.

4.2.6.2 blePower (Function)

Architecture: NX32L
Device support: LX2
Firmware version: 2.20.00

This function turns the Bluetooth interface on and off.

Input:

power : BOOL *(default ON)*
If the power should be turned on or off

Returns: INT

1	- _BLE_OK	Success
0	- _BLE_ERR_NOT_SUPPORTED	The API is not supported.
-2	_BLE_ERR_ALREADY_OPEN	The interface is already on.
-4	_BLE_ERR_NOT_OPEN	The interface is already off.
-5	- _BLE_ERR_NO_ADAP	Could not find interface.

Declaration:

```
FUNCTION blePower : INT;
VAR_INPUT
   power  : BOOL := ON;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

   // Turn on power
   blePower();

BEGIN
  ...
END;

END_PROGRAM;
```

4.2.6.3 bleObserverStart (Function)

Architecture: NX32L
Device support: LX2
Firmware version: 2.20.00

This function makes the Bluetooth interface start listening for advertising data.
This can both be used to receive advertising data from known devices and to detect new devices.

Input:

scan_interval: UINT (4..16384) *(default 16384)*
The time interval from from one scan is started until the next is started.
The time is provided in ticks of 0.625 ms. This gives a range from 2.5ms to 10.24 s.

scan_window: UINT (4..16384) *(default 16384)*
The duration to scan for. Must be less than or equal to the scan interval.
The time is provided in ticks of 0.625 ms. This gives a range from 2.5ms to 10.24 s.

scan_type: USINT (0..1) *(default 0)*
The kind of scan to perform:

0 - Passive scan	The device only listens for advertisements from other devices.
1 - Active scan	The device may request further information when it receives an advertisement.

filter_policy: INT *(default 0)*
The policy to use to filter incoming packets:
0 -Accept all
1 -Ignore devices not on the accept list.

Returns: INT

1 - _BLE_OK	Success
0 - _BLE_ERR_NOT_SUPPORTED	The API is not supported.
-1 - _BLE_ERR_NOT_OPEN	The interface is not powered(see blePower).
-8 - _BLE_ERR_INVAL	Invalid parameter.

-13	- _BLE_ERR_STATE	Already listening
-99	- _BLE_ERR_GENERIC	Failed to start listening

Declaration:

```
FUNCTION bleObserverStart : INT;
VAR_INPUT
   scan_interval   : UINT  := 16#4000;
   scan_window     : UINT  := 16#4000;
   scan_type       : USINT := 0;
   filter_dup      : BOOL  := TRUE;
   filter_policy   : USINT := 0;
END_VAR;
```

Example:

```
...
   rc := bleObserverStart(
      scan_interval:=bleTimeFromMs(v:=1000),
      scan_window := bleTimeFromMs(v:=1000),
      scan_type:=1,
      filter_policy := 1
   );
   DebugFmt(message:="bleObserverStart: \1", v1:=rc);
...
```

4.2.6.4 bleObserverStop (Function)

Architecture: NX32L
Device support: LX2
Firmware version: 2.20.00

This function makes the Bluetooth interface stop listening for advertising data.

Returns: **INT**

1	- _BLE_OK	Success
0	- _BLE_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BLE_ERR_NOT_OPEN	The interface is not powered(see blePower).
-13	- _BLE_ERR_STATE	Adapter was not listening.
-99	- _BLE_ERR_GENERIC	Failed to start listening

Declaration:

```
FUNCTION bleObserverStop : INT;
```

Example:

```
...
// Stop listening
rc := bleObserverStop();
...
```

4.2.6.5 bleRegisterAdvData (Function)

```
Architecture:       NX32L
Device support:     LX2
Firmware version:   2.20.00
```

This function registers a callback function to handle incoming advertising data.
This callback will be provided with each individual piece of advertising data as it is received. To receive the raw advertising data, use bleRegisterAdvRaw instead.

Input:

ev_type: UINT *(default 0)*
The type of advertising events to handle. This value is combined with ev_mask to find matching events.
Bitmask:

Bit 0 - Connectable advertising	It is allowed to connect to this device.
Bit 1 - Scannable advertising	The device can provide additional data as a scan response when using active scanning.
Bit 2 - Directed advertising	The advertising is directed at a specific device.
Bit 3 - Scan response	This is a response to an active scan.
Bit 4 - Legacy advertising.	If set to 0, the device uses extended advertising.
Remaining bits: Reserved	

ev_mask: UINT *(default 0)*
Bitmask determining how ev_type should be used. If a bit is set in this mask, it must match the same bit in ev_type.
Setting ev_mask to 0 will handle all events.
Setting ev_mask to 16#FFFF will require that ev_type matches on all defined bits.
To receive scannable and scan response legacy advertising, set ev_mask to 16#0017 and ev_type to 16#0012.
To receive only non connectable legacy advertising, set ev_mask to 16#001F and ev_type to 16#0010.

adv_type: INT (-1, 0..256) *(default -1)*
The type of advertising data to handle. Use -1 to handle all data types.
See 2.3 Common Data Types in the Assigned Numbers Document at
https://www.bluetooth.com/specifications/assigned-numbers/

Examples:

`16#01`	- Flags
`16#03`	- Complete list of 16-bit Service Class UUIDs
`16#09`	- Complete local name
`16#FF`	- Manufacturer specific data.

cb_adv: *CALLBACK*
The function to call when advertising data is received. See the callback declaration below.

cb_arg: DINT
User argument to pass on to the callback.

Returns: INT

1	- _BLE_OK	Success
0	- _BLE_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BLE_ERR_NOT_OPEN	The interface is not powered(see blePower).
-7	_BLE_ERR_NO_RES	Too many callbacks have been registered.
-8	- _BLE_ERR_INVAL	Invalid parameter.

Declaration:

```
FUNCTION bleRegisterAdvData : INT;
VAR_INPUT
   ev_type    : UINT  := 0;
   ev_mask    : UINT  := 0;
   adv_type   : INT   := -1;
   cb_adv     : CALLBACK;
   cb_arg     : DINT;
END_VAR;
```

Callback declaration:

```
FUNCTION CALLBACK cbAdvData : INT;
VAR_INPUT
   mac        : STRING; // MAC address for the sender
   ev_type    : UINT;   // Event type for the data
   adv_type   : INT;    // Advertising data type.
   data       : PTR;    // The advertising data. This pointer may not be used
outside the callback.
   len        : INT;    // The size of the advertising data.
   rssi       : SINT;   // The RSSI of the packet.
   arg        : DINT;   // The user argument.
END_VAR;
```

Example:

```
FUNCTION CALLBACK cbAdvName;
VAR_INPUT
   mac      : STRING;
   ev_type  : UINT;
   adv_type : USINT;
   data     : PTR;
   len      : INT;
   rssi     : SINT;
   arg      : DINT;
END_VAR;
VAR
   str     : STRING;
END_VAR;
   str:=strFromMemory(src:=data, len:=len);
   DebugFmt(message:=mac+"(\1): "+str, v1:=rssi);
END_FUNCTION;

...
// Register callback to show complete local name
rc := bleRegisterAdvData(cb_Adv:=@cbAdvName, adv_type := 9);
...
```

4.2.6.6 bleRegisterAdvRaw (Function)

Architecture: NX32L
Device support: LX2
Firmware version: 2.20.00

This function registers a callback function to handle incoming advertising data.
This callback will be provided with the raw advertising data as it is received. To receive the parsed advertising data, use bleRegisterAdvData instead.

Input:

ev_type: UINT *(default 0)*
The type of advertising events to handle. This value is combined with ev_mask to find matching events.
Bitmask:

Bit 0 - Connectable advertising	It is allowed to connect to this device.
Bit 1 - Scannable advertising	The device can provide additional data as a scan response when using active scanning.
Bit 2 - Directed advertising	The advertising is directed at a specific device.
Bit 3 - Scan response	This is a response to an active scan.
Bit 4 - Legacy advertising.	If set to 0, the device uses extended advertising.
Remaining bits: Reserved	

ev_mask: UINT *(default 0)*
Bitmask determining how ev_type should be used. If a bit is set in this mask, it must match the same bit in ev_type.
Setting ev_mask to 0 will handle all events.
Setting ev_mask to 16#FFFF will require that ev_type matches on all defined bits.
To receive scannable and scan response legacy advertising, set ev_mask to 16#0017 and ev_type to 16#0012.
To receive only non connectable legacy advertising, set ev_mask to 16#001F and ev_type to 16#0010.

cb_adv: *CALLBACK*
The function to call when advertising data is received. See the callback declaration below.

cb_arg: DINT
User argument to pass on to the callback.

Returns: INT

1 - _BLE_OK	Success
0 - _BLE_ERR_NOT_SUPPORTED	The API is not supported.
-1 - _BLE_ERR_NOT_OPEN	The interface is not powered(see blePower).
-7 _BLE_ERR_NO_RES	Too many callbacks have been registered.
-8 - _BLE_ERR_INVAL	Invalid parameter.

Declaration:

```
FUNCTION bleRegisterAdvRaw : INT;
VAR_INPUT
   ev_type    : UINT  := 0;
   ev_mask    : UINT  := 0;
   cb_adv     : CALLBACK;
   cb_arg     : DINT;
END_VAR;
```

Callback declaration:

```
FUNCTION CALLBACK cbAdvRaw : INT;
VAR_INPUT
   mac        : STRING; // MAC address for the sender
   ev_type    : UINT;   // Event type for the data
   data       : PTR;    // The advertising data. This pointer may not be used
outside the callback.
   len        : INT;    // The size of the advertising data.
   rssi       : SINT;   // The RSSI of the packet.
   arg        : DINT;   // The user argument.
END_VAR;
```

Example:

```
FUNCTION CALLBACK cbAdvRaw;
VAR_INPUT
   mac      : STRING;
   ev_type  : UINT;
   data     : PTR;
   len      : INT;
   rssi     : SINT;
   arg      : DINT;
END_VAR;
VAR
   i        : INT :=0;
   adv_type : USINT;
   adv_len  : USINT;
END_VAR;
   DebugFmt(message:=mac+" RSSI: \2, Len: \3, Type: "+intToHex(v:=ev_type), v2:=rssi, v3:
   // Print out the size and type of the contained advertising data
   WHILE i < len DO
      memcpy(dst := ADDR(adv_len), src:=data+i, len:=1);
      IF adv_len > 0 THEN
         memcpy(dst:=ADDR(adv_type), src:=data+i+1, len:=1);
         DebugFmt(message:="  Len: \1, Type: "+sintToHex(v:=adv_type), v1:=adv_len);

         i := i+1+adv_len;
      ELSE
         // Empty
         i := i+1;
      END_IF;
   END_WHILE;
END_FUNCTION;


...
   rc := bleRegisterAdvRaw(cb_Adv:=@cbAdvRaw, ev_type := 16#0010, ev_mask := 16#0010);
   DebugFmt(message := "BleRegisterAdvRaw: \1", v1 := rc);
...
```

4.2.6.7 bleAcceptFilterAdd (Function)

Architecture: NX32L
Device support: LX2
Firmware version: 2.20.00

This function adds a MAC address to the accept filter.

Input:

MAC: STRING
The MAC address to add to the accept filter.

Returns: **INT**

1	- _BLE_OK	Success
0	- _BLE_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BLE_ERR_NOT_OPEN	The interface is not powered(see blePower).
-8	- _BLE_ERR_INVAL	Invalid MAC address.

-99	- _BLE_ERR_GENERIC	Failed to add address.

Declaration:

```
FUNCTION bleAcceptFilterAdd : INT;
VAR_INPUT
   MAC       : STRING;
END_VAR;
```

Example:

```
...
// Add MAC address to accept filter
rc := bleAcceptFilterAdd(mac:=bleMacCreate(mac:="E3:1C:E6:43:F5:2D",
type:=1));
...
```

4.2.6.8 bleAcceptFilterRemove (Function)

Architecture: NX32L
Device support: LX2
Firmware version: 2.20.00

This function removes a MAC address from the accept filter.

Input:

MAC: STRING
The MAC address to remove from the accept filter.

Returns: **INT**

1	- _BLE_OK	Success
0	- _BLE_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BLE_ERR_NOT_OPEN	The interface is not powered(see blePower).
-8	- _BLE_ERR_INVAL	Invalid MAC address.
-99	- _BLE_ERR_GENERIC	Failed to remove address.

Declaration:

```
FUNCTION bleAcceptFilterRemove : INT;
VAR_INPUT
   MAC       : STRING;
END_VAR;
```

Example:

```
...
// Remove MAC address from accept filter
rc := bleAcceptFilterRemove(mac:="rE3:1C:E6:43:F5:2D");
...
```

4.2.6.9 bleAcceptFilterClear (Function)

Architecture: NX32L
Device support: LX2
Firmware version: 2.20.00

This function removes all MAC addresses from the accept filter.

Input:
None.

Returns: INT

1	- _BLE_OK	Success
0	- _BLE_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BLE_ERR_NOT_OPEN	The interface is not powered(see blePower).
-99	- _BLE_ERR_GENERIC	Failed to remove addresses.

Declaration:
```
FUNCTION bleAcceptFilterClear : INT;
```

Example:

```
...
// Remove all MAC addresses from accept filter
rc := bleAcceptFilterClear();
...
```

4.2.6.10 bleConnect (Function)

Architecture: NX32L
Device support: LX2
Firmware version: 2.25.00

This function makes the Bluetooth interface start connecting to a peripheral device.

The default parameters should work in most cases, so it is normally only necessary to provide the mac and dev parameters.

Once this function has returned successfully, the function bleDeviceStatus can be used to monitor the actual status of the peripheral device.

For advanced usage, some of the other connection parameters can be adjusted.

Scan interval and window
These parameters determine how it searches for the device to connect to, if they are set to the same value, it will search continuously.

Connection interval
This is how often the central asks the connected peripheral for data (connection event), which determines how often data can be transferred. A longer connection interval can reduce the power consumption of the peripheral but increases the latency and reduces the throughput.

Latency
The latency determines how many connection events the peripheral is allowed to skip before it must respond.
This enables the peripheral to sleep through events if it does not have anything to send, reducing the power consumption.

Supervision timeout
The supervision timeout specifies how much time may pass without receiving a packet before considering the connection lost.
Must be larger than (1 + latency_max) * con_interval_max * 2.

Connection length
The connection length is used to indicate how long the connection must be, to plan the time allocation for multiple connections.

Note: Scanning must be stopped before it is possible to connect.

Input:

mac: STRING
The MAC address of the device to connect to.

scan_interval: UINT (4..16384) ***(default 16384, 10.24ms)***
The time interval from from one scan is started until the next is started.
The time is provided in ticks of 0.625 ms. This gives a range from 2.5ms to 10.24 s.

scan_window: UINT (4..16384) ***(default 16384, 10.24ms)***
The duration to scan for. Must be less than or equal to the scan interval.
The time is provided in ticks of 0.625 ms. This gives a range from 2.5ms to 10.24 s.

con_interval_min: UINT (6..3200) ***(default 24, 30ms)***
The minimum connection interval.
The time is provided in ticks of 1.25 ms. This gives a range from 7.5ms to 4 s.

con_interval_max: UINT (6..3200) ***(default 40, 50ms)***
The maximum connection interval.
The time is provided in ticks of 1.25 ms. This gives a range from 7.5ms to 4 s.

latency_max: UINT (0..499) ***(default 0)***
The maximum latency for the connection.
If set to 0, the peripheral must respond to every event.

supervision_timeout: UINT (10..3200) ***(default 3200)***
If more than this time has elapsed since the most recent packet, the connection will be terminated.
The value will be multiplied by 10 ms. This gives a range from 100ms to 32s.

con_length_min: UINT (0..65535) ***(default 16, 10ms)***
The minimum length of the connection needed.
The time is provided in ticks of 0.625 ms. This gives a range from 0ms to 40.96 s.

con_length_max: UINT (0..65535) ***(default 16, 10ms)***
The maximum length of the connection needed.
The time is provided in ticks of 0.625 ms. This gives a range from 0ms to 40.96 s.

Output:

dev : SYSHANDLE
A handle to the device.

Returns: INT

1 - _BLE_OK	Success. Make sure to call bleDisconnect when done with the connection.
0 - _BLE_ERR_NOT_SUPPORTED	The API is not supported.

-1	- _BLE_ERR_NOT_OPEN	The interface is not powered(see blePower).
-7	_BLE_ERR_NO_RES	All connections are in use.
-8	- _BLE_ERR_INVAL	Invalid parameter.
-13	- _BLE_ERR_STATE	Can not connect at this time, try again.
-99	- _BLE_ERR_GENERIC	Failed to start connection

Declaration:

```
FUNCTION bleConnect : INT;
VAR_INPUT
   mac                : STRING;
   scan_interval      : UINT  := 16#4000;
   scan_window        : UINT  := 16#4000;
   con_interval_min   : UINT  := 16#0018;
   con_interval_max   : UINT  := 16#0028;
   latency_max        : UINT  := 0;
   supervision_timeout : UINT  := 16#0C80;
   con_length_min     : UINT  := 16;
   con_length_max     : UINT  := 16;
   dev                : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
...
   rc := bleConnect(dev := dev, mac := bleMacCreate(mac:="E3:1C:E6:43:F5:2D",
type:=1));
   DebugFmt(message:="bleConnect: \1", v1:=rc);
...
```

4.2.6.11 bleDisconnect (Function)

Architecture: NX32L
Device support: LX2
Firmware version: 2.25.00

This function makes the Bluetooth interface disconnect from a device.
The handle is invalid after this function is done.

Input:

dev: SYSHANDLE
Handle to the device to disconnect from.

Returns: INT

1	- _BLE_OK	Success
0	- _BLE_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BLE_ERR_NOT_OPEN	The interface is not powered(see blePower).
-4	- _BLE_ERR_NOT_FOUND	Failed to find device.
-20	- _BLE_ERR_INVAL_HANDLE	Invalid handle.

Declaration:

```
FUNCTION bleDisconnect : INT;
VAR_INPUT
   dev    : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
...
   rc := bleDisconnect(dev := dev);
   DebugFmt(message:="bleDisconnect: \1", v1:=rc);
...
```

4.2.6.12 bleDeviceStatus (Function)

Architecture: NX32L
Device support: LX2
Firmware version: 2.25.00

This function returns the connection status of the device.
If the device is connected, the return code also tells the status of the service cache, see bleServiceCacheUpdate.

Input:

dev: SYSHANDLE
Handle to the device to get the status of.

Returns: INT

22	-	-	Connection is ready for use, using cache from this connection.
21	-	-	Connection is ready for use, using cache from previous connection.
20	-	-	Connection is ready for use, no cache loaded. Use bleServiceCacheUpdate to create the cache.
10	-	-	Connected, waiting to configure connection.
5	-	-	Trying to establish connection
2	-	-	Connection lost. Call bleDisconnect to clean up the connection.
1	-	-	Not connected.
0	-	_BLE_ERR_NOT_SUPPORTED	The API is not supported.
-1	-	_BLE_ERR_NOT_OPEN	The interface is not powered(see blePower).
-4	-	_BLE_ERR_NOT_FOUND	Failed to find device.
-20	-	_BLE_ERR_INVAL_HANDLE	Invalid handle.

Declaration:

```
FUNCTION bleDeviceStatus : INT;
VAR_INPUT
   dev    : SYSHANDLE;
END_VAR;
```

Example:

```
...
   rc := bleDeviceStatus(dev := dev);
   DebugFmt(message:="bleDeviceStatus: \1", v1:=rc);
...
```

4.2.6.13 bleDeviceMacGet (Function)

Architecture: NX32L
Device support: LX2
Firmware version: 2.25.00

This function returns the MAC address of a device.

Input:

dev: SYSHANDLE
Handle to the device to get the MAC address of.

Returns: STRING

The MAC address of the device.
An empty string is returned on error.

Declaration:

```
FUNCTION bleDeviceMacGet : INT;
VAR_INPUT
   dev    : SYSHANDLE;
END_VAR;
```

Example:

```
...
   DebugMsg(message:="MAC: " + bleDeviceMacGet(dev := dev));
...
```

4.2.6.14 bleServiceCacheUpdate (Function)

Architecture: NX32L
Device support: LX2
Firmware version: 2.25.00

This function tries to update the service cache for a connected device.

The service cache is used to cache all the services and characteristics of the device, so it is not necessary to read them all every time the device is connected. This makes it possible to quickly connect to a device, read or write a value and disconnect again.

This cache is persistent across runtime resets and is deleted on system resets(boardReset(level := 1)).

To manually delete the cache for a single device or for all devices, see bleDeviceCacheDelete.

Input:

dev: SYSHANDLE
Handle to the device to update the cache for.

Returns: **INT**

1	- _BLE_OK	Success
0	- _BLE_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BLE_ERR_NOT_OPEN	The interface is not powered(see blePower).
-4	- _BLE_ERR_NOT_FOUND	Failed to find device.
-13	_BLE_ERR_STATE	Not allowed to update cache at this time.
-17	_BLE_ERR_TIMEOUT	Cache update timed out.
-19	_BLE_ERR_NOT_CONNECTED	Device is not connected.
-20	- _BLE_ERR_INVAL_HANDLE	Invalid handle.

Declaration:

```
FUNCTION bleServiceCacheUpdate : INT;
VAR_INPUT
   dev    : SYSHANDLE;
END_VAR;
```

Example:

```
...
   rc := bleServiceCacheUpdate(dev := dev);
   DebugFmt(message:="bleServiceCacheUpdate: \1", v1:=rc);
...
```

4.2.6.15 bleDeviceCacheDelete (Function)

Architecture: **NX32L**
Device support: **LX2**
Firmware version: **2.25.00**

This function deletes the service cache for a connected device or for all devices.
See bleServiceCacheUpdate for details about the cache.

Input:

dev: SYSHANDLE
Handle to the device to delete the cache for. If not provided, it will delete the cache for all devices.

Returns: **INT**

1	- _BLE_OK	Success
0	- _BLE_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BLE_ERR_NOT_OPEN	The interface is not powered(see blePower).
-4	- _BLE_ERR_NOT_FOUND	Failed to find device.
-99	- _BLE_ERR_GENERIC	Failed to delete cache

Declaration:

```
FUNCTION bleDeviceCacheDelete : INT;
VAR_INPUT
   dev    : SYSHANDLE;
END_VAR;
```

Example:

```
...
   rc := bleDeviceCacheDelete(dev := dev);
```

```
   DebugFmt(message:="bleDeviceCacheDelete: \1", v1:=rc);
...
```

4.2.6.16 bleServiceGet (Function)

Architecture: NX32L
Device support: LX2
Firmware version: 2.25.00

This function will retrieve a service from the device.
To list all the known services, keep calling this function with incrementing index until it returns -9 (_BLE_ERR_NODATA).

Input:

dev: SYSHANDLE
Handle to the device to get the service from.

idx: INT*(default 0)*
Zero-based index of the service to read.

Output:

service : UINT
The ID of the service.

primary : BOOL
TRUE if this is a primary service.

UUID : STRING
The UUID of this service.

Returns: INT

1	- _BLE_OK	Success
0	- _BLE_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BLE_ERR_NOT_OPEN	The interface is not powered(see blePower).
-4	- _BLE_ERR_NOT_FOUND	Failed to find device.
-9	_BLE_ERR_NODATA	Service not found.
-20	_BLE_ERR_INVAL_HANDLE	Invalid handle.
-21	_BLE_ERR_NO_CACHE	No cache found, use bleServiceCacheUpdate to update the cache.

Declaration:

```
FUNCTION bleServiceGet : INT;
VAR_INPUT
   dev       : SYSHANDLE;
   idx       : INT;
   service   : ACCESS UINT;
   primary   : ACCESS BOOL;
   uuid      : ACCESS STRING;
END_VAR;
```

Example:

```
FUNCTION ShowServices;
VAR_INPUT
```

```
   dev     : SYSHANDLE;
END_VAR;
VAR
   i, j    : INT;
   s, c    : UINT;
   rc, rc2 : INT;
   uuid    : STRING;
   primary : BOOL;
   flags   : DINT;
END_VAR;
   DebugMsg(message := "-- Services: --");
   i := 0;
   REPEAT
      rc := bleServiceGet(dev := dev, idx := i, service := s, uuid := uuid,
primary := primary);
      IF rc = 1 THEN
         DebugFmt(message := "  Service: \1, primary: \2, UUID: " + UUID, v1
:= s, v2 := INT(primary));
         j := 0;
         REPEAT
            rc2 := bleCharGet(dev := dev, service := s, idx := j, char := c,
uuid := uuid, flags := flags);
            IF rc2 = 1 THEN
               DebugFmt(message := "    Char: \1, flags: " + dintToHex(v :=
flags) + ", UUID: " + UUID, v1 := c);
            ELSE
               DebugFmt(message := "    End(\2): \1", v1 := rc2, v2 := j);
            END_IF;
            j := j + 1;
         UNTIL rc2 <> _BLE_OK
         END_REPEAT;
      ELSE
         DebugFmt(message := "  End(\2): \1", v1 := rc, v2 := i);
      END_IF;
      i := i + 1;
   UNTIL rc <> _BLE_OK
   END_REPEAT;
END_FUNCTION;
```

4.2.6.17 bleServiceFind (Function)

Architecture: NX32L
Device support: LX2
Firmware version: 2.30.00

This function will find a service on the device that matches a specific UUID.
If the device has multiple services with the same UUID, this function can be called with incrementing index until it returns -9 (_BLE_ERR_NODATA).

The service can then be used by either bleCharGet or bleCharFind to find the wanted characteristics.

Input:

dev: SYSHANDLE
Handle to the device to get the service from.

UUID : STRING
The UUID of this service to find.
Three types of UUID are supported:

16-bit: "2a06"
32-bit: "00002a06"
128-bit: "00002a06-0000-1000-8000-00805f9b34fb"

idx: INT *(default 0)*
Zero-based index of the service to read.

Output:

service : UINT
The ID of the service.

primary : BOOL
TRUE if this is a primary service.

Returns: INT

1	- _BLE_OK	Success
0	- _BLE_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BLE_ERR_NOT_OPEN	The interface is not powered(see blePower).
-4	- _BLE_ERR_NOT_FOUND	Failed to find device.
-8	_BLE_ERR_INVAL	Invalid UUID
-9	_BLE_ERR_NODATA	Service not found.
-20	_BLE_ERR_INVAL_HANDLE	Invalid handle.
-21	_BLE_ERR_NO_CACHE	No cache found, use bleServiceCacheUpdate to update the cache.

Declaration:

```
FUNCTION bleServiceFind : INT;
VAR_INPUT
   dev       : SYSHANDLE;
   uuid      : STRING;
   idx       : INT := 0;
   service   : ACCESS UINT;
   primary   : ACCESS BOOL;
END_VAR;
```

Example:

```
FUNCTION ShowCharsForService;
VAR_INPUT
   dev     : SYSHANDLE;
   service : STRING;
END_VAR;
VAR
   i       : INT;
   s, c    : UINT;
   rc      : INT;
   uuid    : STRING;
   primary : BOOL;
   flags   : DINT;
END_VAR;
   DebugMsg(message := "-- Service: --");
   rc := bleServiceFind(dev:=dev, uuid:=service, service:=s, primary :=
primary);
   IF rc = _BLE_OK THEN
      DebugFmt(message := "  Service: \1, primary: \2", v1 := s, v2 :=
INT(primary));
      i := 0;
      REPEAT
```

```
            rc:= bleCharGet(dev := dev, service := s, idx := i, char := c, uuid
:= uuid, flags := flags);
            IF rc = _BLE_OK THEN
               DebugFmt(message := "    Char: \1, flags: " + dintToHex(v :=
flags) + ", UUID: " + UUID, v1 := c);
            ELSE
               DebugFmt(message := "    End(\2): \1", v1 := rc, v2 := i);
            END_IF;
            i := i + 1;
         UNTIL rc <> _BLE_OK
         END_REPEAT;
      ELSE
         DebugFmt(message:="bleServiceFind failed: \1", v1:=rc);
      END_IF;
END_FUNCTION;
```

4.2.6.18 bleCharGet (Function)

Architecture: NX32L
Device support: LX2
Firmware version: 2.25.00

This function will retrieve a characteristic from the device.
To list all the known characteristics, keep calling this function with incrementing index until it returns -9 (_BLE_ERR_NODATA).

Input:

dev: SYSHANDLE
Handle to the device.

idx: INT
Zero-based index of the characteristic to read.

service: UINT
ID of the service to get the characteristic from.

Output:

char : UINT
The ID of the characteristic.

flags : DINT
A bit field with the flags for this characteristic.

Bit	Hex	Name	Description
2	02	Read	The characteristic can be read, see bleReadVal.
3	04	Write no response	The characteristic can be written without waiting for response, which allows for higher throughput. See bleWriteVal.
4	08	Write	The characteristic can be written and waits for the response. See bleWriteVal.
5	10	Notify	The characteristic supports notifications, which can be used to receive data whenever it is ready, e.g. when triggered by events. See bleNotifyRegister.
6	20	Indicate	The characteristic supports indications, which can be used to receive data whenever it is ready. Unlike Notify, Indicate lets the sender know that the data has been received. See bleIndicationRegister.

UUID : STRING
The UUID of this characteristic.

Returns: INT

1	- _BLE_OK	Success
0	- _BLE_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BLE_ERR_NOT_OPEN	The interface is not powered(see blePower).
-4	- _BLE_ERR_NOT_FOUND	Failed to find device.
-9	_BLE_ERR_NODATA	Characteristic not found.
-20	_BLE_ERR_INVAL_HANDLE	Invalid handle.
-21	_BLE_ERR_NO_CACHE	No cache found, use bleServiceCacheUpdate to update the cache.

Declaration:

```
FUNCTION bleCharGet : INT;
VAR_INPUT
   dev       : SYSHANDLE;
   idx       : INT;
   service   : UINT;
   char      : ACCESS UINT;
   flags     : ACCESS DINT;
   uuid      : ACCESS STRING;
END_VAR;
```

Example:

```
FUNCTION ShowServices;
VAR_INPUT
   dev     : SYSHANDLE;
END_VAR;
VAR
   i, j    : INT;
   s, c    : UINT;
   rc, rc2 : INT;
   uuid    : STRING;
   primary : BOOL;
   flags   : DINT;
END_VAR;
   DebugMsg(message := "-- Services: --");
   i := 0;
   REPEAT
      rc := bleServiceGet(dev := dev, idx := i, service := s, uuid := uuid,
primary := primary);
      IF rc = 1 THEN
         DebugFmt(message := "  Service: \1, primary: \2, UUID: " + UUID, v1
:= s, v2 := INT(primary));
         j := 0;
         REPEAT
            rc2 := bleCharGet(dev := dev, service := s, idx := j, char := c,
uuid := uuid, flags := flags);
            IF rc2 = 1 THEN
               DebugFmt(message := "    Char: \1, flags: " + dintToHex(v :=
flags) + ", UUID: " + UUID, v1 := c);
            ELSE
               DebugFmt(message := "    End(\2): \1", v1 := rc2, v2 := j);
            END_IF;
            j := j + 1;
```

```
            UNTIL rc2 <> _BLE_OK
            END_REPEAT;
         ELSE
            DebugFmt(message := "  End(\2): \1", v1 := rc, v2 := i);
         END_IF;
         i := i + 1;
      UNTIL rc <> _BLE_OK
      END_REPEAT;
END_FUNCTION;
```

4.2.6.19 bleCharFind (Function)

Architecture: NX32L
Device support: LX2
Firmware version: 2.30.00

This function will find a characteristic on the device that matches a specific UUID.
If the device has multiple characteristics with the same UUID, this function can be called with incrementing index until it returns -9 (_BLE_ERR_NODATA).

Input:

dev: SYSHANDLE
Handle to the device.

UUID : STRING
The UUID of the characteristic to find.
Three types of UUID are supported:
16-bit: "2a06"
32-bit: "00002a06"
128-bit: "00002a06-0000-1000-8000-00805f9b34fb"

idx: INT*(default 0)*
Zero-based index of the characteristic to read.

service: UINT*(default 0)*
ID of the service to get the characteristic from. Leave at 0 to search all services.

Output:

char : UINT
The ID of the characteristic.

flags : DINT
A bit field with the flags for this characteristic.

Bit	Hex	Name	Description
2	02	Read	The characteristic can be read, see bleReadVal.
3	04	Write no response	The characteristic can be written without waiting for response, which allows for higher throughput. See bleWriteVal.
4	08	Write	The characteristic can be written and waits for the response. See bleWriteVal.
5	10	Notify	The characteristic supports notifications, which can be used to receive data whenever it is ready, e.g. when triggered by events. See bleNotifyRegister.
6	20	Indicate	The characteristic supports indications, which can be used to receive data whenever it is ready. Unlike Notify, Indicate lets the sender know that the data has been received. See bleIndicationRegister.

Returns: INT

1	- _BLE_OK	Success
0	- _BLE_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BLE_ERR_NOT_OPEN	The interface is not powered(see blePower).
-4	- _BLE_ERR_NOT_FOUND	Failed to find device.
-8	_BLE_ERR_INVAL	Invalid UUID
-9	_BLE_ERR_NODATA	Characteristic not found.
-20	_BLE_ERR_INVAL_HANDLE	Invalid handle.
-21	_BLE_ERR_NO_CACHE	No cache found, use bleServiceCacheUpdate to update the cache.

Declaration:

```
FUNCTION bleCharFind : INT;
VAR_INPUT
   dev       : SYSHANDLE;
   uuid      : STRING;
   idx       : INT  := 0;
   service   : UINT := 0;
   char      : ACCESS UINT;
   flags     : ACCESS DINT;
END_VAR;
```

Example:

```
...
         // Ready, try reading cache
         rc := bleServiceCacheUpdate(dev:=dev);
         DebugFmt(message:="bleServiceCacheUpdate: \1", v1:=rc);

         // Find wanted characteristic

         // Battery level characteristic
         rc := bleCharFind(dev:=dev, uuid:="00002a19-0000-1000-8000-
00805f9b34fb", char:=c, flags := flags);
         DebugFmt(message:="bleCharFind: \1, Char: \2, flags: \4", v1:=rc,
v2:=c, v4:=flags);
         IF rc = _BLE_OK THEN
            bat_level_id := c;
         END_IF;
         // Read from characteristic
         size:=1;
         rc := bleReadVal(dev:=dev, char:=bat_level_id, size := size, data :=
ADDR(bat_level));
         DebugFmt(message:="bleReadVal(\2): \1", v1:=rc, v2:=bat_level_id);
...
```

4.2.6.20 bleWriteVal (Function)

Architecture: NX32L
Device support: LX2
Firmware version: 2.25.00

This function will write data to a characteristic on a device.

Input:

dev: SYSHANDLE
Handle to the device.

char : UINT
The ID of the characteristic to write to.

mode : INT *(default 0)*
The kind of write to perform.

0	-Auto	Use write with response if supported, otherwise try write with no response.
1	-No response	Use write with no response.
2	-Response	Use write with response.

size : INT
The size of the data to write.

data : PTR
A pointer to the data to write.

Returns: INT

1	- _BLE_OK	Success
0	- _BLE_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BLE_ERR_NOT_OPEN	The interface is not powered(see blePower).
-4	- _BLE_ERR_NOT_FOUND	Failed to find device.
-7	- _BLE_ERR_NO_RES	Data length is not supported by device.
-8	- _BLE_ERR_INVAL	Selected mode not supported. If mode is 0, the characteristic does not support writing.
-9	- _BLE_ERR_NODATA	Characteristic not found
-19	- _BLE_ERR_NOT_CONNECTED	Device is not connected.
-20	- _BLE_ERR_INVAL_HANDLE	Invalid handle.
-21	- _BLE_ERR_NO_CACHE	No cache found, use bleServiceCacheUpdate to update the cache.
-99	- _BLE_ERR_GENERIC	Failed to write

Declaration:

```
FUNCTION bleWriteVal : INT;
VAR_INPUT
   dev        : SYSHANDLE;
   char       : UINT;
   mode       : INT;
   size       : INT;
   data       : PTR;
END_VAR;
```

Example:

```
...
   rc := bleWriteVal(dev:=dev, char:=led_color_id, size := 4, data :=
ADDR(colors));
   DebugFmt(message:="bleWriteVal(): \1: ", v1:=rc);
...
```

4.2.6.21 bleReadVal (Function)

Architecture: NX32L
Device support: LX2
Firmware version: 2.25.00

This function will read data from a characteristic on a device.

Input:

dev: SYSHANDLE
Handle to the device.

char : UINT
The ID of the characteristic to read from.

size : INT
The size of the buffer to read the data into.

data : PTR
A pointer to a buffer to store the data in.

Output:

size : INT
The size of the read data.

Returns: INT

1	- _BLE_OK	Success
0	- _BLE_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BLE_ERR_NOT_OPEN	The interface is not powered(see blePower).
-4	- _BLE_ERR_NOT_FOUND	Failed to find device.
-7	- _BLE_ERR_NO_RES	Data length is not supported by device.
-8	- _BLE_ERR_INVAL	Read is not supported.
-9	- _BLE_ERR_NODATA	Characteristic not found
-13	_BLE_ERR_STATE	Not allowed to read at this time.
-19	- _BLE_ERR_NOT_CONNECTED	Device is not connected.
-20	- _BLE_ERR_INVAL_HANDLE	Invalid handle.
-21	- _BLE_ERR_NO_CACHE	No cache found, use bleServiceCacheUpdate to update the cache.
-99	- _BLE_ERR_GENERIC	Failed to read

Declaration:

```
FUNCTION bleReadVal : INT;
VAR_INPUT
   dev       : SYSHANDLE;
   char      : UINT;
   size      : ACCESS INT;
   data      : PTR;
END_VAR;
```

Example:

```
...
         size:=1;
         rc := bleReadVal(dev:=dev, char:=bat_level_id, size := size, data :=
ADDR(bat_level));
         DebugFmt(message:="bleReadVal(\2): \1", v1:=rc, v2:=bat_level_id);
```

```
...
```

4.2.6.22 bleNotifyRegister (Function)

Architecture: NX32L
Device support: LX2
Firmware version: 2.25.00

This function registers a callback function to handle notification data for a characteristic. Notification data is used for events that do not need a response.

The callback can be released by calling bleNotifyRelease.

Input:

dev: SYSHANDLE
Handle to the device.

char : UINT
The ID of the characteristic.

func: *CALLBACK*
The function to call when a notification is received. See the callback declaration below.

arg: DINT
User argument to pass on to the callback.

Returns: INT

1 - _BLE_OK	Success
0 - _BLE_ERR_NOT_SUPPORTED	The API is not supported.
-1 - _BLE_ERR_NOT_OPEN	The interface is not powered(see blePower).
-7 _BLE_ERR_NO_RES	Too many notification callbacks have been registered.
-8 - _BLE_ERR_INVAL	Characteristic does not support notification.
-9 - _BLE_ERR_NODATA	Characteristic not found
-13 _BLE_ERR_STATE	Not allowed to read at this time.
-19 - _BLE_ERR_NOT_CONNECTED	Device is not connected.
-17 _BLE_ERR_TIMEOUT	Timed out when trying to enable notification.
-20 - _BLE_ERR_INVAL_HANDLE	Invalid handle.
-21 - _BLE_ERR_NO_CACHE	No cache found, use bleServiceCacheUpdate to update the cache.
-99 - _BLE_ERR_GENERIC	Failed to enable notification

Declaration:

```
FUNCTION bleNotifyRegister : INT;
VAR_INPUT
   dev    : SYSHANDLE;
   char   : INT;
   func   : CALLBACK;
   arg    : DINT;
END_VAR;
```

Callback declaration:

```
FUNCTION CALLBACK bleNotifyCb : INT;
VAR_INPUT
   dev       : SYSHANDLE; // Handle to device
```

```
   service    : UINT;       // Service
   char       : UINT;       // Characteristic
   data       : PTR;        // The notification data. This pointer may not be
used outside the callback.
   len        : INT;        // The size of the data.
   arg        : DINT;       // The user argument.
END_VAR;
```

Example:

```
FUNCTION CALLBACK bleNotifyCb;
VAR_INPUT
   dev        : SYSHANDLE;
   service    : UINT;
   char       : UINT;
   data       : PTR;
   len        : INT;
   arg        : DINT;
END_VAR;
VAR
   str      : STRING;
END_VAR;
   str:=strFromMemory(src:=data, len:=len);
   DebugFmt(message:="\1.\2: "+str, v1:=service, v2:=char);
END_FUNCTION;

...
// Register callback to show notifications for characteristic 10
rc := bleNotifyRegister(dev := dev, char := 10, func:=@bleNotifyCb);
...
```

4.2.6.23 bleNotifyRelease (Function)

Architecture: NX32L
Device support: LX2
Firmware version: 2.25.00

This function releases the notification callback function from a characteristic.

Input:

dev: SYSHANDLE
Handle to the device.

char : UINT
The ID of the characteristic.

Returns: **INT**

1	- _BLE_OK	Success
0	- _BLE_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BLE_ERR_NOT_OPEN	The interface is not powered(see blePower).
-8	- _BLE_ERR_INVAL	Characteristic does not support notification.
-9	- _BLE_ERR_NODATA	Characteristic not found
-19	- _BLE_ERR_NOT_CONNECTED	Device is not connected.
-17	_BLE_ERR_TIMEOUT	Timed out when trying to disable notification.
-20	- _BLE_ERR_INVAL_HANDLE	Invalid handle.

-21	- _BLE_ERR_NO_CACHE	No cache found, use bleServiceCacheUpdate to update the cache.
-99	- _BLE_ERR_GENERIC	Failed to disable notification

Declaration:

```
FUNCTION bleNotifyRelease : INT;
VAR_INPUT
   dev    : SYSHANDLE;
   char   : INT;
END_VAR;
```

Example:

```
...
// Release notification callback for characteristic 10
rc := bleNotifyRelease(dev := dev, char := 10);
...
```

4.2.6.24 bleIndicationRegister (Function)

Architecture: NX32L
Device support: LX2
Firmware version: 2.25.00

This function registers a callback function to handle indication data for a characteristic.
Indication data is used for events that requires confirmation that the data has been received.
This confirmation is sent when the callback returns.
The callback can be released by calling bleIndicationRelease.

Input:

dev: SYSHANDLE
Handle to the device.

char : UINT
The ID of the characteristic.

func: *CALLBACK*
The function to call when an indication is received. See the callback declaration below.

arg: DINT
User argument to pass on to the callback.

Returns: INT

1	- _BLE_OK	Success
0	- _BLE_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BLE_ERR_NOT_OPEN	The interface is not powered(see blePower).
-7	_BLE_ERR_NO_RES	Too many indication callbacks have been registered.
-8	- _BLE_ERR_INVAL	Characteristic does not support indications.
-9	- _BLE_ERR_NODATA	Characteristic not found
-13	_BLE_ERR_STATE	Not allowed to read at this time.
-19	- _BLE_ERR_NOT_CONNECTED	Device is not connected.
-17	_BLE_ERR_TIMEOUT	Timed out when trying to enable indications.
-20	- _BLE_ERR_INVAL_HANDLE	Invalid handle.
-21	- _BLE_ERR_NO_CACHE	No cache found, use bleServiceCacheUpdate to update the cache.
-99	- _BLE_ERR_GENERIC	Failed to enable indications

Declaration:

```
FUNCTION bleIndicationRegister : INT;
VAR_INPUT
   dev    : SYSHANDLE;
   char   : INT;
   func   : CALLBACK;
   arg    : DINT;
END_VAR;
```

Callback declaration:

```
FUNCTION CALLBACK bleIndicationCb : INT;
VAR_INPUT
   dev       : SYSHANDLE; // Handle to device
   service   : UINT;      // Service
   char      : UINT;      // Characteristic
   data      : PTR;       // The indication data. This pointer may not be used
outside the callback.
   len       : INT;       // The size of the data.
   arg       : DINT;      // The user argument.
END_VAR;
```

Example:

```
FUNCTION CALLBACK bleIndicationCb;
VAR_INPUT
   dev       : SYSHANDLE;
   service   : UINT;
   char      : UINT;
   data      : PTR;
   len       : INT;
   arg       : DINT;
END_VAR;
VAR
   str     : STRING;
END_VAR;
   str:=strFromMemory(src:=data, len:=len);
   DebugFmt(message:="\1.\2: "+str, v1:=service, v2:=char);
END_FUNCTION;

...
// Register callback to show indications for characteristic 10
rc := bleIndicationRegister(dev := dev, char := 10, func:=@bleIndicationCb);
...
```

4.2.6.25 bleIndicationRelease (Function)

Architecture: NX32L
Device support: LX2
Firmware version: 2.25.00

This function releases the indication callback function from a characteristic.

Input:

dev: SYSHANDLE

Handle to the device.

char : UINT
The ID of the characteristic.

Returns: INT

1	- _BLE_OK	Success
0	- _BLE_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BLE_ERR_NOT_OPEN	The interface is not powered(see blePower).
-8	- _BLE_ERR_INVAL	Characteristic does not support indications.
-9	- _BLE_ERR_NODATA	Characteristic not found
-19	- _BLE_ERR_NOT_CONNECTED	Device is not connected.
-17	_BLE_ERR_TIMEOUT	Timed out when trying to disable indication.
-20	- _BLE_ERR_INVAL_HANDLE	Invalid handle.
-21	- _BLE_ERR_NO_CACHE	No cache found, use bleServiceCacheUpdate to update the cache.
-99	- _BLE_ERR_GENERIC	Failed to disable indication

Declaration:

```
FUNCTION bleIndicationRelease : INT;
VAR_INPUT
   dev    : SYSHANDLE;
   char   : INT;
END_VAR;
```

Example:

```
...
// Release indication callback for characteristic 10
rc := bleIndicationRelease(dev := dev, char := 10);
...
```

4.2.6.26 bleTimeFromMs (Function)

Architecture: NX32L
Device support: LX2
Firmware version: 2.20.00

This helper function converts a time interval provided in milliseconds to the number of 0.625ms Bluetooth ticks it corresponds to.

Input:

v : INT
The time interval in milliseconds to convert.

Returns: INT

The time interval in Bluetooth ticks.

Declaration:

```
FUNCTION bleTimeFromMs : INT;
VAR_INPUT
   v       : INT;
END_VAR;
```

Example:

```
...
// Get 10 ms in ticks
rc := bleTimeFromMs(v:=10);
...
```

4.2.6.27 bleMacCreate (Function)

Architecture: NX32L
Device support: LX2
Firmware version: 2.20.00

This helper function combines a simple MAC address with the address type, to create a complete MAC address.
This is a simple helper function that adds the address type to a traditional MAC address in the format "XX:XX:XX:XX:XX:XX".
This function performs no validation of the provided MAC address, so the created MAC address might be invalid.

Input:

MAC: STRING
The MAC address to add to the accept filter.

type: USINT (0..1) ***(default 0)***
The address type:
0 - Public address
1 - Random address

Returns: STRING

The new MAC address.

Declaration:

```
FUNCTION bleMacCreate : STRING;
VAR_INPUT
   MAC       : STRING;
   type      : USINT;
END_VAR;
```

Example:

```
...
// Add MAC address to accept filter
rc := bleAcceptFilterAdd(mac:=bleMacCreate(mac:="E3:1C:E6:43:F5:2D",
type:=1));
...
```

4.2.7 bt: Bluetooth (NX)

4.2.7.1 bt: Bluetooth functions (NX32L)

This Bluetooth library is used to manage and communicate wireless with Bluetooth enabled devices. It supports both Bluetooth Classic and Bluetooth Low Energy. For LX devices a dedicated BLE API is available. See the section Bluetooth Low Energy (LX).

Among the supported features are support for serial port profile(SPP) and Bluetooth Low Energy devices(BLE).
The Bluetooth Server example provides an example of an SPP application.

Common Bluetooth management

- btWaitEvent — Waits for events from the Bluetooth module.
- btEventDone — Marks the event as done.
- btPower — Controls power to the Bluetooth module.
- btNameSet — Sets the name to show to other devices.
- btAdapterAddressGet — Get the address of the adapter.
- btMakeDiscoverable — Controls if the device is discoverable.
- btIsDiscoverable — Checks if the device is discoverable.
- btScanStart — Starts scanning for other devices.
- btScanStop — Stops scanning for devices.
- btIsScanning — Checks if the adapter is scanning for devices.
- btDeviceGet — Retrieves the address of a found device.
- btDeviceInfo — Retrieves information about a device.
- btDevicePair — Begins pairing with a device.
- btDeviceRemove — Remove devices to e.g. remove pairing.
- btHandlePairRequest — Handles an incoming pairing request.
- btSendPairResponse — Sends a response to a pairing request.

Profiles

- btSerialPortProfileConnect — Connects to a serial port profile on a remote device.
- btDeviceProfileDisconnect — Disconnects from a profile on a remote device.
- btSerialPortProfileEnable — Enables serial port profiles on the device.
- btHandleSerialPortIncomingConnection — Handles an incoming serial port connection.

Bluetooth Low Energy(BLE)

BLE devices typically broadcasts(advertises) some data which is made available in the manufacturer data.
They may also provide a number of characteristics which are grouped by functionality into services.
Each characteristic is used to manage a specific piece of data, e.g. a temperature or the battery level.

When dealing with a new device, it is necessary to use the btLEServiceGet and btLECharGet functions to determine the IDs of the wanted characteristics.
Once the IDs are known, they can be used directly in the future, removing the need for the search.

The Bluetooth Low Energy example provides an example on how to use the BLE functionality.

- btleManufacturerDataGet — Retrieves the manufacturer data of a device.
- btleServiceDataGet — Retrieves the service data of a device.
- btleConnect — Connect to a BLE device.
- btleDisconnect — Disconnect from a BLE device.
- btleLastSeen — Tells how old the data received from the device is.
- btleServiceGet — Retrieves a service from the device.
- btleCharGet — Retrieves a characteristic from the device.
- btleWriteVal — Writes a value to a characteristic.
- btleReadVal — Reads the value of a characteristic.
- btleNotifyStart — Starts listening for notifications on a characteristic.
- btleNotifyStop — Stops listening for notifications on a characteristic.
- btleHandleNotification — Reads the data belonging to a BLE notification.

Error codes

The Bluetooth API uses a number of common error codes across the functions.
The following table lists the common error codes. The meaning may change slightly from function to function.

1	- _BT_OK	Success
0	- _BT_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BT_ERR_NOT_OPEN	The adapter is not powered(see btPower).
-2	- _BT_ERR_ALREADY_OPEN	The adapter is already turned on.
-4	- _BT_ERR_NOT_FOUND	Could not find device.
-5	- _BT_ERR_NO_ADAP	Could not find adapter.
-6	- _BT_ERR_INVAL_ADAP	Invalid adapter provided.
-7	- _BT_ERR_NO_RES	Could not allocate resources.
-8	- _BT_ERR_INVAL	Invalid argument.
-9	- _BT_ERR_NODATA	No valid data found.
-10	- _BT_ERR_ALREADY_PAIRED	Already paired.
-12	- _BT_ERR_BUSY	Busy
-13	- _BT_ERR_STATE	Invalid state.
-14	- _BT_ERR_CON_FAILED	Connection failed.
-15	- _BT_ERR_NOT_AVAIL	Not available
-16	- _BT_ERR_NOT_PERM	Not permitted.
-17	- _BT_ERR_TIMEOUT	Timeout
-18	- _BT_ERR_AUTH	Authentication failed.
-19	- _BT_ERR_NOT_CONNECTED	Not connected.
-99	- _BT_ERR_GENERIC	Unknown error.

4.2.7.2 btWaitEvent (Function)

```
Architecture:       NX32L
Device support:     NX-200, NX-400, NX-900
Firmware version:   1.10.00
```

This function will wait until a Bluetooth event is raised. It has an optional timeout.

An event is a notification that something has happened on the Bluetooth interface.

The events are queued until btEventDone has been called.
For some events, an appropriate action should be done before calling btEventDone, as described in the table:

Event#	Event	Action needed	Action
1	An incoming connection has been received.	Yes	Call btHandleSerialPortIncomingConnection to handle the serial port.
2	A pairing request have been received.	Yes	Call btHandlePairingRequest to find more information about the request and call btSendPairResponse to send the response.
3	A new device has been found.	No	Call btDeviceInfo to find more information about the device. Note that this event is not generated for known devices, e.g. paired or connected devices, or devices that have been found previously.
4	A known device has been lost.	No	None.
16	A BLE notification has been received.	Yes	Call btleHandleNotification to get the value from the notification.

btWaitEvent notifies the application of the oldest queued event.

This means that until btEventDone is called, btWaitEvent will continue to report the same event.

Note: waiting for an event using this function will block the calling thread.

Input:

timeout : INT (-1,0..32767) *(default -1)*
Timeout period in milliseconds to wait.
0 - Disable timeout. The function will return immediately.
-1 - Wait forever. The function will only return when data is received.

Output:

dev : STRING
The address of the device the event is related to.

Returns: INT

16	- _BT_EVENT_NOTIFY	A BLE notification has been received.
4	- _BT_EVENT_DEV_LOST	A known device has been lost.
3	- _BT_EVENT_DEV_FOUND	A new device has found.
2	- _BT_EVENT_PAIR_REQ	A pairing request have been received.
1	- _BT_EVENT_INCOMING	An incoming connection has been received.
0	- _BT_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BT_ERR_NOT_OPEN	The adapter is not powered(see btPower).
-5	- _BT_ERR_NO_ADAP	Could not find adapter.
-8	- _BT_ERR_INVAL	Invalid argument.
-9	- _BT_ERR_NODATA	No valid data found.
-17	- _BT_ERR_TIMEOUT	Timeout

Declaration:

```
FUNCTION btWaitEvent : INT;
VAR_INPUT
   timeout  : INT := -1;
```

```
   dev      : ACCESS STRING;
END_VAR;
```

Example:

```
...
THREAD_BLOCK btThread
VAR
   rc      : INT;
   address : STRING;
   ch      : SINT;
   pass    : DINT;
END_VAR;
   WHILE TRUE DO
      //
      rc := btWaitEvent(timeout:=10000, dev := address);
      IF rc <> _BT_ERR_TIMEOUT THEN
         DebugFmt(message:="event \1: "+address, v1:=rc);
         CASE rc OF
         _BT_EVENT_INCOMING:
            rc := btHandleSerialPortIncomingConnection(ch := ch, port :=
port);
            DebugFmt(message:="btHandleSerialPortIncomingConnection(\2) : \1,
\3", v1:=rc, v2:=ch, v3:=port);
         _BT_EVENT_DEV_FOUND:
            DebugFmt(message:="Found "+address);
         _BT_EVENT_DEV_LOST:
            DebugFmt(message:="Lost "+address);
         _BT_EVENT_PAIR_REQ:
            DebugFmt(message:="Requested confirm for "+address);
            rc := btHandlePairRequest(passkey:=pass);
            DebugFmt(message:="Pass: \4, \1", v1:=rc, v4:=pass);
            rc := guiShowMessage(message:="Is the pairing code
"+dintToStr(v:=pass)+" correct?", type := 2, timeout := 20);
            DebugFmt(message:="guiShowMessage: \1", v1:=rc);
            // Accept if "Yes" was pressed
            rc := btSendPairResponse(accept:=(rc = 3));
            DebugFmt(message:="btSendPairResponse: \1", v1:=rc);
         ELSE
            DebugFmt(message:="unknown event: \1", v1:=rc);
         END_CASE;
         btEventDone();
      END_IF;

   END_WHILE;
END_THREAD_BLOCK;
...
```

4.2.7.3 btEventDone (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900
Firmware version: 1.10.00

This function will make the current event as done, allowing btWaitEvent to find the next event.

Input:

None.

Returns: INT

1	- _BT_OK	Success
0	- _BT_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BT_ERR_NOT_OPEN	The adapter is not powered(see btPower).
-9	- _BT_ERR_NODATA	No valid event found to mark as done.

Declaration:

```
FUNCTION btEventDone : INT;
```

Example:

```
...
THREAD_BLOCK btThread
VAR
   rc      : INT;
   address : STRING;
   ch      : SINT;
   pass    : DINT;
END_VAR;
   WHILE TRUE DO
      //
      rc := btWaitEvent(timeout:=10000, dev := address);
      IF rc <> _BT_ERR_TIMEOUT THEN
         DebugFmt(message:="event \1: "+address, v1:=rc);
         CASE rc OF
         _BT_EVENT_INCOMING:
            rc := btHandleSerialPortIncomingConnection(ch := ch, port :=
port);
            DebugFmt(message:="btHandleSerialPortIncomingConnection(\2) : \1,
\3", v1:=rc, v2:=ch, v3:=port);
         _BT_EVENT_DEV_FOUND:
            DebugFmt(message:="Found "+address);
         _BT_EVENT_DEV_LOST:
            DebugFmt(message:="Lost "+address);
         _BT_EVENT_PAIR_REQ:
            DebugFmt(message:="Requested confirm for "+address);
            rc := btHandlePairRequest(passkey:=pass);
            DebugFmt(message:="Pass: \4, \1", v1:=rc, v4:=pass);
            rc := guiShowMessage(message:="Is the pairing code
"+dintToStr(v:=pass)+" correct?", type := 2, timeout := 20);
            DebugFmt(message:="guiShowMessage: \1", v1:=rc);
            // Accept if "Yes" was pressed
            rc := btSendPairResponse(accept:=(rc = 3));
            DebugFmt(message:="btSendPairResponse: \1", v1:=rc);
         ELSE
            DebugFmt(message:="unknown event: \1", v1:=rc);
         END_CASE;
         btEventDone();
      END_IF;

   END_WHILE;
END_THREAD_BLOCK;
...
```

4.2.7.4 btPower (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900
Firmware version: 1.10.00

This function turns the Bluetooth adapter on and off.

Input:

power : BOOL *(default ON)*
If the power should be turned on or off

Returns: **INT**

1	- _BT_OK	Success
0	- _BT_ERR_NOT_SUPPORTED	The API is not supported.
-5	- _BT_ERR_NO_ADAP	Could not find adapter.
-13	- _BT_ERR_STATE	The adapter cannot be started.

Declaration:

```
FUNCTION btPower : INT;
VAR_INPUT
   power  : BOOL := ON;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

   // Turn on power
   btPower();

BEGIN
  ...
END;

END_PROGRAM;
```

4.2.7.5 btNameSet (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900
Firmware version: 1.10.00

This function will set the name to show to other devices when discoverable.
By default, the name is "RTCU" followed by the serial number.

Input:

name : STRING
The wanted name. If longer than 248 characters, it will be truncated.

Returns: INT

1	- _BT_OK	Success
0	- _BT_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BT_ERR_NOT_OPEN	The adapter is not powered(see btPower).
-5	- _BT_ERR_NO_ADAP	Could not find adapter.

Declaration:

```
FUNCTION btNameSet : INT;
VAR_INPUT
   name     : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

   // Set name to "Master"
   btNameSet(name := "Master");

BEGIN
  ...
END;

END_PROGRAM;
```

4.2.7.6 btAdapterAddressGet (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900
Firmware version: 1.10.00

This function will retrieve the MAC address of the Bluetooth adapter.

Input:
None.

Output:
address : STRING
The address of the adapter.

Returns: INT

1	- _BT_OK	Success
0	- _BT_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BT_ERR_NOT_OPEN	The adapter is not powered(see btPower).
-5	- _BT_ERR_NO_ADAP	Could not find adapter.

Declaration:

```
FUNCTION btAdapterAddressGet : INT;
VAR_INPUT
   address  : ACCESS STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc        : INT;
   address   : STRING;
END_VAR;

// Turn on Bluetooth module
btPower();

BEGIN
   ...
   rc:=btAdapterAddressGet(address:=address);
   DebugFmt(message:="btGetAdapterAddress: \1, " + address, v1:=rc);
   ...
END;

END_PROGRAM;
```

4.2.7.7 btMakeDiscoverable (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900
Firmware version: 1.10.00

This function can make the device discoverable, i.e. visible for other devices.
With this function, it is possible to make the device visible for a period of time when e.g. a DIP switch is toggled, an SMS is received or a button is pushed on a menu.
Alternatively the device can be kept visible at all times.

Input:

discoverable : BOOL (*default TRUE*)
Set to TRUE to make the device visible, set to FALSE to make the device stop being visible immediately.

timeout : DINT *(default 60)*
The timeout in seconds before turning invisible again. If set to 0, it will stay visible until btMakeDiscoverable is called with discoverable set to FALSE.

Returns: INT

1	- _BT_OK	Success
0	- _BT_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BT_ERR_NOT_OPEN	The adapter is not powered(see btPower).
-5	- _BT_ERR_NO_ADAP	Could not find adapter.
-17	- _BT_ERR_TIMEOUT	Timeout.

Declaration:

```
FUNCTION btMakeDiscoverable : INT;
VAR_INPUT
```

```
   timeout         : DINT := 60;
   discoverable    : BOOL := TRUE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Turn on the Bluetooth library
btPower();

BEGIN
   ...
   // Make device visible for 30 seconds
   btMakeDiscoverable(timeout := 30);
   ...
END;

END_PROGRAM;
```

4.2.7.8 btIsDiscoverable (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900
Firmware version: 1.10.00

This function checks if the device is discoverable, i.e. visible to other devices.

Input:
None.

Returns: **INT**

1 -	The device is visible
0 - _BT_ERR_NOT_SUPPORTED	The device is not visible or the API is not supported.
-1 - _BT_ERR_NOT_OPEN	The adapter is not powered(see btPower).
-5 - _BT_ERR_NO_ADAP	Could not find adapter.

Declaration:

```
FUNCTION btIsDiscoverable : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Turn on the Bluetooth library
btPower();

BEGIN
   ...
   IF btIsDiscoverable() = 1 THEN
      ...
   END_IF;
```

```
   .
   ...
END;

END_PROGRAM;
```

4.2.7.9 btScanStart (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900
Firmware version: 1.10.00

This function makes the adapter start scanning for other devices.
It will find all the paired devices as well as the devices that match the provided parameters.
It is also used to receive BLE advertisement data.

Input:

transport : SINT (0..2) ***(default 0)***
The type of device to search for:

0	Any kind
1	Classic Bluetooth.
2	BLE.

UUID : STRING
Scan for a specific profile that has this UUID. Note that some devices may not have all profiles available to unpaired and unconnected devices.
The official UUIDs can be found in the Bluetooth specification (https://www.bluetooth.com/specifications/assigned-numbers/service-discovery).

RSSI : INT ***(default 0)***
The required signal strength in dB.

pathloss: INT ***(default 0)***
The device must advertise TX Power and have a computed pathloss less than this value.

timeout : DINT ***(default 0)***
The timeout in seconds before stopping the scan automatically. If set to 0, it will keep scanning until btScanStop is called.

Returns: INT

1 - _BT_OK	Success
0 - _BT_ERR_NOT_SUPPORTED	The API is not supported.
-1 - _BT_ERR_NOT_OPEN	The adapter is not powered(see btPower).
-5 - _BT_ERR_NO_ADAP	Could not find adapter.
-12 - _BT_ERR_BUSY	Adapter is already scanning.
-17 - _BT_ERR_TIMEOUT	Failed to start scan withing the expected time.

Declaration:

```
FUNCTION btScanStart : INT;
VAR_INPUT
   transport   : SINT   := 0;
   UUID        : STRING := "";
   RSSI        : INT    := 0;
   pathloss    : INT    := 0;
   timeout     : DINT   := 0;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
// These are the local variables of the program block
VAR
   rc      : INT;
   idx     : INT;
   address : STRING;
   devInfo : btDeviceInfo;
   i       : INT;
END_VAR;

   rc := btPower();
   DebugFmt(message:="btPower: \1 ", v1:=rc);

BEGIN
   // If not scanning; List the found devices and start a new scan
   IF btIsScanning() <> 1 THEN
      rc := _BT_OK;
      idx:=1;
      WHILE rc >= _BT_OK DO
         rc := btDeviceGet(idx:=idx, dev := address);
         IF rc >= _BT_OK THEN
            DebugFmt(message:="Address(\2): \1 , " + address, v1:=rc,
v2:=idx);

            devInfo(dev := address);
            DebugFmt(message:="  " + devInfo.name+" , Class: \4, type: \3,
connected: \1, tx power: \2",
               v2:=devInfo.tx_power, v4:=devInfo.clss, v3:=devInfo.addr_type,
v1:=SINT(devInfo.connected));
            DebugFmt(message:="  status: \1, paired: \2, trusted: \3",
v1:=devInfo.status,
               v2:=INT(devInfo.paired), v3:=INT(devInfo.trusted));
            DebugFmt(message:="  Profiles: \1", v1:=devInfo.profiles);

            FOR i:= 1 TO 16 DO
               IF devInfo.uuid[i] <> "" THEN
                  DebugFmt(message:="   [\1]: " + devInfo.uuid[i], v1:=i);
               END_IF;
            END_FOR;
         END_IF;

         idx := idx+1;
      END_WHILE;
      DebugFmt(message:="btDeviceGet: \1 ", v1:=rc);

      rc := btScanStart(timeout := 5, transport := 1);
      DebugFmt(message:="btScanStart: \1 ", v1:=rc);
   ELSE
      Sleep(delay:=100);
   END_IF;

END;

END_PROGRAM;
```

4.2.7.10 btScanStop (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900
Firmware version: 1.10.00

This function makes the adapter stop scanning for other devices.

Input:
None.

Returns: **INT**

1	- _BT_OK	Success
0	- _BT_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BT_ERR_NOT_OPEN	The adapter is not powered(see btPower).
-5	- _BT_ERR_NO_ADAP	Could not find adapter.
-13	- _BT_ERR_STATE	Scanning is already stopped.

Declaration:

```
FUNCTION btScanStop : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Turn on the Bluetooth library
btPower();

BEGIN
   ...
   // Stop scanning for devices
   btScanStop();
   ...
END;

END_PROGRAM;
```

4.2.7.11 btIsScanning (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900
Firmware version: 1.10.00

This function checks if the adapter is scanning for other devices.

Input:
None.

Returns: **INT**

1	-	The adapter is scanning.

0 - _BT_ERR_NOT_SUPPORTED	The adapter is not scanning or the API is not supported.
-1 - _BT_ERR_NOT_OPEN	The adapter is not powered(see btPower).
-5 - _BT_ERR_NO_ADAP	Could not find adapter.

Declaration:

```
FUNCTION btIsScanning : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
// These are the local variables of the program block
VAR
   rc      : INT;
   idx     : INT;
   address : STRING;
   devInfo : btDeviceInfo;
   i       : INT;
END_VAR;

   rc := btPower();
   DebugFmt(message:="btPower: \1 ", v1:=rc);

BEGIN
   // If not scanning; List the found devices and start a new scan
   IF btIsScanning() <> 1 THEN
      rc := _BT_OK;
      idx:=1;
      WHILE rc >= _BT_OK DO
         rc := btDeviceGet(idx:=idx, dev := address);
         IF rc >= _BT_OK THEN
            DebugFmt(message:="Address(\2): \1 , " + address, v1:=rc,
v2:=idx);

            devInfo(dev := address);
            DebugFmt(message:="  " + devInfo.name+" , Class: \4, type: \3,
connected: \1, tx power: \2",
               v2:=devInfo.tx_power, v4:=devInfo.clss, v3:=devInfo.addr_type,
v1:=SINT(devInfo.connected));
            DebugFmt(message:="  status: \1, paired: \2, trusted: \3",
v1:=devInfo.status,
               v2:=INT(devInfo.paired), v3:=INT(devInfo.trusted));
            DebugFmt(message:="  Profiles: \1", v1:=devInfo.profiles);

            FOR i:= 1 TO 16 DO
               IF devInfo.uuid[i] <> "" THEN
                  DebugFmt(message:="   [\1]: " + devInfo.uuid[i], v1:=i);
               END_IF;
            END_FOR;
         END_IF;

         idx := idx+1;
      END_WHILE;
      DebugFmt(message:="btDeviceGet: \1 ", v1:=rc);

      rc := btScanStart(timeout := 5, transport := 1);
      DebugFmt(message:="btScanStart: \1 ", v1:=rc);
   ELSE
      Sleep(delay:=100);
```

```
   END_IF;

END;

END_PROGRAM;
```

4.2.7.12 btDeviceGet (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900
Firmware version: 1.10.00

This function will retrieve the MAC address of a known remote Bluetooth device.
This will include both already known devices such as paired devices and connected BLE devices, as well as new devices found during scanning (see btScanStart).
The known devices are kept across reset, until btDeviceRemove is used to remove them.
To list all the known devices, keep calling this function with incrementing index until it returns -4 (_BT_ERR_NOT_FOUND).
Note: Scanning must be stopped for this function to work.

Input:

idx: INT (1..150) ***(default 1)***
The index of the device to get the address of.

Output:

dev : STRING
The address of the device.

Returns: INT

1 - _BT_OK	Success
0 - _BT_ERR_NOT_SUPPORTED	The API is not supported.
-1 - _BT_ERR_NOT_OPEN	The adapter is not powered(see btPower).
-4 - _BT_ERR_NOT_FOUND	No device is available at this or higher indexes.
-5 - _BT_ERR_NO_ADAP	Could not find adapter.

Declaration:

```
FUNCTION btDeviceGet : INT;
VAR_INPUT
   idx      : INT := 1;
   address  : ACCESS STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
// These are the local variables of the program block
VAR
   rc      : INT;
   idx     : INT;
   address : STRING;
   devInfo : btDeviceInfo;
   i       : INT;
```

```
END_VAR;

   rc := btPower();
   DebugFmt(message:="btPower: \1 ", v1:=rc);

BEGIN
   // If not scanning; List the found devices and start a new scan
   IF btIsScanning() <> 1 THEN
      rc := _BT_OK;
      idx:=1;
      WHILE rc >= _BT_OK DO
         rc := btDeviceGet(idx:=idx, dev := address);
         IF rc >= _BT_OK THEN
            DebugFmt(message:="Address(\2): \1 , " + address, v1:=rc,
v2:=idx);

            devInfo(dev := address);
            DebugFmt(message:="  " + devInfo.name+" , Class: \4, type: \3,
connected: \1, tx power: \2",
               v2:=devInfo.tx_power, v4:=devInfo.clss, v3:=devInfo.addr_type,
v1:=SINT(devInfo.connected));
            DebugFmt(message:="  status: \1, paired: \2, trusted: \3",
v1:=devInfo.status,
               v2:=INT(devInfo.paired), v3:=INT(devInfo.trusted));
            DebugFmt(message:="  Profiles: \1", v1:=devInfo.profiles);

            FOR i:= 1 TO 16 DO
               IF devInfo.uuid[i] <> "" THEN
                  DebugFmt(message:="   [\1]: " + devInfo.uuid[i], v1:=i);
               END_IF;
            END_FOR;
         END_IF;

         idx := idx+1;
      END_WHILE;
      DebugFmt(message:="btDeviceGet: \1 ", v1:=rc);

      rc := btScanStart(timeout := 5, transport := 1);
      DebugFmt(message:="btScanStart: \1 ", v1:=rc);
   ELSE
      Sleep(delay:=100);
   END_IF;

END;

END_PROGRAM;
```

4.2.7.13 btDeviceInfo (Functionblock)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900
Firmware version: 1.10.00

The btDeviceInfo function block is used to read information about a remote device.

Input:

dev : STRING
The address of the device to get information about, retrieved from either btDeviceGet or from the _BT_EVENT_DEV_FOUND event on btWaitEvent.

Output:

Status : INT
The status of the function:

1	- _BT_OK	The information has been retrieved and is valid.
0	- _BT_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BT_ERR_NOT_OPEN	The adapter is not powered(see btPower).
-4	- _BT_ERR_NOT_FOUND	Could not find device.
-5	- _BT_ERR_NO_ADAP	Could not find adapter.

name : STRING
The name of the device.

paired : BOOL
TRUE if the device is paired. Note that if the pairing has been removed on the remote device, this will still report TRUE, as it reflects the local status.

trusted : BOOL
TRUE if the device is trusted (this currently happens automatically when it is paired).

clss : DINT
The Class Of Device(COD) of the device.

addr_type : SINT
The type of address. 0 =Classic Bluetooth, 1 = BLE Public address, 2 = BLE Random address.

connected : BOOL
TRUE if the device is currently connected.

tx_power : SINT
The transmit power advertised by the device in dB.

RSSI : INT
The signal strength of the device in dB. This value is only valid while scanning, otherwise it is set to 0.

profiles : INT
The number of supported profiles reported by the device.

UUID[n] : STRING
The UUID of the supported profiles.
The official UUIDs can be found in the Bluetooth specification (https://www.bluetooth.com/specifications/assigned-numbers/service-discovery).

Declaration:

```
FUNCTION_BLOCK btDeviceInfo;
VAR_INPUT
   dev      : STRING;
END_VAR;
VAR_OUTPUT
   status   : INT;
   name     : STRING;
```

```
   paired   : BOOL;
   trusted  : BOOL;
   clss     : DINT;
   addr_type: SINT;
   connected: BOOL;
   tx_power : SINT;
   RSSI     : INT;
   profiles : INT;
   UUID     : ARRAY [1..32] OF STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
// These are the local variables of the program block
VAR
   rc      : INT;
   idx     : INT;
   address : STRING;
   devInfo : btDeviceInfo;
   i       : INT;
END_VAR;

   rc := btPower();
   DebugFmt(message:="btPower: \1 ", v1:=rc);

BEGIN
   // If not scanning; List the found devices and start a new scan
   IF btIsScanning() <> 1 THEN
      rc := _BT_OK;
      idx:=1;
      WHILE rc >= _BT_OK DO
         rc := btDeviceGet(idx:=idx, dev := address);
         IF rc >= _BT_OK THEN
            DebugFmt(message:="Address(\2): \1 , " + address, v1:=rc,
v2:=idx);

            devInfo(dev := address);
            DebugFmt(message:="  " + devInfo.name+" , Class: \4, type: \3,
connected: \1, tx power: \2",
               v2:=devInfo.tx_power, v4:=devInfo.clss, v3:=devInfo.addr_type,
v1:=SINT(devInfo.connected));
            DebugFmt(message:="  status: \1, paired: \2, trusted: \3",
v1:=devInfo.status,
               v2:=INT(devInfo.paired), v3:=INT(devInfo.trusted));
            DebugFmt(message:="  Profiles: \1", v1:=devInfo.profiles);

            FOR i:= 1 TO 16 DO
               IF devInfo.uuid[i] <> "" THEN
                  DebugFmt(message:="   [\1]: " + devInfo.uuid[i], v1:=i);
               END_IF;
            END_FOR;
         END_IF;

         idx := idx+1;
      END_WHILE;
      DebugFmt(message:="btDeviceGet: \1 ", v1:=rc);

      rc := btScanStart(timeout := 5, transport := 1);
      DebugFmt(message:="btScanStart: \1 ", v1:=rc);
   ELSE
```

```
      Sleep(delay:=100);
   END_IF;

END;

END_PROGRAM;
```

4.2.7.14 btDevicePair (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900
Firmware version: 1.10.00

This function makes the adapter start pairing with a remote device.

Depending on the status of the remote device, it may trigger a _BT_EVENT_PAIR_REQ event, which must be handled before the pairing is successful.
To be able to handle the event, btWaitEvent must either be called from a different thread than btDevicePair or the auto_accept parameter must be set to TRUE, which will automatically send the confirmation for the device without triggering the event.

If the device does not need confirmation, btDevicePair can be called from the same thread as btWaitEvent without problems.

The PIN code is used for legacy devices that typically requires a fixed 4-digit PIN code such as "0000" or "1234", which is described in the documentation for it.

Input:

dev : STRING
The address of the device to pair with.

auto_accept : BOOL *(default FALSE)*
Set to true to automatically accept the device instead of triggering the _BT_EVENT_PAIR_REQ event.
The passkey will not be made available in btHandlePairRequest and can therefore not be compared with the passkey on the remote device, which must instead just assume that it matches.

PIN : STRING
The PIN code to use for legacy pairing.

Returns: INT

1	- _BT_OK	Pairing was successful
0	- _BT_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BT_ERR_NOT_OPEN	The adapter is not powered(see btPower).
-4	- _BT_ERR_NOT_FOUND	Could not find device.
-10	- _BT_ERR_ALREADY_PAIRED	Already paired.
-12	- _BT_ERR_BUSY	The adapter is busy.
-13	- _BT_ERR_STATE	The pairing triggers the _BT_EVENT_PAIR_REQ event, but btDevicePair is called from the same thread as btWaitEvent.
-14	- _BT_ERR_CON_FAILED	Failed to connect to device.
-17	- _BT_ERR_TIMEOUT	Timeout
-18	- _BT_ERR_AUTH	Authentication failed. The remote device may have canceled or rejected the pairing.

-19	- _BT_ERR_NOT_CONNECTED	Not connected.

Declaration:

```
FUNCTION btDevicePair : INT;
VAR_INPUT
   dev         : STRING;
   auto_accept : BOOL := FALSE;
   PIN         : STRING := "";
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc      : INT;
   address : STRING;
END_VAR;

// Turn on the Bluetooth library
btPower();

BEGIN
   ...
   // Pair to device
   rc := btDevicePair(dev:=address, pin:="0000");
   DebugFmt(message:="btDevicePair "+address+": \1 ", v1:=rc);
   ...
END;

END_PROGRAM;
```

4.2.7.15 btDeviceRemove (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900
Firmware version: 1.10.00

This function is used to remove remote devices, e.g. to remove the pairing.
When a device has been removed, it is necessary to run a scan(btStartScan) to find it again.

Input:

dev : STRING
The address of the device to remove. If empty all devices will be removed.

Returns: INT

1	- _BT_OK	Pairing was successful
0	- _BT_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BT_ERR_NOT_OPEN	The adapter is not powered(see btPower).
-4	- _BT_ERR_NOT_FOUND	Could not find device.

Declaration:

```
FUNCTION btDeviceRemove : INT;
VAR_INPUT
   dev      : STRING := "";
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc      : INT;
END_VAR;

// Turn on the Bluetooth library
btPower();

// Remove all devices
rc := btDeviceRemove();
DebugFmt(message:="btDeviceRemove: \1 ", v1:=rc);
...

BEGIN
   ...
END;

END_PROGRAM;
```

4.2.7.16 btHandlePairRequest (Function)

Architecture: **NX32L**
Device support: **NX-200, NX-400, NX-900**
Firmware version: **1.10.00**

This function is used to read the information about a pairing request.
Based on this information and the device address provided by btWaitEvent, it must be decided if the pairing should be approved and btSendPairResponse should be called with the result.

Input:

None.

Output:

passkey : DINT
The 6-digit passkey to confirm. It must typically be compared with the one shown on the remote device. This can e.g. be done by showing it on the display or by sending an SMS. If enough security is applied at other levels(e.g. password on higher level protocols), this value can also be ignored and all devices can be accepted.

Returns: INT

1 - _BT_OK	Success
0 - _BT_ERR_NOT_SUPPORTED	The API is not supported.
-1 - _BT_ERR_NOT_OPEN	The adapter is not powered(see btPower).
-9 - _BT_ERR_NODATA	The request is not valid. No pair request event is pending or it has expired.

Declaration:

```
FUNCTION btHandlePairRequest : INT;
VAR_INPUT
   passkey  : ACCESS DINT;
END_VAR;
```

Example:

```
...
THREAD_BLOCK btThread
VAR
   rc      : INT;
   address : STRING;
   ch      : SINT;
   pass    : DINT;
END_VAR;
   WHILE TRUE DO
      //
      rc := btWaitEvent(timeout:=10000, dev := address);
      IF rc <> _BT_ERR_TIMEOUT THEN
         DebugFmt(message:="event \1: "+address, v1:=rc);
         CASE rc OF
         _BT_EVENT_INCOMING:
            rc := btHandleSerialPortIncomingConnection(ch := ch, port :=
port);
            DebugFmt(message:="btHandleSerialPortIncomingConnection(\2) : \1,
\3", v1:=rc, v2:=ch, v3:=port);
         _BT_EVENT_DEV_FOUND:
            DebugFmt(message:="Found "+address);
         _BT_EVENT_DEV_LOST:
            DebugFmt(message:="Lost "+address);
         _BT_EVENT_PAIR_REQ:
            DebugFmt(message:="Requested confirm for "+address);
            rc := btHandlePairRequest(passkey:=pass);
            DebugFmt(message:="Pass: \4, \1", v1:=rc, v4:=pass);
            rc := guiShowMessage(message:="Is the pairing code
"+dintToStr(v:=pass)+" correct?", type := 2, timeout := 20);
            DebugFmt(message:="guiShowMessage: \1", v1:=rc);
            // Accept if "Yes" was pressed
            rc := btSendPairResponse(accept:=(rc = 3));
            DebugFmt(message:="btSendPairResponse: \1", v1:=rc);
         ELSE
            DebugFmt(message:="unknown event: \1", v1:=rc);
         END_CASE;
         btEventDone();
      END_IF;

   END_WHILE;
END_THREAD_BLOCK;
...
```

4.2.7.17 btSendPairResponse (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900
Firmware version: 1.10.00

This function sends the response to a pairing request.

Input:

accept : BOOL *(default TRUE)*
Set to TRUE to accept the pairing, FALSE to reject it.

Returns: INT

1 - _BT_OK	Success
0 - _BT_ERR_NOT_SUPPORTED	The API is not supported.
-1 - _BT_ERR_NOT_OPEN	The adapter is not powered(see btPower).
-4 - _BT_ERR_NOT_FOUND	No pending request could be found. The pairing may have been canceled.

Declaration:

```
FUNCTION btSendPairResponse : INT;
VAR_INPUT
   accept   : BOOL := TRUE;
END_VAR;
```

Example:

```
...
THREAD_BLOCK btThread
VAR
   rc      : INT;
   address : STRING;
   ch      : SINT;
   pass    : DINT;
END_VAR;
   WHILE TRUE DO
      //
      rc := btWaitEvent(timeout:=10000, dev := address);
      IF rc <> _BT_ERR_TIMEOUT THEN
         DebugFmt(message:="event \1: "+address, v1:=rc);
         CASE rc OF
         _BT_EVENT_INCOMING:
            rc := btHandleSerialPortIncomingConnection(ch := ch, port :=
port);
            DebugFmt(message:="btHandleSerialPortIncomingConnection(\2) : \1,
\3", v1:=rc, v2:=ch, v3:=port);
         _BT_EVENT_DEV_FOUND:
            DebugFmt(message:="Found "+address);
         _BT_EVENT_DEV_LOST:
            DebugFmt(message:="Lost "+address);
         _BT_EVENT_PAIR_REQ:
            DebugFmt(message:="Requested confirm for "+address);
            rc := btHandlePairRequest(passkey:=pass);
            DebugFmt(message:="Pass: \4, \1", v1:=rc, v4:=pass);
            rc := guiShowMessage(message:="Is the pairing code
"+dintToStr(v:=pass)+" correct?", type := 2, timeout := 20);
            DebugFmt(message:="guiShowMessage: \1", v1:=rc);
            // Accept if "Yes" was pressed
            rc := btSendPairResponse(accept:=(rc = 3));
            DebugFmt(message:="btSendPairResponse: \1", v1:=rc);
         ELSE
            DebugFmt(message:="unknown event: \1", v1:=rc);
         END_CASE;
         btEventDone();
      END_IF;

   END_WHILE;
```

```
END_THREAD_BLOCK;
...
```

4.2.7.18 btSerialPortProfileConnect (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900
Firmware version: 1.10.00

This function tries to connect to a Serial Port Profile(SPP) on a remote device.

Input:

dev : STRING
The address of the device to connect to.

UUID : STRING
The UUID of the profile to connect to. If empty, the default SPP profile will be used.

Output:

port : SINT
The port number to use with serOpen.

Returns: INT

1 - _BT_OK	The connection to the port has been established.
0 - _BT_ERR_NOT_SUPPORTED	The API is not supported.
-1 - _BT_ERR_NOT_OPEN	The adapter is not powered(see btPower).
-4 - _BT_ERR_NOT_FOUND	Could not find device.
-7 - _BT_ERR_NO_RES	No serial port resources available.
-12 - _BT_ERR_BUSY	The adapter is busy.
-14 - _BT_ERR_CON_FAILED	Failed to connect to device.
-15 - _BT_ERR_NOT_AVAIL	The profile is not currently available.
-17 - _BT_ERR_TIMEOUT	Timeout

Declaration:

```
FUNCTION btSerialPortProfileConnect : INT;
VAR_INPUT
   dev      : STRING;
   UUID     : STRING := "";
   port     : ACCESS SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc      : INT;
   port    : SINT;
   address : STRING;
END_VAR;

// Turn on the Bluetooth library
btPower();
```

```
BEGIN
   ...
   // Connect to SPP on <address>. The port number is stored in <port>
   rc := btSerialPortProfileConnect(dev:=address, port:=port);
   DebugFmt(message:="btSerialPortProfileConnect "+address+": \1 , port: \2",
v1:=rc, v2 := port);
   ...
END;

END_PROGRAM;
```

4.2.7.19 btDeviceProfileDisconnect (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900
Firmware version: 1.10.00

This function disconnects a Profile from a remote device.

Input:

dev : STRING
The address of the device to disconnect from.

UUID : STRING
The UUID of the profile to disconnect. If empty, the default SPP profile will be used.

Returns: INT

1	- _BT_OK	Success.
0	- _BT_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BT_ERR_NOT_OPEN	The adapter is not powered(see btPower).
-4	- _BT_ERR_NOT_FOUND	Could not find device.
-12	- _BT_ERR_BUSY	Profile already disconnected.

Declaration:

```
FUNCTION btDeviceProfileDisconnect : INT;
VAR_INPUT
   dev      : STRING;
   UUID     : STRING := "";
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc      : INT;
   address : STRING;
END_VAR;

// Turn on the Bluetooth library
btPower();

BEGIN
```

```
    ...
    // Disconnect from SPP on <address>.
    rc := btDeviceProfileDisconnect(dev:=address);
    DebugFmt(message:="btDeviceProfileDisconnect "+address+": \1", v1:=rc);
    ...
END;

END_PROGRAM;
```

4.2.7.20 btSerialPortProfileEnable (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900
Firmware version: 1.10.00

This function enables or disables a Serial Port Profile(SPP) service on the local device.
When remote devices then connects to the port, the _BT_EVENT_INCOMING event is triggered.

Input:

ch : SINT
The channel to listen on. A typical value for SPP is 22.

UUID : STRING
The UUID of the profile to create. If empty, the default SPP profile will be used.

enable : BOOL
Set to TRUE to enable the profile, set to FALSE to disable it.

max_con : SINT
The maximum number of simultaneous connections from different devices to the same profile.
Use 0 to allow all connections, 1 to only allow a single connection.

Returns: INT

1 - _BT_OK	Success.
0 - _BT_ERR_NOT_SUPPORTED	The API is not supported.
-1 - _BT_ERR_NOT_OPEN	The adapter is not powered(see btPower).
-4 - _BT_ERR_NOT_FOUND	Could not find device.
-7 - _BT_ERR_NO_RES	No profile resources available.
-8 - _BT_ERR_INVAL	Invalid value for max_con.

Declaration:

```
FUNCTION btSerialPortProfileEnable : INT;
VAR_INPUT
    ch        : SINT;
    UUID      : STRING := "";
    enable    : BOOL := TRUE;
    max_con   : SINT := 0;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
```

```
VAR
   rc       : INT;
END_VAR;

// Turn on the Bluetooth library
btPower();

// Enable SPP on channel 22
rc := btSerialPortProfileEnable(ch := 22, max_con := 1);
DebugFmt(message:="btSerialPortProfileEnable: \1 ", v1:=rc);
...

BEGIN
   ...
END;

END_PROGRAM;
```

4.2.7.21 btHandleSerialPortIncomingConnection (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900
Firmware version: 1.10.00

This function handles an incoming Serial Port connection from a remote device.
It is used to handle event 1 (_BT_EVENT_INCOMING) from btWaitEvent.

Input:
None.

Output:
ch : SINT
The channel the device is connecting to.

port : SINT
The port number to use with serOpen.

Returns: INT

1 - _BT_OK	The connection to the port has been established.
0 - _BT_ERR_NOT_SUPPORTED	The API is not supported.
-1 - _BT_ERR_NOT_OPEN	The adapter is not powered(see btPower).
-9 - _BT_ERR_NODATA	The connection is not valid. The other device may have disconnected again.

Declaration:

```
FUNCTION btHandleSerialPortIncomingConnection : INT;
VAR_INPUT
   ch       : ACCESS SINT;
   port     : ACCESS SINT;
END_VAR;
```

Example:

```
...
THREAD_BLOCK btThread
VAR
   rc      : INT;
   address : STRING;
   ch      : SINT;
   pass    : DINT;
END_VAR;
   WHILE TRUE DO
      //
      rc := btWaitEvent(timeout:=10000, dev := address);
      IF rc <> _BT_ERR_TIMEOUT THEN
         DebugFmt(message:="event \1: "+address, v1:=rc);
         CASE rc OF
         _BT_EVENT_INCOMING:
            rc := btHandleSerialPortIncomingConnection(ch := ch, port :=
port);
            DebugFmt(message:="btHandleSerialPortIncomingConnection(\2) : \1,
\3", v1:=rc, v2:=ch, v3:=port);
         _BT_EVENT_DEV_FOUND:
            DebugFmt(message:="Found "+address);
         _BT_EVENT_DEV_LOST:
            DebugFmt(message:="Lost "+address);
         _BT_EVENT_PAIR_REQ:
            DebugFmt(message:="Requested confirm for "+address);
            rc := btHandlePairRequest(passkey:=pass);
            DebugFmt(message:="Pass: \4, \1", v1:=rc, v4:=pass);
            rc := guiShowMessage(message:="Is the pairing code
"+dintToStr(v:=pass)+" correct?", type := 2, timeout := 20);
            DebugFmt(message:="guiShowMessage: \1", v1:=rc);
            // Accept if "Yes" was pressed
            rc := btSendPairResponse(accept:=(rc = 3));
            DebugFmt(message:="btSendPairResponse: \1", v1:=rc);
         ELSE
            DebugFmt(message:="unknown event: \1", v1:=rc);
         END_CASE;
         btEventDone();
      END_IF;

   END_WHILE;
END_THREAD_BLOCK;
...
```

4.2.7.22 btleManufacturerDataGet (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900
Firmware version: 1.10.00

This function retrieves the manufacturer data from a remote device.
The manufacturer data is custom data that is part of the 30 custom bytes in the advertising data.
Note: The manufacturer data included in the Scan Response is currently not available.

There may be multiple sets of manufacturer data in the data, so the sets are concatenated in the buffer, using the following format for each set:

Field	Size
ID 1	2 bytes, Little Endian

Data Length 1	1 byte
Data 1	Data Length 1 bytes
ID 2	2 bytes, Little Endian
...	
Data N	Data Length N bytes

The format means that if the wanted device is known to only have one set of manufacturer data, it is enough to check the ID and then just use the data directly.

Note: To be able to receive advertising data, the adapter must be scanning, using btScanStart.

Input:

dev : STRING
The address of the device to get the data from.

size : INT
The size of the buffer.

data : PTR
The buffer to store the data in. Must be at least 31 bytes long. If there are multiple sets of manufacturer data, it must be larger.

Output:

size : INT
The number of valid bytes in the buffer. If the buffer is too small, this will contain the needed size.

Returns: INT

1 - _BT_OK	Success.
0 - _BT_ERR_NOT_SUPPORTED	The API is not supported.
-1 - _BT_ERR_NOT_OPEN	The adapter is not powered(see btPower).
-4 - _BT_ERR_NOT_FOUND	Could not find device.
-7 - _BT_ERR_NO_RES	The buffer is not large enough for the data. The size parameter contains the needed size.
-8 - _BT_ERR_INVAL	Buffer is smaller than 31 bytes.
-9 - _BT_ERR_NODATA	The device has not manufacturer data

Declaration:

```
FUNCTION btleManufacturerDataGet : INT;
VAR_INPUT
   dev      : STRING;
   size     : ACCESS INT;
   data     : PTR;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc      : INT;
   sz      : INT;
   address : STRING;
   data    : ARRAY [1..32] OF SINT;
END_VAR;
```

```
// Turn on the Bluetooth library
btPower();

// Start scanning for BLE devices to receive advertising data.
btScanStart(transport:=2);

BEGIN
   ...
   // retrieve manufacturer data from the device
   sz := SIZEOF(data);
   rc := btleManufacturerDataGet(dev:=address, data:=ADDR(data), size := sz);
   DebugFmt(message:="btleManufacturerDataGet: \1, sz \2$N", v1:=rc, v2 :=
sz);
   ...
END;

END_PROGRAM;
```

4.2.7.23 btleServiceDataGet (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900
Firmware version: 1.54.00

This function retrieves the service data from a remote device.
The service data is custom data that is part of the 32 bytes in the advertising data.

A device may be sending multiple types of service data, so it may be necessary to call this function multiple times with increasing index until it returns BT_ERR_NODATA.

Note: To be able to receive advertising data, the adapter must be scanning, using btScanStart.

Input:

dev : STRING
The address of the device to get the data from.

index : INT *(default 0)*
The index of the service data to read.

size : INT
The size of the buffer.

data : PTR
The buffer to store the data in. Must be at least 31 bytes long to be able to store all normal data.

Output:

size : INT
The number of valid bytes in the buffer. If the buffer is too small, this will contain the needed size.

UUID : STRING
The UUID of the service data.

Returns: INT

1	- _BT_OK	Success.
0	- _BT_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BT_ERR_NOT_OPEN	The adapter is not powered(see btPower).
-4	- _BT_ERR_NOT_FOUND	Could not find device.
-7	- _BT_ERR_NO_RES	The buffer is not large enough for the data.
-8	- _BT_ERR_INVAL	Buffer is smaller than 31 bytes.
-9	- _BT_ERR_NODATA	The device has no service data for this index

Declaration:

```
FUNCTION btleServiceDataGet : INT;
VAR_INPUT
   dev      : STRING;
   index    : INT := 0;
   size     : ACCESS INT;
   data     : MANDATORY PTR;
   UUID     : ACCESS STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc      : INT;
   sz      : INT;
   address : STRING;
   uuid    : STRING;
   data    : ARRAY [1..32] OF SINT;
END_VAR;

// Turn on the Bluetooth library
btPower();

// Start scanning for BLE devices to receive advertising data.
btScanStart(transport:=2);

BEGIN
   ...
   // retrieve service data from the device
   sz := SIZEOF(data);
   rc := btleServiceDataGet(dev:=address, data:=ADDR(data), size := sz, UUID
:= uuid);
   DebugFmt(message:="btleServiceDataGet: \1, sz \2$N", v1:=rc, v2 := sz);
   ...
END;

END_PROGRAM;
```

4.2.7.24 btleConnect (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900
Firmware version: 1.10.00

This function connects to a Bluetooth Low Energy(BLE) device. This is needed to be able to read the characteristics and services of the device.

Input:

dev : STRING
The address of the device to connect to.

Returns: INT

1	- _BT_OK	Success.
0	- _BT_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BT_ERR_NOT_OPEN	The adapter is not powered(see btPower).
-4	- _BT_ERR_NOT_FOUND	Could not find device.
-7	- _BT_ERR_NO_RES	Could not allocate resources for connection.
-12	- _BT_ERR_BUSY	The adapter is busy.
-14	- _BT_ERR_CON_FAILED	The connection failed to be established.
-15	- _BT_ERR_NOT_AVAIL	The device does not support BLE connections.
-17	_BT_ERR_TIMEOUT	Timeout

Declaration:

```
FUNCTION btleConnect : INT;
VAR_INPUT
   dev      : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc      : INT;
   address : STRING;
END_VAR;

// Turn on the Bluetooth library
btPower();

BEGIN
   ...
   // Connect to BLE device.
   rc := btleConnect(dev:=address);
   DebugFmt(message:="btleConnect "+address+": \1", v1:=rc);
   ...
END;

END_PROGRAM;
```

4.2.7.25 btleDisconnect (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900
Firmware version: 1.10.00

This function disconnects from a Bluetooth Low Energy(BLE) device.

Input:
dev : STRING
The address of the device to disconnect from.

Returns: **INT**

1	- _BT_OK	Success.
0	- _BT_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BT_ERR_NOT_OPEN	The adapter is not powered(see btPower).
-4	- _BT_ERR_NOT_FOUND	Could not find device.

Declaration:

```
FUNCTION btleDisconnect : INT;
VAR_INPUT
   dev      : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc      : INT;
   address : STRING;
END_VAR;

// Turn on the Bluetooth library
btPower();

BEGIN
   ...
   // Disconnect from BLE device.
   rc := btleDisconnect(dev:=address);
   DebugFmt(message:="btleDisconnect "+address+": \1", v1:=rc);
   ...
END;

END_PROGRAM;
```

4.2.7.26 btleLastSeen (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900
Firmware version: 2.26.00

This function is used to determine how much time has passed since data was last received from the provided device.
This can e.g. be used to quickly detect when a BLE device that keeps sending advertising data has disappeared.
Note that scanning must be active to receive advertising data.

Input:

dev : STRING
The address of the device.

Returns: INT

>0	-	Number of seconds since the most recent data was received.
0	- _BT_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BT_ERR_NOT_OPEN	The adapter is not powered(see btPower).
-4	- _BT_ERR_NOT_FOUND	Could not find device.

Declaration:

```
FUNCTION btleLastSeen : INT;
VAR_INPUT
   dev      : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc      : INT;
   address : STRING;
END_VAR;

// Turn on the Bluetooth library
btPower();

BEGIN
   ...
   rc := btleLastSeen(dev:=address);
   IF rc > 30 THEN
      // No data has been received for over 30 seconds: The device must have
gone.
      DebugFmt(message:="Lost "+address+": \1 s", v1:=rc);
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.7.27 btleServiceGet (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900
Firmware version: 1.10.00

This function will retrieve a service from the device.
To list all the known services, keep calling this function with incrementing index until it returns -4 (_BT_ERR_NOT_FOUND).
Note that some services that are used internally, such as Generic Access(UUID 00001800-0000-1000-8000-00805f9b34fb) are not included.

Input:

dev : STRING
The address of the device to get the data from.

idx : INT
The index of the service to read.

Output:

service : INT
The ID to the service.

primary : BOOL
TRUE if this is a primary service.

UUID : STRING
The UUID of this service.

Returns: INT

1 - _BT_OK	Success.
0 - _BT_ERR_NOT_SUPPORTED	The API is not supported.
-1 - _BT_ERR_NOT_OPEN	The adapter is not powered(see btPower).
-4 - _BT_ERR_NOT_FOUND	The device or service could not be found.
-8 - _BT_ERR_INVAL	This device does not have any services.

Declaration:

```
FUNCTION btleServiceGet : INT;
VAR_INPUT
   dev      : STRING;
   idx      : INT;
   service  : ACCESS INT;
   primary  : ACCESS BOOL;
   UUID     : ACCESS STRING;
END_VAR;
```

Example:

```
...
FUNCTION getServices
VAR_INPUT
   dev     : STRING;
END_VAR;
VAR
   rc      : INT;
   rc2     : INT;
   service : INT;
   ch      : INT;
   primary : BOOL;
   UUID    : STRING;
```

```
   i       : INT;
   c       : INT;
   flags   : DINT;
END_VAR;
DebugMsg(message:="Get services for "+dev);

i := 1;
REPEAT
   rc := btleServiceGet(dev:=dev, idx:=i, service:=service, primary:=primary,
UUID:=UUID);
   DebugFmt(message:="  btleServiceGet "+dev+": \1", v1:=rc);
   IF rc = _BT_OK THEN
      DebugFmt(message:="  Service: \1, primary: \2, UUID: "+UUID,
v1:=service, v2:=INT(primary));
      c := 1;
      REPEAT

         rc2 := btleCharGet(dev:=dev, idx:=c, service:=service, char:=ch,
UUID:=UUID, flags := flags);
         DebugFmt(message:="    btleCharGet "+dev+": \1", v1:=rc2);
         IF rc2 = _BT_OK THEN
            DebugFmt(message:="    Service: \1, Char: \2, flags: \4, UUID:
"+UUID, v1:=service, v2:=ch, v4:=flags);

         END_IF;
        c := c + 1;
      UNTIL rc2 <> _BT_OK END_REPEAT;
   END_IF;
i := i + 1;
UNTIL rc <> _BT_OK END_REPEAT;

END_FUNCTION;
...
```

4.2.7.28 btleCharGet (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900
Firmware version: 1.10.00

This function will retrieve a characteristic from the device.
To list all the known characteristics, keep calling this function with incrementing index until it returns -4 (_BT_ERR_NOT_FOUND).

Input:

dev : STRING
The address of the device to get the data from.

idx : INT
The index of the characteristic to read.

service : INT
The service to read the characteristics of.

Output:

char : INT
The ID of the characteristic.

flags : DINT
A bit field with the flags for this characteristic.

Bit	Hex	Name	Description
2	02	Read	The characteristic can be read, see btleReadVal.
3	04	Write no response	The characteristic can be written without waiting for response, which allows for higher throughput. See btleWriteVal.
4	08	Write	The characteristic can be written and waits for the response. See btleWriteVal.
5	10	Notify	The characteristic supports notifications, which can trigger events when the value changes. See btleNotifyStart.

UUID : STRING
The UUID of this characteristic.

Returns: INT

1 - _BT_OK	Success.
0 - _BT_ERR_NOT_SUPPORTED	The API is not supported.
-1 - _BT_ERR_NOT_OPEN	The adapter is not powered(see btPower).
-4 - _BT_ERR_NOT_FOUND	The device or characteristic could not be found.
-8 - _BT_ERR_INVAL	This device does not have any characteristics.

Declaration:

```
FUNCTION btleCharGet : INT;
VAR_INPUT
   dev      : STRING;
   idx      : INT;
   service  : INT;
   char     : ACCESS INT;
   flags    : ACCESS DINT;
   UUID     : ACCESS STRING;
END_VAR;
```

Example:

```
...
FUNCTION getServices
VAR_INPUT
   dev      : STRING;
END_VAR;
VAR
   rc       : INT;
   rc2      : INT;
   service  : INT;
   ch       : INT;
   primary  : BOOL;
   UUID     : STRING;
   i        : INT;
   c        : INT;
   flags    : DINT;
END_VAR;
DebugMsg(message:="Get services for "+dev);

i := 1;
REPEAT
   rc := btleServiceGet(dev:=dev, idx:=i, service:=service, primary:=primary,
UUID:=UUID);
```

```
   DebugFmt(message:="  btleServiceGet "+dev+": \1", v1:=rc);
   IF rc = _BT_OK THEN
      DebugFmt(message:="  Service: \1, primary: \2, UUID: "+UUID,
v1:=service, v2:=INT(primary));
      c := 1;
      REPEAT

         rc2 := btleCharGet(dev:=dev, idx:=c, service:=service, char:=ch,
UUID:=UUID, flags := flags);
         DebugFmt(message:="    btleCharGet "+dev+": \1", v1:=rc2);
         IF rc2 = _BT_OK THEN
            DebugFmt(message:="    Service: \1, Char: \2, flags: \4, UUID:
"+UUID, v1:=service, v2:=ch, v4:=flags);

         END_IF;
        c := c + 1;
      UNTIL rc2 <> _BT_OK END_REPEAT;
   END_IF;
i := i + 1;
UNTIL rc <> _BT_OK END_REPEAT;

END_FUNCTION;
...
```

4.2.7.29 btleWriteVal (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900
Firmware version: 1.10.00

This function will write a value to a characteristic.

Input:

dev : STRING
The address of the device to write to.

service : INT
The service the characteristic belongs to.

char : INT
The characteristic to write to.

size : INT
The number of bytes to write.

data : PTR
The data to write.

Returns: **INT**

1	- _BT_OK	Success.
0	- _BT_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BT_ERR_NOT_OPEN	The adapter is not powered(see btPower).
-4	- _BT_ERR_NOT_FOUND	The device or characteristic could not be found.
-8	- _BT_ERR_INVAL	This device does not have any characteristics.
-16	- _BT_ERR_NOT_PERM	The characteristic does not support write.
-19	- _BT_ERR_NOT_CONNECTED	The device is not connected.

Declaration:

```
FUNCTION btleWriteVal : INT;
VAR_INPUT
   dev      : STRING;
   service  : INT;
   char     : INT;
   size     : INT;
   data     : PTR;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc      : INT;
   address : STRING;
   data    : ARRAY [1..60] OF SINT;
END_VAR;

// Turn on the Bluetooth library
btPower();

BEGIN
   ...
   data[1] := 45;
   data[2] := 21;
   data[3] := 10;
   data[4] := 71;
   // Send data to BLE device.
   rc := btleWriteVal(dev := address, service := 16#0c, char := 16#1a, data :=
ADDR(data), size := 4);
   DebugFmt(message:="btleWriteVal "+address+": \1", v1:=rc);
   ...
END;

END_PROGRAM;
```

4.2.7.30 btleReadVal (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900
Firmware version: 1.10.00

This function will read a value from a characteristic.

Input:

dev : STRING
The address of the device to read from.

service : INT
The service the characteristic belongs to.

char : INT
The characteristic to read from.

size : INT
The size of the buffer.

data : PTR
The buffer to read the data into.

Output:

size : INT
The number of bytes read. If the buffer is too small, this will contain the needed size.

Returns: INT

1 - _BT_OK	Success.
0 - _BT_ERR_NOT_SUPPORTED	The API is not supported.
-1 - _BT_ERR_NOT_OPEN	The adapter is not powered(see btPower).
-4 - _BT_ERR_NOT_FOUND	The device or characteristic could not be found.
-7 - _BT_ERR_NO_RES	The buffer is too small for the data. The size parameter contains the needed size.
-8 - _BT_ERR_INVAL	This device does not have any characteristics.
-16 - _BT_ERR_NOT_PERM	The characteristic does not support read.
-19 - _BT_ERR_NOT_CONNECTED	The device is not connected.

Declaration:

```
FUNCTION btleReadVal : INT;
VAR_INPUT
   dev      : STRING;
   service  : INT;
   char     : INT;
   size     : ACCESS INT;
   data     : PTR;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc      : INT;
   address : STRING;
   data    : ARRAY [1..60] OF SINT;
   size    : INT;
END_VAR;

// Turn on the Bluetooth library
btPower();

BEGIN
   ...
   size := SIZEOF(data);
   // Read data from BLE device <address>.
   rc := btleReadVal(dev := address, service := 16#0c, char := 16#1a, data :=
ADDR(data), size := size);
   DebugFmt(message:="btleReadVal "+address+": \1, size: \2", v1:=rc, v2 :=
size);
   ...
END;

END_PROGRAM;
```

4.2.7.31 btleNotifyStart (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900
Firmware version: 1.10.00

This function will make the adapter start listening for notifications for a characteristic.
When a notification occurs, the event _BT_EVENT_NOTIFY is triggered and the data can be read using btleHandleNotification.

Input:

dev : STRING
The address of the device.

service : INT
The service the characteristic belongs to.

char : INT
The characteristic to receive notifications from.

Returns: INT

1 - _BT_OK	Success.
0 - _BT_ERR_NOT_SUPPORTED	The API is not supported.
-1 - _BT_ERR_NOT_OPEN	The adapter is not powered(see btPower).
-4 - _BT_ERR_NOT_FOUND	The device or characteristic could not be found.
-8 - _BT_ERR_INVAL	This device does not have any characteristics.
-12 - _BT_ERR_BUSY	The notification is already enabled.
-16 - _BT_ERR_NOT_PERM	The characteristic does not support notifications.
-19 - _BT_ERR_NOT_CONNECTED	The device is not connected.

Declaration:

```
FUNCTION btleNotifyStart : INT;
VAR_INPUT
   dev      : STRING;
   service  : INT;
   char     : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc      : INT;
   address : STRING;
END_VAR;

// Turn on the Bluetooth library
btPower();

BEGIN
   ...
```

```
   // Start listening for notifications from BLE device <address>.
   rc := btleNotifyStart(dev := address, service := 16#18, char := 16#19);
   DebugFmt(message:="btleNotifyStart "+address+": \1", v1:=rc);
   ...
END;

END_PROGRAM;
```

4.2.7.32 btleNotifyStop (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900
Firmware version: 1.10.00

This function will make the adapter stop listening for notifications for a characteristic.

Input:

dev : STRING
The address of the device.

service : INT
The service the characteristic belongs to.

char : INT
The characteristic to stop receiving notifications from.

Returns: INT

1	- _BT_OK	Success.
0	- _BT_ERR_NOT_SUPPORTED	The API is not supported.
-1	- _BT_ERR_NOT_OPEN	The adapter is not powered(see btPower).
-4	- _BT_ERR_NOT_FOUND	The device or characteristic could not be found.
-8	- _BT_ERR_INVAL	This device does not have any characteristics.
-16	- _BT_ERR_NOT_PERM	The characteristic does not support notifications.
-19	- _BT_ERR_NOT_CONNECTED	The device is not connected.

Declaration:

```
FUNCTION btleNotifyStop : INT;
VAR_INPUT
   dev      : STRING;
   service  : INT;
   char     : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc      : INT;
   address : STRING;
END_VAR;

// Turn on the Bluetooth library
btPower();

BEGIN
```

```
    ...
    // Disconnect from BLE device <address>.
    rc := btleNotifyStop(dev := address, service := 16#18, char := 16#19);
    DebugFmt(message:="btleNotifyStop "+address+": \1", v1:=rc);
    ...
END;

END_PROGRAM;
```

4.2.7.33 btleHandleNotification (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900
Firmware version: 1.10.00

This function will read the value that belongs to a notification and is used to handle event 16 (_BT_EVENT_NOTIFY) from btWaitEvent.

Input:

size : INT
The size of the buffer.

data : PTR
The buffer to read the data into.

Output:

service : INT
The service of the notification.

char : INT
The characteristic of the notification.

size : INT
The number of bytes read. If the buffer is too small, this will contain the needed size.

Returns: INT

1 - _BT_OK	Success.
0 - _BT_ERR_NOT_SUPPORTED	The API is not supported.
-1 - _BT_ERR_NOT_OPEN	The adapter is not powered(see btPower).
-7 - _BT_ERR_NO_RES	The buffer is too small for the data. The size parameter contains the needed size.
-8 - _BT_ERR_INVAL	Invalid size provided.
-9 - _BT_ERR_NODATA	No valid event found.

Declaration:

```
FUNCTION btleHandleNotification : INT;
VAR_INPUT
    service    : ACCESS INT;
    char       : ACCESS INT;
    size       : ACCESS INT;
    data       : PTR;
END_VAR;
```

Example:

```
...
THREAD_BLOCK btThread
VAR
   rc      : INT;
   address : STRING;
   ch      : SINT;
   pass    : DINT;
   service : INT;
   chara   : INT;
   size    : INT;
   data    : ARRAY [1..10] OF SINT;
END_VAR;
   WHILE TRUE DO
      //
      rc := btWaitEvent(timeout:=10000, dev := address);
      IF rc <> _BT_ERR_TIMEOUT THEN
         DebugFmt(message:="event \1: "+address, v1:=rc);
         CASE rc OF
         _BT_EVENT_INCOMING:
            rc := btHandleSerialPortIncomingConnection(ch := ch, port :=
port);
            DebugFmt(message:="btHandleSerialPortIncomingConnection(\2) : \1,
\3", v1:=rc, v2:=ch, v3:=port);
         _BT_EVENT_DEV_FOUND:
            DebugFmt(message:="Found "+address);
         _BT_EVENT_DEV_LOST:
            DebugFmt(message:="Lost "+address);
         _BT_EVENT_PAIR_REQ:
            DebugFmt(message:="Requested confirm for "+address);
            rc := btHandlePairRequest(passkey:=pass);
            DebugFmt(message:="Pass: \4, \1", v1:=rc, v4:=pass);
            rc := guiShowMessage(message:="Is the pairing code
"+dintToStr(v:=pass)+" correct?", type := 2, timeout := 20);
            DebugFmt(message:="guiShowMessage: \1", v1:=rc);
            // Accept if "Yes" was pressed
            rc := btSendPairResponse(accept:=(rc = 3));
            DebugFmt(message:="btSendPairResponse: \1", v1:=rc);
         _BT_EVENT_NOTIFY:
            size := SIZEOF(data);
            rc := btleHandleNotification(service := service, char := chara,
size := size, data:=ADDR(data));
            DebugFmt(message:="btleHandleNotification: \1, s\2,\3, sz: \4",
v1:=rc, v2:=service, v3:=chara, v4:=size);
         ELSE
            DebugFmt(message:="unknown event: \1", v1:=rc);
         END_CASE;
         btEventDone();
      END_IF;

   END_WHILE;
END_THREAD_BLOCK;
...
```

4.2.8 cam: Camera functions

4.2.8.1 cam: Camera module

The camera library functions are used to access the optional VGA CMOS Color camera module that is available.
The picture taken is in the standard JPEG format and eight different resolutions are available.

For more information about the camera module, please consult the technical manual for the product.

The NX32L devices with USB host port has support for USB cameras that conform to the USB video device class and uses the MJPEG format.

The following functions are available:

- camOpen — Opens the camera interface.
- camClose — Closes the camera interface.
- camPresent — Queries whether the camera module is present or not.
- camGetInfo — Retrieves information about a connected camera.
- camSnapshot — Takes a snapshot with the camera module.
- camSnapshotToFile — Takes a snapshot with the camera module and saves it to a file.
- camParameterRange — Gets information about a parameter on the camera.
- camParameterGet — Reads a parameter from the camera.
- camParameterSet — Sets a parameter on the camera.
- camCaptureStart — Records video from the camera to a file.
- camCaptureStop — Stops the recording of video.
- camCaptureStatus — Queries the recording status.

4.2.8.2 camOpen (Function)

```
Architecture:       X32 / NX32 / NX32L
Device support:     MX2/DX4/AX9 pro/MX2 turbo/encore/warp, AX9 turbo,
                    NX-200, NX-400
Firmware version:   1.07 / 1.00.00
```

This opens the camera module and prepares it for use. This function must be called and returned successfully before any other camera function is called.
The camera module is connected to the RS232 port 0 or 1 of the RTCU device and will take over this port until camClose is called.

Firmware V3.10 or later is required for CAM-200 series camera support.

For NX32L devices, USB cameras are supported instead of RS232 camera modules, and use the usbHostGetCameraPort function to determine the port to open.

Input:

port : SINT (0/1, *Default: 0*)
The serial port where the camera module is connected (see serial port enumeration for X32/NX32 and usbHostGetCameraPort for NX32L).

type : SINT (1/2, *Default: 1*)
The type of camera module connected. Not used under NX32L.
1 - CAM-100 series camera module.
2 - CAM-200 series camera module.

Returns: **INT**

0 - Success.
-1 - Failed. No camera module present or not supported?
-3 - Invalid parameter.
-4 - Serial port is already used by application.

Declaration:

```
FUNCTION camOpen : INT;
VAR_INPUT
   port  : SINT := 0;
   type  : SINT := 1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Open the camera library
camOpen();

BEGIN
  ...
END;

END_PROGRAM;
```

4.2.8.3 camClose (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2/DX4/AX9 pro/MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400
Firmware version: 1.07 / 1.00.00

This closes the camera module and the library functions. To reopen the camera module again, call camOpen.

For devices using port 0 as programming port, the RS232 programming port will resume to work as the service port after calling camClose.

Input:

None

Returns: INT

0 - Success.

Declaration:

```
FUNCTION camClose : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Open the camera library
camOpen();

BEGIN
  ...
  // Close the camera library
```

```
  camClose();
  ...
END;

END_PROGRAM;
```

4.2.8.4 camPresent (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2/DX4 pro/AX9 pro/MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400
Firmware version: 1.07 / 1.00.00

The function returns information about whether the camera module is present or not.

Input:
None.

Returns: **BOOL**
TRUE: If the camera module is present.
FALSE: If the camera module is not present.

Declaration:

```
FUNCTION camPresent : BOOL;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

   // Open the camera library
   camOpen();

BEGIN
    ...
    // Test for camera module
    IF camPresent() THEN
        ...
    END_IF;
    ...
END;

END_PROGRAM;
```

4.2.8.5 camGetInfo (Functionblock)

Architecture: NX32L
Device support: NX-200, NX-400
Firmware version: 1.00.00

This functionblock returns information about a connected camera.

Input:
None.

Output:

status : INT
The status of the camera:
1 - Camera is open.
-1 - Failed. No camera module present.
-5 - Camera interface not open.

max_width : INT
The maximum width of the camera.
max_height : INT
The maximum height of the camera.
format : DINT
The ID of the image format of the camera. If the camera supports multiple formats, the format with best support will be shown.
Known formats:

Format	Decimal	Hex
MJPEG	1196444237	47504A4D

Declaration:

```
FUNCTION_BLOCK camGetInfo;
VAR_OUTPUT
   status     : INT;
   max_width  : INT;
   max_height : INT;
   format     : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   camInfo  : camGetInfo;
END_VAR;

PROGRAM test;

   // Open the camera library
   camOpen();

BEGIN
    ...
    // Get information about the camera module
    camInfo();
    IF camInfo.status > 0 THEN
        DebugFmt(message := "Max resolution: \1 x \2, format \4", v1 :=
camInfo.max_width,
                                          v2:=camInfo.max_height,
v2:=camInfo.format);
    END_IF;
    ...
END;
END_PROGRAM;
```

4.2.8.6 camSnapshot (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2/DX4/AX9 pro/MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400
Firmware version: 1.07 / 1.00.00

This function takes a snapshot picture from the camera module.
The picture taken is in the industry standard JPEG format and four different resolutions are available.

The following resolutions are available:
80 x 64 pixels. *Not supported on NX32L*
160 x 128 pixels. *Not supported on NX32L*
320 x 240 pixels.
640 x 480 pixels.

The recommended resolution is 320 x 240 pixels as it represents a good balance between buffer size, quality, and resolution.

Input:

res : INT
The resolution of the snapshot picture:
1 80 x 64 pixels.
2 160 x 128 pixels.
3 320 x 240 pixels.
4 640 x 480 pixels.

pic : PTR
The buffer for the picture snapshot.
The picture returned is in the standard JPEG-compressed format.

picsize : DINT
The size of the picture snapshot buffer.

Recommended buffer sizes:

Resolution	Buffer size
80 x 64	1..2 kB
160 x 128	3..5 kB
320 x 240	10..15 kB
640 x 480	35..45 kB

Note: the actual picture size depends on the level of JPEG compression possible on the actual picture.
If the buffer specified in too small for the actual picture. an error code will be returned (see below).

Returns: DINT

The size of the picture snapshot or:
-1 - Failed (camera removed, communication problem, camera does not support MJPEG format(see camGetInfo)).
-2 - Buffer too small.
-3 - Invalid parameter.
-4 - Camera is currently recording.
-5 - Camera interface not open.

Declaration:

```
FUNCTION camSnapshot : DINT;
VAR_INPUT
   res     : INT;
   pic     : PTR;
   picsize : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   snapshot : BOOL R_EDGE;
END_VAR;

VAR
   size    : DINT;
   filedes : FILE;
   buffer  : ARRAY[1..15360] OF SINT;
END_VAR;

PROGRAM test;

   fsMediaOpen(media := 0); // Open the filesystem
   camOpen();               // Open the camera library

BEGIN
   // Take a snapshot
   IF snapshot THEN
      // Is the camera module present?
      IF camPresent() THEN
         // Take a snapshot
         size := camSnapshot(res := 3, pic := ADDR(buffer), picsize :=
SIZEOF(buffer));
         IF size > 0 THEN
            // Write to file (standard JPEG):
            filedes := fsFileCreate(name := "PIC.JPG");
            fsFileWrite(fd := filedes, buffer := ADDR(buffer), length :=
INT(size));
            fsFileClose(fd := filedes);
         END_IF;
      END_IF;
   END_IF;
END;
END_PROGRAM;
```

4.2.8.7 camSnapshotToFile (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2/DX4/AX9 pro/MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400
Firmware version: 3.10 / 1.00.00

This function takes a snapshot picture from the camera module and save it in the file system. The picture taken is in the industry standard JPEG format and multiple different resolutions are available.

The following resolutions are available on X32/NX32:
80 x 64 pixels.
160 x 128 pixels.
320 x 240 pixels.
640 x 480 pixels.
800 x 600 pixels.
1024 x 768 pixels.
1280 x 960 pixels.
1600 x 1200 pixels.

The following basic resolutions are available on NX32L:
320 x 240 pixels.

640 x 480 pixels.
800 x 600 pixels.
1280 x 720 pixels.
1920 x 1080 pixels.

The actual resolution used depends on the connected USB camera and may be smaller than the requested resolution, if the camera does not support it.

Note:
On X32/NX32, this function is only available for CAM-200 series camera.

Input:

res : INT
The resolution of the snapshot picture.
For X32/NX32:

1	80 x 64 pixels.
2	160 x 128 pixels.
3	320 x 240 pixels.
4	640 x 480 pixels.
5	800 x 600 pixels.
6	1024 x 768 pixels.
7	1280 x 960 pixels.
8	1600 x 1200 pixels

For NX32L:

3	320 x 240 pixels.
4	640 x 480 pixels.
5	800 x 600 pixels.
6	1280 x 720 pixels.
7	1920 x 1080 pixels.

filename : STRING
Name of the file to save image as. Both absolute and relative paths are allowed.
If file exists it will be overwritten.

Returns: DINT

The size of the picture snapshot or:

-1	- Failed (camera removed, communication problem, camera does not support MJPEG format(see camGetInfo)).
-3	- Invalid parameter.
-4	- Camera is currently recording.
-5	- Camera interface not open for CAM-200 series.
-6	- Failed to write image to file

Declaration:

```
FUNCTION camSnapshotToFile : DINT;
VAR_INPUT
   res        : INT;
   filename   : PTR;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
```

```
   snapshot : BOOL R_EDGE;
END_VAR;

VAR
   size    : DINT;
END_VAR;

PROGRAM test;

   fsMediaOpen(media := 0); // Open the file system
   camOpen(type := 2);      // Open the camera library

BEGIN
   // Take a snapshot
   IF snapshot THEN
      // Is the camera module present?
      IF camPresent() THEN
         // Take a snapshot
         camSnapshotToFile(res := 3, filename := "A:\PIC.JPG");
      END_IF;
   END_IF;
END;
END_PROGRAM;
```

4.2.8.8 camParameterRange (Function)

Architecture: NX32L
Device support: NX-200, NX-400
Firmware version: 1.12.00

This function retrieves the range and other information for a parameter on a connected camera. This information can then be used to determine the parameters to use for camParameterGet.

Input:

type : INT
The type of parameter to get information about.

1 - Brightness
2 - Contrast
3 - Saturation
4 - Hue
5 - Auto gain
6 - Auto white balance
7 - Adjust white balance.
8 - Red Balance
9 - Blue balance
10 - Gamma
11 - Power line frequency filter: 0 = Disabled, 1 = 50Hz, 2 = 60Hz, 3 = Auto(if supported).
12 - Auto exposure: 0 = Auto, 1 = Manual, 2 = Shutter priority, 3 = Aperture priority.
13 - Absolute exposure. Exposure time in 100µs units.
14 - Allow dynamic frame rate when Auto exposure is Auto or Aperture priority

Output:

min_value : DINT
The minimum allowed value.

max_value : DINT
The minimum allowed value.

step : DINT
The smallest value that makes a change in the camera hardware. Changing the value with less that the step size will not make a difference.

kind : INT
The kind of parameter this is:
0 - Unknown
1 - Integer.
2 - Boolean. Set to 1 to enable, 0 to disable.
3 - Enumeration. Each value in the range have a specific value. See the type table for a description of the valid values.
4 - Trigger. The value is not used, setting this parameter triggers the specified functionality.

flags : DINT
Bit-mask with additional information about the parameter:

Bit
0 - The parameter is disabled and can not be used.
1 - The parameter is read-only.
2 - The parameter is write-only.
3 - The parameter is volatile and will change by itself.

Returns: INT

0 - The parameter info was retrieved.
-1 - Failed to get the info. The camera may not support reading this parameter.
-3 - Invalid type.
-5 - Camera interface not open.

Declaration:

```
FUNCTION camParameterRange : INT;
VAR_INPUT
   type      : INT;
   min_value : ACCESS DINT;
   max_value : ACCESS DINT;
   step      : ACCESS DINT;
   kind      : ACCESS INT;
   flags     : ACCESS DINT;
END_VAR;
```

Example:

```
// get information about the brightness:
camParameterRange(type:=1, min_value := min_val, max_value := max_val, step :=
step, kind := kind, flags := flags);
```

4.2.8.9 camParameterGet (Function)

Architecture: NX32L
Device support: NX-200, NX-400
Firmware version: 1.12.00

This function gets parameters from a connected camera.
To determine the supported types and values, the function camParameterRange can be used.

Input:

type : INT
The type of parameter to get.

1 - Brightness
2 - Contrast
3 - Saturation
4 - Hue
5 - Auto gain
6 - Auto white balance
7 - Adjust white balance.
8 - Red Balance
9 - Blue balance
10 - Gamma
11 - Power line frequency filter: 0 = Disabled, 1 = 50Hz, 2 = 60Hz, 3 = Auto(if supported).
12 - Auto exposure: 0 = Auto, 1 = Manual, 2 = Shutter priority, 3 = Aperture priority.
13 - Absolute exposure. Exposure time in 100µs units.
14 - Allow dynamic frame rate when Auto exposure is Auto or Aperture priority

Output:

value : DINT
The retrieved value. The exact meaning of this variable depends on the kind of parameter.

Returns: INT

0 - The parameter was retrieved.
-1 - Failed to get the value. The camera may not support reading this parameter.
-3 - Invalid type.
-5 - Camera interface not open.

Declaration:

```
FUNCTION camParameterGet : INT;
VAR_INPUT
   type     : INT;
   value    : ACCESS DINT;
END_VAR;
```

Example:

```
// get the brightness:
camParameterGet(type:=1, value := brightness);
```

4.2.8.10 camParameterSet (Function)

Architecture: NX32L
Device support: NX-200, NX-400
Firmware version: 1.12.00

This function sets parameters in a connected camera.
To determine the supported types and values, the function camParameterRange can be used.

Input:

type : INT
The type of parameter to set.

1 - Brightness
2 - Contrast
3 - Saturation

4 - Hue
5 - Auto gain
6 - Auto white balance
7 - Adjust white balance.
8 - Red Balance
9 - Blue balance
10 - Gamma
11 - Power line frequency filter: 0 = Disabled, 1 = 50Hz, 2 = 60Hz, 3 = Auto(if supported).
12 - Auto exposure: 0 = Auto, 1 = Manual, 2 = Shutter priority, 3 = Aperture priority.
13 - Absolute exposure. Exposure time in 100µs units.
14 - Allow dynamic frame rate when Auto exposure is Auto or Aperture priority

value : DINT
The value to set. The exact meaning of this variable depends on the kind of parameter to set.

Returns: INT

0 - The parameter was set.
-1 - Failed to set the value. The camera may not support changing this parameter.
-3 - Invalid type.
-5 - Camera interface not open.
-7 - Value is outside the valid range.

Declaration:

```
FUNCTION camParameterSet : INT;
VAR_INPUT
   type     : INT;
   value    : DINT;
END_VAR;
```

Example:

```
// set brightness to 72.
camParameterSet(type:=1, value := 72);
```

4.2.8.11 camCaptureStart (Function)

Architecture: NX32L
Device support: NX-200, NX-400
Firmware version: 1.12.00

This function begins recording video from a camera.
The video is either saved as a raw MJPEG stream, or as MJPEG inside e.g. an MKV container.
When recording to the internal drive, care must be taken not to fill it up, as it will affect the performance.

Input:

width : INT
The wanted width of the video. The actual width used may be smaller, depending on the camera.

height : INT
The wanted height of the video. The actual height used may be smaller, depending on the camera.

filename : STRING
The name of the file to store the recoding in. It is overwritten if it already exists.

duration : DINT
The number of seconds to keep recording for. If set to 0, the recoding continues until it runs out of space or it is stopped using camCaptureStop.

format : INT *default 2*
The file format to use when saving the file:

Format	Name	Decription
0	MJPEG	Raw MJPEG stream directly from the camera.
2	MKV	Matroska container containing the MJPEG data.

Returns: INT

- 0 - The recording was started. Use camCaptureStatus to check the status of the recording.
- -1 - Failed (camera removed, communication problem, camera does not support MJPEG format(see camGetInfo)).
- -3 - Invalid parameter
- -4 - Camera is already recording.
- -5 - Camera interface not open.
- -6 - Failed to create the file.
- -7 - Invalid format provided.

Declaration:

```
FUNCTION camCaptureStart : INT;
VAR_INPUT
   width    : INT;
   height   : INT;
   filename : STRING;
   duration : DINT;
   format   : INT := 2;
END_VAR;
```

Example:

```
// Record 720p video for 10 seconds to the file A:\REC.MKV
camCaptureStart(width:=1280, height := 720, filename := "A:\REC.MKV",
duration:=10, format:=2);
```

4.2.8.12 camCaptureStop (Function)

Architecture: NX32L
Device support: NX-200, NX-400
Firmware version: 1.12.00

This function stops the recording of video from a camera, started with camCaptureStart.

Input:
None.

Returns: INT

- 0 - The recording is stopped.

-1 - The recording was not stopped. It may still be writing the data to the storage media. Use camCaptureStatus to determine when the writing is done.
-5 - Camera interface not open.

Declaration:

```
FUNCTION camCaptureStop : INT;
```

Example:

```
// Stop the recording
camCaptureStop();
```

4.2.8.13 camCaptureStatus (Function)

Architecture: NX32L
Device support: NX-200, NX-400
Firmware version: 1.12.00

This function reports if the camera is recording.
It is useful to determine when the recording is done, so it can be moved or transmitted.

Input:

None.

Returns: INT

2 - The recording is stopped, the file is still being written.
1 - The recording is running.
0 - The recording is stopped.
-1 - The recording failed.
-5 - Camera interface not open.

Declaration:

```
FUNCTION camCaptureStatus : INT;
```

Example:

```
// Get the status of the recording
camCaptureStatus();
```

4.2.9 can: CAN functions

4.2.9.1 can: CAN functions

CAN (Controller Area Network) is a serial bus system that was developed for robust and fast communication in harsh environments like those realised in the automotive industry. The network is a multi-master network where all the devices attached to it receive the transmitted message and then decides if it is relevant. The hardware controller automatically retransmits messages that were interrupted due to data collision etc. Within this fault protection is also implemented a message priority so that messages with higher priority will be transmitted before those with a lower priority.

The CAN network is divided into layers. This starts with the physical layer (the network hardware connection). Then there are the higher layers (software) which implement the protocols that are application-specific. However, various standards have been developed - each aimed at a specific area. The J1939 and FMS standards are, for example, aimed for the automotive industry.
CAN messages all consist (from the user's point of view) of a message identifier (ID) and up to 8 data bytes. The ID is then divided into two groups: one with an 11-bit wide ID (CAN 2.0 A) called "standard identifier" and another with a 29-bit wide ID (CAN 2.0 B) called "extended identifier". Common for both types is the fact that the ID sets the message priority – the lowest ID has the highest priority.

As some nodes on the network communicate very often, there will be extensive traffic most of the time. Most of this information is not relevant, and in order not to handle irrelevant information the whole time, a need for filtering arises. The message is filtered based on the ID and is a hardware-realised filter - it discards all messages with an irrelevant ID.
The CAN implementation includes an optional logging mechanism which will automatically collect received CAN messages with a minimum of resource requirement - thus allowing a continuous collection of data while the device is performing other tasks.

The logging feature is independent and separate from the normal operations, which means that neither the logger nor the logging functions will affect sending or receiving messages normally. This means that the logging feature can be added to any application without changes to the existing code.
The filters are shared between normal operation and the logger, and as such, only the functions to create and destroy filters affect both.
The logger will ignore all new messages once it is full, so care should be taken to ensure that the logger data is moved to permanent storage at regular intervals to prevent this.
There is one logger for each CAN bus present on the device.

The following functions are used to access the CAN network.

- canOpen — Opens a CAN port.
- canClose — Closes a CAN port after use.
- canLoopBackMode — Activates loopback mode for test purposes.
- canSetWakeup — Configure CAN as a wake-up source.
- canSendMessage — Sends a CAN message.
- canReceiveMessage — Receives a CAN message.
- canReceiveX — Receives a CAN message, with timeout.
- canFilterCreate — Creates a receive filter.
- canFilterCreateOnce — Creates a receive filter that only accepts one message.
- canFilterCreateX — Creates a receive filter with advanced filter options.
- canFilterDestroy — Destroys a receive filter.
- canFilterStatus — Gets status for receive filter.
- canFlush — Clears all received messages.
- canBufferLevel — Returns the amount of buffer used.

- canLoggerSetup — Configures logging feature.
- canLoggerStart — Starts logging received messages.
- canLoggerStop — Stops logging received messages.
- canLoggerLevel — Returns the amount of space used by the logger.
- canLoggerRead — Reads the oldest message on the logger. (mode 1)
- canLoggerRead2 — Reads the oldest message on the logger. (mode 2)
- canLoggerToMemory — Moves the contents of the logger to memory.
- canFMSFilterCreate — Creates a receive filter for a FMS/J1939 message.
- canFMSGetPGN — Extracts the FMS/J1939 PGN from a CAN message ID.

The following functions are used to create a single connection to send and receive ISO-TP messages on:

- canitpOpen — Opens an ISO-TP connection via a CAN port.
- canitpClose — Closes an ISO-TP connection.
- canitpSend — Sends a message.

The following functions are used to create multiple connections for sending and receiving ISO-TP messages:

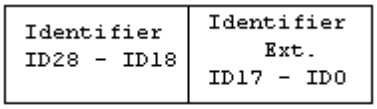
- canitpSessionCreate — Create an ISO-TP session.
- canitpSessionDestroy — Closes the ISO-TP session.
- canitpSessionSend — Sends a message via the ISO-TP session.

CAN message ID

The CAN message identifier (ID) is represented as a DINT in VPL. As the CAN functions are common for both standard and extended identifiers, an illustration below shows how the bit is organized in the two cases.

The ID consist of either 11-bit (standard) or 29-bit (extended). The standard ID is always located ID28..18, and the extended ID is located in ID17..0. This is illustrated in (a).

a) CAN message ID:



For convenience in VPL, the standard identifier is shifted down - starting from zero (decimal) (b). Extended identifiers are then represented as they are received (c).

b) Std. ID presented in VPL:

c) Ext. ID presented in VPL:

CAN termination and Write enable

The CAN bus of the device should be properly terminated according to the CAN bus standard. For devices with hardware jumpers for termination and write enable the necessary information can be found in the technical manual and for devices with software controlled jumpers the function boardSwitchSet can be used.

The CAN implementation has the following limitations:
- There is a maximum of 30 filters available for each port.
- The internal buffer can hold up to 100 received CAN messages.

4.2.9.2 canOpen (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2 pro, DX4 pro, CX1 pro-c/warp-c, MX2 turbo/encore/warp, NX-200, NX-400, LX2, LX5
Firmware version: 1.00 / 1.00.00

This will open a CAN port. Before the CAN functions can be used, the interface must be initialized and the baud rate must be set. A standard baud rate range is available for easy configuration. When the function has successfully returned, the messages can be transmitted (canSendMessage) and received (if a receive filter is created with canFilterCreate, canFilterCreateOnce, or canFilterCreateX). Once done with sending and receiving messages, the port can be closed with canClose.

Monitor mode is a way for the RTCU device to listen for and receive messages without interacting with the CAN network.
When in monitor mode, the canSendMessage is disabled and no messages can be sent. The RTCU device does not acknowledge messages received.
This means that if no other CAN device is present on the CAN network, the sender will repeat the message regularly; or if the sender is another RTCU device, that canSendMessage will return 4 (Timeout).

Note: when the CAN port is opened without having monitor enabled (and the hardware has write access, please see the technical documentation for the device), an acknowledgement is sent to the CAN network for all incoming messages as per the CAN standard.

Input:

port : SINT (1/2) ***(default 1)***
The port of the CAN bus.

baud : INT (50, 62 (62.5 kbps), 100,125,250,500,1000) ***(default 100)***
Selects the desired baud rate in Kbps.

monitor : BOOL ***(default TRUE)***
Selects monitor mode.

Returns: INT

0 - Successful.
1 - Unsupported baud rate.
2 - The baud rate is not possible at the selected CPU speed (see pmSetSpeed).
3 - The CAN bus is not present.

Declaration:

```
FUNCTION canOpen  : BOOL;
VAR_INPUT
   port    : SINT := 1;
   baud    : INT  := 100;
   monitor : BOOL := TRUE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   canin : canReceiveMessage;
   buf   : ARRAY[1..8] OF SINT;
END_VAR;

PROGRAM CANExample;
VAR
   CAN_ID : SINT;
END_VAR;

// Open can
canOpen(baud:=250);
canin(data:=ADDR(buf));

// startID: Priority=3 Reserved=1 Data page=0 PGN=00FDD6
CAN_ID := canFilterCreate(xtd:=TRUE,startID:=16#0EFDD600,length:=6);

BEGIN
   canin();
   ...
   IF canin.ready THEN
      // Message received
      ...
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.9.3 canClose (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2 pro, DX4 pro, CX1 pro-c/warp-c, MX2 turbo/encore/warp, NX-200, NX-400, LX2, LX5
Firmware version: 1.00 / 1.00.00

This closes a CAN port previously opened by canOpen. Call this function to close the CAN port when done with sending and receiving messages.

Input:

port : SINT (1/2) ***(default 1)***
The port of the CAN bus.

Returns: INT

1 - Successful.
0 - The CAN bus is not open.

Declaration:

```
FUNCTION canClose : INT;
VAR_INPUT
   port : SINT := 1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   ignition : BOOL;
END_VAR;

PROGRAM test;
VAR
   can_open : BOOL;
END_VAR;

BEGIN
   ...
   IF ignition THEN
      IF NOT can_open THEN
         IF canOpen(baud:=250) = 0 THEN
            can_open := THEN;
         ELSE
            DebugMsg(message:="CanOpen - Failed!");
         END_IF;
      END_IF;
   ELSE
      IF can_open THEN
         canClose();
         can_open := FALSE;
      END_IF;
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.9.4 canLoopBackMode (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2 pro, DX4 pro, CX1 pro-c/warp-c, MX2 turbo/encore/warp, NX-200, NX-400, LX2, LX5
Firmware version: 1.00 / 1.00.00

This activates loopback mode. This functions makes it possible to test the application configuration (send, receive, and filter) with or without a connection to a physical CAN network. The loopback mode loops all outgoing traffic back to the CAN receiver, and the transmitted message will be received if a receive filter is created with canFilterCreate, canFilterCreateOnce, or canFilterCreateX.

Input:

port : SINT (1/2) ***(default 1)***
The port of the CAN bus.

enable : BOOL ***(default FALSE)***
If true, loopback is enabled.

Note: using loopback mode requires that canOpen() has been called with the "monitor" parameter set to FALSE.

Returns: INT

1 - Successful.
0 - The CAN bus is not open.

Declaration:

```
FUNCTION canLoopBackMode : INT;
VAR_INPUT
   port   : SINT := 1;
   enable : BOOL := FALSE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   canRX : canReceiveMessage;
   buf   : ARRAY [1..8] OF SINT;
END_VAR;

PROGRAM CANExample;
VAR
   FilterID : SINT;
   rc       : INT;
END_VAR;

// Open can
canOpen(baud := 250, monitor := FALSE);
canRX(data := ADDR(buf));

canLoopBackMode(enable := ON);

// startID: Priority=3 Reserved=1 Data page=0 PGN=00FDD6
FilterID := canFilterCreate(xtd:=TRUE,startID:=16#0EFDD600,length:=6);

rc := canSendMessage(xtd:=TRUE,ID:=16#0EFDD600,data:=ADDR(buf),datasize:=8);
IF rc = 0 THEN
   DebugMsg(message:="CAN message sent");
ELSE
   DebugFmt(message:="CAN message failed (\1)",v1:=rc);
END_IF;

BEGIN
   canRX();
   ...
   IF canRX.ready THEN
      DebugMsg(message:="Message received!");
      DebugFmt(message:="canRX.xtd= \1", v1:=INT(canRX.xtd));
      DebugFmt(message:="canRX.ID= \4", v4:=canRX.ID);
      DebugFmt(message:="canRX.DataSize= \1", v1:=INT(canRX.DataSize));
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.9.5 canSendMessage (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2 pro, DX4 pro, CX1 pro-c/warp-c, MX2 turbo/encore/warp, NX-200, NX-400, LX2, LX5
Firmware version: 1.00 / 1.00.00

Send a CAN message with the defined ID and contents of a buffer. The message Identifier (ID) is by default set to be an extended ID. The maximum data size to send is limited to 8 bytes per transmission.

Note:
From firmware version 4.68 / R1.10.00 this function supports reporting of bus-error conditions.

Input:

port : SINT (1/2) *(default 1)*
The port of the CAN bus.

xtd : BOOL *(default TRUE)*
If true, the package uses the extended identifier (29 bits).

ID : DINT
The identifier of the package.

data : PTR
Address of the buffer that contains the data.

datasize : INT (1..8)
Number of bytes in buffer.

Returns: SINT

0 - Success. The Message was sent and acknowledged on the bus.
1 - The CAN bus is not open.
2 - Datasize out of range.
3 - Invalid buffer.
4 - Timeout. The message was not acknowledged on the bus.
5 - Monitor mode. Transmission not allowed.
6 - An bus-error was detected. This includes a short-circuit condition or write operation impossible (missing jumper condition).

Declaration:

```
FUNCTION canSendMessage : SINT;
VAR_INPUT
   port     : SINT := 1;
   xtd      : BOOL := TRUE;
   ID       : DINT;
   data     : PTR;
   datasize : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   canRX : canReceiveMessage;
   buf   : ARRAY [1..8] OF SINT;
END_VAR;

PROGRAM CANExample;
VAR
   FilterID : SINT;
   rc       : INT;
END_VAR;
```

```
// Open can
canOpen(baud := 250, monitor := FALSE);
canRX(data := ADDR(buf));

canLoopBackMode(enable := ON);

// startID: Priority=3 Reserved=1 Data page=0 PGN=00FDD6
FilterID := canFilterCreate(xtd:=TRUE,startID:=16#0EFDD600,length:=6);

rc := canSendMessage(xtd:=TRUE,ID:=16#0EFDD600,data:=ADDR(buf),datasize:=8);
IF rc = 0 THEN
   DebugMsg(message:="CAN message sent");
ELSE
   DebugFmt(message:="CAN message failed (\1)",v1:=rc);
END_IF;

BEGIN
   canRX();
   ...
   IF canRX.ready THEN
      DebugMsg(message:="Message received!");
      DebugFmt(message:="canRX.xtd= \1", v1:=INT(canRX.xtd));
      DebugFmt(message:="canRX.ID= \4", v4:=canRX.ID);
      DebugFmt(message:="canRX.DataSize= \1", v1:=INT(canRX.DataSize));
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.9.6 canReceiveMessage (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: MX2 pro, DX4 pro, CX1 pro-c/warp-c, MX2 turbo/encore/warp, NX-200, NX-400, LX2, LX5
Firmware version: 1.00 / 1.00.00

This receives a message from the CAN network. Before any reception is realised, a receive filter must have been created with canFilterCreate, canFilterCreateOnce, canFilterCreateX, or canFMSFilterCreate. If the canLoopBackMode is enabled, all transmitted messages from the RTCU will be received - but only if a receive filter is set up. The function block must also be initialized and manually updated in order to receive any messages. When a message has been received, the ready flag will be set, and the buffer defined during initialization contains the data while the function block output variables contain the message identifier (ID) information. The data is valid until the function block is updated again. The RTCU will buffer all valid incoming messages in an internal buffer with room for 100 messages - further messages will be lost. This will not, however, affect the logger which has its own buffer.

The function block can only receive CAN messages from the first CAN bus.

Input:

data : PTR
Address of the buffer to receive the data in.
Note: the buffer must be 8 bytes long.

Output:

xtd : BOOL
TRUE: Message uses the extended identifier (29 bits).

FALSE: Message uses the standard identifier (11 bits).

id : DINT
The identifier of the message.

datasize : SINT
Number of bytes received.

ready : BOOL
True if a message has been received.

Declaration:

```
FUNCTION_BLOCK canReceiveMessage;
VAR_INPUT
   data     : PTR;
END_VAR;
VAR_OUTPUT
   xtd      : BOOL;
   ID       : DINT;
   datasize : SINT;
   ready    : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   canRX : canReceiveMessage;
   buf   : ARRAY [1..8] OF SINT;
END_VAR;

PROGRAM CANExample;
VAR
   FilterID : SINT;
   rc       : INT;
END_VAR;

// Open can
canOpen(baud := 250, monitor := FALSE);
canRX(data := ADDR(buf));

canLoopBackMode(enable := ON);

// startID: Priority=3 Reserved=1 Data page=0 PGN=00FDD6
FilterID := canFilterCreate(xtd:=TRUE,startID:=16#0EFDD600,length:=6);

rc := canSendMessage(xtd:=TRUE,ID:=16#0EFDD600,data:=ADDR(buf),datasize:=8);
IF rc = 0 THEN
   DebugMsg(message:="CAN message sent");
ELSE
   DebugFmt(message:="CAN message failed (\1)",v1:=rc);
END_IF;

BEGIN
   canRX();
   ...
   IF canRX.ready THEN
      DebugMsg(message:="Message received!");
      DebugFmt(message:="canRX.xtd= \1", v1:=INT(canRX.xtd));
```

```
      DebugFmt(message:="canRX.ID= \4", v4:=canRX.ID);
      DebugFmt(message:="canRX.DataSize= \1", v1:=INT(canRX.DataSize));
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.9.7 canReceiveX (Function)

Architecture: NX32L
Device support: NX-200, NX-400, LX2, LX5
Firmware version: 1.30.02

This receives a message from a CAN network. Before any reception is realized, a receive filter must have been created with canFilterCreate, canFilterCreateOnce, canFilterCreateX, or canFMSFilterCreate. If the canLoopBackMode is enabled, all transmitted messages from the RTCU will be received - but only if a receive filter is set up.

Input:

port : SINT (1/2) ***(default 1)***
The port of the CAN bus.

timeout : INT (-1..32767) ***(default 0)***
The number of milliseconds to wait for a CAN message
0 - Return immediately if no message
-1 - Wait for ever.

data : PTR
Address of the buffer to receive the data in.
Note: the buffer must be 8 bytes long.

Output:

xtd : BOOL
TRUE: Message uses the extended identifier (29 bits).
FALSE: Message uses the standard identifier (11 bits).

id : DINT
The identifier of the message.

datasize : INT
Number of bytes received.

Returns: INT

1 - Successful.
0 - The canReceiveX function is not supported.
-1 - The CAN bus is not present.
-2 - The CAN bus is not open.
-3 - Illegal parameter.
-4 - Timeout.

Declaration:

```
FUNCTION canReceiveX : INT;
VAR_INPUT
   port     : SINT := 1;
```

```
   timeout  : INT := 0;
   data     : PTR;
   xtd      : ACCESS BOOL;
   ID       : ACCESS DINT;
   datasize : ACCESS INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   buf    : ARRAY [1..8] OF SINT;
END_VAR;

PROGRAM CANExample;
VAR
   FilterID : SINT;
   rc       : INT;
   msg_xtd  : BOOL;
   msg_id   : DINT;
   msg_size : INT;
END_VAR;

// Open can
canOpen(baud := 250, monitor := FALSE);

canLoopBackMode(enable := ON);

FilterID := canFilterCreate(xtd:=TRUE,startID:=16#0EFDD600,length:=6);

rc := canSendMessage(xtd:=TRUE,ID:=16#0EFDD600,data:=ADDR(buf),datasize:=8);
IF rc = 0 THEN
   DebugMsg(message:="CAN message sent");
ELSE
   DebugFmt(message:="CAN message failed (\1)",v1:=rc);
END_IF;

BEGIN
   ...
   rc := canReceiveX(port := 1, timeout := -1, data := ADDR(buf), xtd :=
msg_xtd, id := msg_id, datasize := msg_size);
   CASE rc OF
   1:  DebugMsg(message:="CAN message received.");
       DebugFmt(message:="canRX - xtdd= \1", v1:=INT(msg_xtd));
       DebugFmt(message:="canRX - ID= \4", v4:=msg_id);
       DebugFmt(message:="canRX - DataSize= \1", v1:=msg_size);
   0:  DebugMsg(message:="The canReceiveX function is not supported.");
   -1: DebugMsg(message:="The CAN bus is not present.");
   -2: DebugMsg(message:="The CAN bus is not open.");
   -3: DebugMsg(message:="Illegal parameter.");
   -4: DebugMsg(message:="Timeout on CAN receive.");
   ELSE
       DebugFmt(message:="canReceiveX (rc=\1)", v1:=rc);
   END_CASE;
   ...
END;

END_PROGRAM;
```

4.2.9.8 canFilterCreate (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2 pro, DX4 pro, CX1 pro-c/warp-c, MX2 turbo/encore/warp, NX-200, NX-400, LX2, LX5
Firmware version: 1.00 / 1.00.00

This function is used to create receive filters, which are required to be able to receive any messages from the network.
A filter can either match a single message ID or a range of message IDs.
The filter ID returned when creating the filter must also be used to destroy the filter with canFilterDestroy when it is no longer needed or if it needs to be changed.
The maximum number of filters is 30 for most devices, but the RTCU NX-200 and the RTCU NX-400 have support for up to 64 filters per CAN port for a maximum of 127 filters.

Each receive filter is defined by a start-ID and a length. If the length is 1, it will only match the start-ID, if the length is e.g. 3, it will match start-ID, start-ID+1, and start-ID + 2.
Internally, the start-ID and the length are used to create a bit-mask that will match all the needed IDs. This mask is then used to create a hardware filter that handles the high-performance filtering.
Depending on the values chosen for start-ID and length, the hardware filter might cover additional message IDs so a second filtering is done using the start-ID and the length.
Because of filtering, care must be taken when choosing the range for the filters, it is especially necessary to consider the following two limitations:

Too many unwanted messages

A filter might cover too many unwanted IDs, which causes additional load on the device which can cause reduced performance for the wanted message.
The number of IDs covered by a filter is determined by the position of the highest-order bit that is different between the first and last ID in a filter. If only the LSB is different it covers 2 IDs, if the next bit is also different, it covers 4 IDs, and so on.

An example of this is a start-ID of `16#07FE` and a length of 3.
Looking at the binary values shows the problem:

Start-ID	`0000_0111_1111_1110`	`16#07FE`
End-ID	`0000_1000_0000_0000`	`16#0800`
Mask	`0000_1111_1111_1111`	`16#0FFF`
Actual filter:		
Start-ID	`0000_0000_0000_0000`	`16#0000`
End-ID	`0000_1111_1111_1111`	`16#0FFF`

The mask will match all IDs in the range from `16#0000` to `16#0FFF`, a length of 4096 instead of the needed 3.

A better solution is to split the range into multiple filters, in this case two filters as shown below:

Filter 1: Start-ID `16#07FE`, length 2:

Start-ID	`0000_0111_1111_1110`	`16#07FE`
End-ID	`0000_0111_1111_1111`	`16#07FF`
Mask	`0000_0000_0000_0001`	`16#0001`
Actual filter:		
Start-ID	`0000_0111_1111_1110`	`16#07FE`
End-ID	`0000_0111_1111_1111`	`16#07FF`

Filter 2: Start-ID `16#0800`, length 1:

Start-ID	`0000_1000_0000_0000`	`16#0800`
End-ID	`0000_1000_0000_0000`	`16#0800`
Mask	`0000_0000_0000_0000`	`16#0000`
Actual filter:		

Start-ID	0000_1000_0000_0000	16#0800
End-ID	0000_1000_0000_0000	16#0800

Overlapping filters

If a hardware filter overlaps with other hardware filters, it might prevent the other filters from receiving the messages.
Different devices handle overlapping filters differently, so for a portable solution, overlapping filters should be avoided.

As an example, when using the following two filters, the messages for the second filter might be lost, because the mask for the first filter overlaps with the second filter while the wanted ranges do not:

Filter 1: Start-ID 16#0050, length 20:

Start-ID	0000_0000_0101_0000	16#0050
End-ID	0000_0000_0110_0011	16#0063
Mask	0000_0000_0011_1111	16#003F
Actual filter:		
Start-ID	0000_0000_0100_0000	16#0040
End-ID	0000_0000_0111_1111	16#007F

Filter 2: Start-ID 16#0040, length 3:

Start-ID	0000_0000_0100_0000	16#0040
End-ID	0000_0000_0100_0010	16#0042
Mask	0000_0000_0000_0011	16#0003
Actual filter:		
Start-ID	0000_0000_0100_0000	16#0040
End-ID	0000_0000_0100_0011	16#0043

A solution is to break the large filter up into two smaller filters that are better aligned:

Filter 1: Start-ID 16#0050, length 16:

Start-ID	0000_0000_0101_0000	16#0050
End-ID	0000_0000_0101_1111	16#005F
Mask	0000_0000_0000_1111	16#000F
Actual filter:		
Start-ID	0000_0000_0101_0000	16#0050
End-ID	0000_0000_0101_1111	16#005F

Filter 2: Start-ID 16#0060, length 4:

Start-ID	0000_0000_0110_0000	16#0060
End-ID	0000_0000_0110_0011	16#0063
Mask	0000_0000_0000_0011	16#0003
Actual filter:		
Start-ID	0000_0000_0110_0000	16#0060
End-ID	0000_0000_0110_0011	16#0063

Filter 3: Start-ID 16#0040, length 3:

Start-ID	0000_0000_0100_0000	16#0040
End-ID	0000_0000_0100_0010	16#0042
Mask	0000_0000_0000_0011	16#0003
Actual filter:		
Start-ID	0000_0000_0100_0000	16#0040
End-ID	0000_0000_0100_0011	16#0043

Input:

port : SINT (1/2) *(default 1)*
The port of the CAN bus.

xtd : BOOL
Filter for standard or extended identifiers. Set to TRUE for extended identifiers.

startID : DINT (16#0...16#1FFF_FFFF)
The first identifier that is accepted.

length : DINT (16#1...16#2000_0000)
The length of the range of identifiers that are accepted. For a single identifier, the length should only be set to 1.

Returns: SINT

>0	- ID of the new filter.
-1	- Illegal start ID.
-2	- Illegal length.
-3	- No free filters.
-8	- The CAN bus is not open.

Declaration:

```
FUNCTION canFilterCreate : SINT;
VAR_INPUT
   port    : SINT := 1;
   xtd     : BOOL;
   startid : DINT;
   length  : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   canRX : canReceiveMessage;
   buf   : ARRAY [1..8] OF SINT;
END_VAR;

PROGRAM CANExample;
VAR
   FilterID : SINT;
   rc       : INT;
END_VAR;

// Open can
canOpen(baud := 250, monitor := FALSE);
canRX(data := ADDR(buf));

canLoopBackMode(enable := ON);

// startID: Priority=3 Reserved=1 Data page=0 PGN=00FDD6
FilterID := canFilterCreate(xtd:=TRUE,startID:=16#0EFDD600,length:=6);

rc := canSendMessage(xtd:=TRUE,ID:=16#0EFDD600,data:=ADDR(buf),datasize:=8);
IF rc = 0 THEN
   DebugMsg(message:="CAN message sent");
ELSE
   DebugFmt(message:="CAN message failed (\1)",v1:=rc);
END_IF;
```

```
BEGIN
   canRX();
   ...
   IF canRX.ready THEN
      DebugMsg(message:="Message received!");
      DebugFmt(message:="canRX.xtd= \1", v1:=INT(canRX.xtd));
      DebugFmt(message:="canRX.ID= \4", v4:=canRX.ID);
      DebugFmt(message:="canRX.DataSize= \1", v1:=INT(canRX.DataSize));
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.9.9 canFilterCreateOnce (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2 pro, DX4 pro, CX1 pro-c/warp-c, MX2 turbo/encore/warp, NX-200, NX-400, LX2, LX5
Firmware version: 1.00 / 1.00.00

This will create a receive filter similar to canFilterCreate but with automatic destruction after the first valid message has been received (one shot).
When a filter has been successfully created, the function returns an ID for the filter.
This ID can be used to check the status of the filter with canFilterStatus. It is recommended to check if a filter is still active (no valid message received) before creating it again.
When using multiple one shot filters, it is mandatory to destroy the filter with canFilterDestroy before creating the filter again. If one is only using a single one shot filter, it is optional to destroy it.
Note: the filter created with this function will add the received message both to the internal buffer and to the Logger.

Please see the canFilterCreate function for more information on working with receive filters.

Input:

port : SINT (1/2) *(default 1)*
The port of the CAN bus.

xtd : BOOL
Filter for standard or extended identifiers. Set to TRUE for extended identifiers.

startID : DINT (16#0...16#1FFF_FFFF)
The first identifier that is accepted.

length : DINT (16#1...16#2000_0000)
The length of the range of identifiers that are accepted. For a single identifier, the length should only be set to 1.

Returns: SINT

>0	- ID of the new filter.
-1	- Illegal start ID.
-2	- Illegal length.
-3	- No free filters.
-8	- The CAN bus is not open.

Declaration:

```
FUNCTION canFilterCreateOnce : SINT;
VAR_INPUT
```

```
   port    : SINT := 1;
   xtd     : BOOL;
   startid : DINT;
   length  : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   canRX : canReceiveMessage;
   buf   : ARRAY [1..8] OF SINT;
END_VAR;

PROGRAM CANExample;
VAR
   Filter_ID  : SINT;
   FilterOnce : ARRAY [1..2] OF SINT;
   timer      : TON;
END_VAR;

// Open can
canOpen(baud := 250, monitor := FALSE);
canRX(data := ADDR(buf));

// Open filters
timer.pt  := 1000;
Filter_ID := canFilterCreate(xtd := TRUE, startID := 16#0EFDD600, length :=
6);

BEGIN
   ...
   canRX();
   IF canRX.ready THEN
      DebugMsg(message:="Message received!");
      DebugFmt(message:="canRX.xtd= \1", v1:=INT(canRX.xtd));
      DebugFmt(message:="canRX.ID= \4", v4:=canRX.ID);
      DebugFmt(message:="canRX.DataSize= \1", v1:=INT(canRX.DataSize));
   END_IF;
   ...
   timer(trig := TRUE);
   IF timer.q THEN
      IF canFilterStatus(filterid := FilterOnce[1]) <> 1 THEN
         canFilterDestroy(filterid := FilterOnce[1]);
         FilterOnce[1] := canFilterCreateOnce(xtd := TRUE, startID :=
16#0E00FDD3, length := 1);
      END_IF;
      IF canFilterStatus(filterid := FilterOnce[2]) <> 1 THEN
         canFilterDestroy(filterid := FilterOnce[2]);
         FilterOnce[2] := canFilterCreateOnce(xtd := TRUE, startID :=
16#0E00FDD4, length := 1);
      END_IF;
      timer(trig := FALSE);
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.9.10 canFilterCreateX (Function)

```
Architecture:       X32 / NX32 / NX32L
Device support:     MX2 pro, DX4 pro, CX1 pro-c/warp-c, MX2
                    turbo/encore/warp, NX-200, NX-400, LX2, LX5
Firmware version:   2.50 (2.62 : changed filtering) / 1.00.00
```

This will create a receive filter similar to canFilterCreate, except that it offers some advanced filtering options on top of the simple ID filtering.
When a filter has been successfully created, the function returns an ID for the filter.
This ID can be used to check the status of the filter with canFilterStatus.

Please see the canFilterCreate function for more information on working with receive filters.

Grouping: *(only supported on NX32/NX32L, requires firmware V5.04 / R1.31.00 or newer)*
A group is a sequence of CAN messages with the same identifier, but with differing data contents.
By defining a group, the advanced filter options below can be applied on the group as a whole, instead of each individual message.
Because a group contains multiple messages with the same identifier, the 'Contents changed filtering' cannot be used with a group.
The start of the group is determined by applying a bit mask to the data contents of a received message, and then comparing it with a data sequence.
Once a message with a matching data content is received, it and the next messages will be accepted (the number of messages are configured in the group parameter).
The start data sequence is 8 bytes, which is configured using the change_hi_mask parameter, and the start bit mask is 8 bytes which is configured using the change_lo_mask.

For example, if a group of messages have the following content:
Message: 01 00 A0 2A 1B 04 78 05
Message: 02 58 30 1B 04 00 00 00
Message: 03 00 AA 2A 1A 04 78 05
Message: 04 4C 30 1A 04 00 00 00

The start data sequence would be:
Start: 01 00 00 00 00 00 00 00

And the bit mask would be:
Mask: FF 00 00 00 00 00 00 00

Time_interval filtering: *(only supported on NX32/NX32L, requires firmware V5.04 / R1.31.00 or newer)*
When you have a message that needs to be read at a regular interval, whether the contents change or not, then this filter option is the one to use.
This option will set a time interval in which incoming messages are ignored.
For example, if time_interval is set to 5000, then for the first 5 seconds all incoming messages are ignored, the next incoming message is then accepted, and for the next 5 seconds all incoming messages are ignored. This repeats until the filter is destroyed.
Both the timeout and time_interval options cannot be enabled simultaneously.
The time_interval is the first advanced filtering done.

Downsampling:
When you have a message that is sent far too often, and where the contents are changing too regularly, then this filter option is the one to use.
This option will count the incoming messages, and the messages will be ignored until the counter reaches the selected number.
For example, if downsample is set to 3, then the 1st and 2nd incoming messages are ignored and the 3rd is received, the 4th and 5th are ignored and the 6th is received - this repeats until the filter is destroyed.
Downsampling is done after the time_interval filtering, but before any other advanced filtering.

Contents changed filtering:
This option will compare an incoming message with the last received message, and if they are identical, then the incoming message is ignored.
The last received message is stored for the whole filter - not the individual IDs. As such, the option will not work as expected in a filter with a length higher than 1.
The two bit masks, change_hi_mask and change_lo_mask, allow for a fine grained-control of the comparison between the message data of the incoming message and the last message.
The change_hi_mask controls which parts of the message data, down to the individual bits, that are ignored in a comparison where they change from low state to high state.
Like the change_hi_mask, the change_lo_mask controls which parts of the message data are ignored in the comparison - except for changes from high state to low state.
Each mask is an array of 8 bytes, where each byte is used to determine which bits of the corresponding message data byte will be ignored during the comparison (byte 1 in mask is used to filter byte 1 in data).

Timeout filtering:
If all the incoming messages have been filtered away in the timeout period, then the next incoming message will bypass the advanced filtering and be received.

Message limit filtering:
This option will automatically disable the filter after a number of messages are received. This is a similar behaviour to the canFilterCreateOnce function - except that it is not limited to only one message.
If a limit is selected for the filter, it is recommended to check if a filter is still active (the limit of valid messages received is not reached) before creating it again.
When using more than one filter with limits, it is mandatory to destroy the filter with canFilterDestroy before creating the filter again. If only one limited filter is used, destroying the filter is optional.

Destination filtering:
This filter option allows for logging CAN messages without having to handle them in the application, and to handle CAN messages in the application without logging them.

Input:

port : SINT (1/2) *(default 1)*
The port of the CAN bus.

xtd : BOOL
Filter for standard or extended identifiers. Set to TRUE for extended identifiers.

startID : DINT (16#0...16#1FFF_FFFF)
The first identifier that is accepted.

length : DINT (16#1...16#2000_0000)
The length of the range of identifiers that are accepted. For a single identifier, the length should only be set to 1.

downsample : SINT (1..127, *Default 1*)
Only accepts each x incoming message - thus allowing logging for a longer period.

changed : BOOL

TRUE	- Only accepts an incoming message if contents are different from the last received message.
FALSE	- All incoming messages are accepted.

change_hi_mask : PTR
Address of a bit mask that is used to determine which parts of the message data is used when checked for low state to high state differences.
If change_hi_mask is not used, all parts of the message data is used. This parameter is ignored if changed is FALSE.
Note: the bit mask must be exactly 8 bytes long.

change_lo_mask : PTR
Address of a bit mask that is used to determine which parts of the message data is used when checked for high state to low state differences.
If change_lo_mask is not used, all parts of the message data is used. This parameter is ignored if changed is FALSE.
Note: the bit mask must be exactly 8 bytes long.

destination : SINT (1...3, *default 3*)
1 - The received messages are forwarded to the application.
2 - The received messages are forwarded to the logger.
3 - The received messages are forwarded to both the application and the logger.

limit : SINT (0...127)
The number of received messages before the filter is disabled. There is no limit if set to 0 (zero).

group : SINT (0,2...127, *default 0*) *(only supported on NX32/NX32L, requires firmware V5.04 / R1.31.00 or newer)*
The number of messages in sequence that are part of the group.
The group is disabled if set to 0 (zero).

timeout : SINT (0...127)
The number of seconds without receiving a message before the next incoming message is automatically accepted.
The timeout is disabled if set to 0 (zero).

time_interval : INT (0,100...32767, *default 0*) *(only supported on NX32/NX32L, requires firmware V5.04 / R1.31.00 or newer)*
The number of milliseconds after receiving a message, where incoming messages are ignored.
The time_interval is disabled if set to 0 (zero).

Returns: SINT

>0 - ID of the new filter.
-1 - Illegal start ID.
-2 - Illegal length.
-3 - No free filters.
-4 - Illegal downsample.
-5 - Illegal limit.
-6 - Illegal timeout.
-7 - Illegal destination.
-8 - The CAN bus is not open.
-10 - Illegal group.
-11 - Both group and changed is enabled.
-12 - Illegal time_interval.
-13 - Both time_interval and timeout is enabled.

Declaration:

```
FUNCTION canFilterCreateX : SINT;
VAR_INPUT
   port           : SINT := 1;
   xtd            : BOOL;
   startid        : DINT;
   length         : DINT;
```

```
   downsample     : SINT := 1;
   changed        : BOOL;
   destination    : SINT := 3;
   limit          : SINT;
   timeout        : SINT;
   group          : SINT := 0;
   time_interval  : INT  := 0;
   change_hi_mask : PTR;
   change_lo_mask : PTR;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   canRX : canReceiveMessage;
   buf   : ARRAY[1..8] OF SINT;
END_VAR;

PROGRAM CANExample;
VAR
   FilterID  : ARRAY[1..4] OF SINT;
   timer     : TON;
   mask_hi   : ARRAY [1..8] OF SINT;
   mask_lo   : ARRAY [1..8] OF SINT;
END_VAR;

// Open can
canOpen(baud := 250);
canRX(data := ADDR(buf));

// Open filters
timer.pt  := 1000;
FilterID[1] := canFilterCreateX(xtd := TRUE, startID := 16#0EFDD600, length :=
 6);

// Define High mask
mask_hi[1] := 16#C0;
mask_hi[2] := 16#00;
mask_hi[3] := 16#00;
mask_hi[4] := 16#03;
mask_hi[5] := 16#00;
mask_hi[6] := 16#00;
mask_hi[7] := 16#00;
mask_hi[8] := 16#00;

// Define Low mask
mask_lo[1] := 16#C0;
mask_lo[2] := 16#00;
mask_lo[3] := 16#00;
mask_lo[4] := 16#00;
mask_lo[5] := 16#00;
mask_lo[6] := 16#00;
mask_lo[7] := 16#00;
mask_lo[8] := 16#00;

FilterID[4] := canFilterCreateX(xtd := TRUE, startID := 16#0EFDD700, length :=
 1,
                                changed := TRUE,
                                change_hi_mask := ADDR(mask_hi),
                                change_lo_mask := ADDR(mask_lo));
```

```
BEGIN
   ...
   canRX();
   IF canRX.ready THEN
      DebugMsg(message := "Message received!");
      DebugFmt(message := "canRX.xtd= \1", v1 := INT(canRX.xtd));
      DebugFmt(message := "canRX.ID= \4", v4 := canRX.ID);
      DebugFmt(message := "canRX.DataSize= \1", v1 := INT(canRX.DataSize));
   END_IF;
   ...
   timer(trig := TRUE);
   IF timer.q THEN
      IF canFilterStatus(filterid := FilterID[2]) <> 1 THEN
         canFilterDestroy(filterid := FilterID[2]);
         FilterID[2] := canFilterCreateX(xtd := TRUE, startID := 16#0E00FDD3,
length := 1, limit := 10);
      END_IF;
      IF canFilterStatus(filterid := FilterID[3]) <> 1 THEN
         canFilterDestroy(filterid := FilterID[3]);
         FilterID[3] := canFilterCreateX(xtd := TRUE, startID := 16#0E00FDD4,
length := 1, limit := 10);
      END_IF;
      timer(trig := FALSE);
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.9.11 canFilterDestroy (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2 pro, DX4 pro, CX1 pro-c/warp-c, MX2 turbo/encore/warp, NX-200, NX-400, LX2, LX5
Firmware version: 1.00 / 1.00.00

---

canFilterDestroy will destroy a receive filter that is no longer needed. The filter ID retrieved from canFilterCreate, canFilterCreateOnce. or canFilterCreateX must be used to identify the filter.

Input:

filterid : SINT
ID for the filter to destroy.

Returns: SINT

0 - Success.
1 - Illegal filter ID.

Declaration:

```
FUNCTION canFilterDestroy : SINT;
VAR_INPUT
   filterid : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
```

```
   canRX : canReceiveMessage;
   buf   : ARRAY [1..8] OF SINT;
END_VAR;

PROGRAM CANExample;
VAR
   Filter_ID : SINT;
   cmd       : SINT;
END_VAR;

// Open can
canOpen(baud := 250);
canRX(data := ADDR(buf));

BEGIN
   ...
   IF cmd = 16#02 THEN
      // startID: Priority=3 Reserved=1 Data page=0 PGN=00FDD6
      Filter_ID := canFilterCreate(xtd:=TRUE,startID:=16#0EFDD600,length:=6);
   ELSIF cmd = 16#03 THEN
      canFilterDestroy(filterid:=Filter_ID);
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.9.12 canFilterStatus (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2 pro, DX4 pro, CX1 pro-c/warp-c, MX2 turbo/encore/warp, NX-200, NX-400, LX2, LX5
Firmware version: 1.40 / 1.00.00

This will return the status of a receive filter. The filter ID retrieved from canFilterCreate, canFilterCreateOnce, or canFilterCreateX must be used to identify the filter.

Input:

filterid : SINT
ID for the filter.

Returns: INT

-1 - Illegal filter ID.
0 - Filter not created.
1 - Filter active.
2 - Filter inactive (the number of received message has reached the limit for the filter).

Declaration:

```
FUNCTION canFilterStatus : INT;
VAR_INPUT
   filterid : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
```

```
   canRX : canReceiveMessage;
   buf   : ARRAY [1..8] OF SINT;
END_VAR;

PROGRAM CANExample;
VAR
   Filter_ID  : SINT;
   FilterOnce : SINT;
   timer      : TON;
END_VAR;

// Open can
canOpen(baud := 250);
canRX(data := ADDR(buf));

// Open filters
timer.pt  := 1000;
Filter_ID := canFilterCreate(xtd := TRUE, startID := 16#0EFDD600, length :=
6);

BEGIN
   ...
   canRX();
   IF canRX.ready THEN
      DebugMsg(message:="Message received!");
      DebugFmt(message:="canRX.xtd= \1", v1:=INT(canRX.xtd));
      DebugFmt(message:="canRX.ID= \4", v4:=canRX.ID);
      DebugFmt(message:="canRX.DataSize= \1", v1:=INT(canRX.DataSize));
   END_IF;
   ...
   timer(trig := TRUE);
   IF timer.q THEN
      IF canFilterStatus(filterid := FilterOnce) <> 1 THEN
         canFilterDestroy(filterid := FilterOnce);
         FilterOnce := canFilterCreateOnce(xtd := TRUE, startID :=
16#0E00FDD3, length := 1);
      END_IF;
      timer(trig := FALSE);
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.9.13 canFlush (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2 pro, DX4 pro, CX1 pro-c/warp-c, MX2 turbo/encore/warp, NX-200, NX-400, LX2, LX5
Firmware version: 1.40 / 1.00.00

This will remove all messages received from the CAN bus and currently stored in the internal receive buffer (see canReceiveMessage).

Input:

port : SINT (1/2) ***(default 1)***
The port of the CAN bus.

Returns: **INT**

0 - The CAN bus is not open.

1 - Success.

Declaration:

```
FUNCTION canFlush : INT;
VAR_INPUT
   port : SINT := 1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM CANExample;

// Open can
canOpen(baud := 250);
...

BEGIN
   ...
   ELSIF cmd = cmd_flush THEN
      // Clear received CAN messages
      canFlush();
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.9.14 canBufferLevel (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2 pro, DX4 pro, CX1 pro-c/warp-c, MX2 turbo/encore/warp, NX-200, NX-400, LX2, LX5
Firmware version: 2.20 / 1.00.00

This function will return the amount of messages currently stored in the internal receive buffer. The returned value is on a scale between 0 and 1000 (permille). The exact size of the receive buffer is specified in the help section for canReceiveMessage.
Note: use canLoggerLevel to get the level of the logger that is used.

Input:

port : SINT (1/2) *(default 1)*
The port of the CAN bus.

Returns: INT

Amount of messages that are used in the internal receive buffer - 0..1000 permille.

Declaration:

```
FUNCTION canBufferLevel : INT;
VAR_INPUT
   port : SINT := 1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM CANExample;

// Open can
canOpen(baud := 250);
...

BEGIN
   ...
   // List the number of received CAN messages
   DebugFmt(message := "CAN buffer: \1 promille", v1 := canBufferLevel());
   ...
END;

END_PROGRAM;
```

4.2.9.15 canLoggerSetup (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2 pro, DX4 pro, CX1 pro-c/warp-c, MX2 turbo/encore/warp, NX-200, NX-400, LX2, LX5
Firmware version: 2.50 / 1.00.00

This function will initialize a buffer where received CAN messages are logged.
The logging to the buffer is started with the canLoggerStart function.

The logger can operate in two modes - with and without time information.
Only filtered messages are logged.

Input:

port : SINT (1/2) *(default 1)*
The port of the logger.

buffer : PTR
Address of buffer that is used to store the logged values.

Size : DINT (13..2147483647)
Size of buffer area in bytes.

Mode : SINT (*default 1*) *(mode 2 is only supported on NX32/NX32L, requires firmware V4.69 / R1.11.00 or newer)*

1 The logger will store CAN messages. (ID, size and 8 bytes data)
2 The logger will store CAN messages with time-stamp (time-stamp, ID, size and 8 bytes data)

The logger time-stamp, available in mode 2, is the number of milliseconds since the previous record in the log.
The time-stamp in mode 2 is stored as an INT type (16 bit), with the following semantics:

0 = first record.
1 = time since previous record <=1 ms.
1..32767 = number of milliseconds since previous record.
-1 = number of milliseconds since previous record is > 32767 milliseconds.

Returns: **INT**

0 - Success.
1 - Logger is running.
2 - Invalid argument.
3 - Invalid logger index.

Declaration:

```
FUNCTION canLoggerSetup : INT;
VAR_INPUT
   port   : SINT := 1;
   buffer : PTR;
   size   : DINT;
   mode   : SINT := 1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   LogBuf : ARRAY [1..1300] OF SINT;
END_VAR;

PROGRAM CANExample;

// Open can
canOpen(baud := 250);
canLoggerSetup(buffer := ADDR(LogBuf), size := SIZEOF(LogBuf));
...

BEGIN
   ...
END;
END_PROGRAM;
```

4.2.9.16 canLoggerStart (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2 pro, DX4 pro, CX1 pro-c/warp-c, MX2 turbo/encore/warp, NX-200, NX-400, LX2, LX5
Firmware version: 2.50 / 1.00.00

This function will start (or resume) the logging.
The logger must first have been configured with the canLoggerSetup function.

Restarting the logging will not remove any logged CAN messages if present.

Note that only filtered messages are logged.

Input:

port : SINT (1/2) ***(default 1)***
The port of the logger.

Returns: **INT**

0 - Success.
1 - Logger is not configured.
2 - Logger is already started.

3 - Illegal logger index.

Declaration:

```
FUNCTION canLoggerStart : INT;
VAR_INPUT
   port : SINT := 1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM CANExample;
VAR_INPUT
   LogBuf : ARRAY [1..1300] OF SINT;
END_VAR;

// Open can
canOpen(baud := 250);
canLoggerSetup(buffer := ADDR(LogBuf), size := SIZEOF(LogBuf));
...

BEGIN
   ...
   canLoggerStart();
   ...
END;
END_PROGRAM;
```

4.2.9.17 canLoggerStop (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2 pro, DX4 pro, CX1 pro-c/warp-c, MX2 turbo/encore/warp, NX-200, NX-400, LX2, LX5
Firmware version: 2.50 / 1.00.00

This function will stop the logging.
The logger must first have been started with the canLoggerStart function.

Input:

port : SINT (1/2) *(default 1)*
The port of the logger.

Returns: INT

0 - Success.
1 - Logger is not started.
3 - Illegal logger index.

Declaration:

```
FUNCTION canLoggerStop : INT;
VAR_INPUT
   port : SINT := 1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM CANExample;
VAR_INPUT
   LogBuf : ARRAY [1..1300] OF SINT;
END_VAR;

// Open can
canOpen(baud := 250);
canLoggerSetup(buffer := ADDR(LogBuf), size := SIZEOF(LogBuf));
...

BEGIN
   ...
   canLoggerStart();
   ...
   canLoggerStop();
   ...
END;
END_PROGRAM;
```

4.2.9.18 canLoggerLevel (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2 pro, DX4 pro, CX1 pro-c/warp-c, MX2 turbo/encore/warp, NX-200, NX-400, LX2, LX5
Firmware version: 2.50 / 1.00.00

This function will return the amount of space used in the logger. The returned value is on a scale between 0 and 1000 (permille).
The exact size of the logger is specified during configuration (see canLoggerSetup).
Note: use canBufferLevel to check the level of use for the internal buffer.

Input:

port : SINT (1/2) ***(default 1)***
The port of the logger.

Returns: INT

Amount of space used in the logger - 0..1000 permille.

Declaration:

```
FUNCTION canLoggerLevel : INT;
VAR_INPUT
   port : SINT := 1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM CANExample;
VAR
   LogBuf : ARRAY [1..1300] OF SINT;
   level  : INT;
END_VAR;
```

```
// Open can
canOpen(baud := 250);
canLoggerSetup(buffer := ADDR(LogBuf), size := SIZEOF(LogBuf));
...

BEGIN
   ...
   canLoggerStart();
   ...
   // List the number of received CAN messages
   level := canLoggerLevel();
   IF level >= 900 THEN
      // Logger level high mark, handle
      ...
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.9.19 canLoggerRead (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: MX2 pro, DX4 pro, CX1 pro-c/warp-c, MX2 turbo/encore/warp, NX-200, NX-400, LX2, LX5
Firmware version: 2.50 / 1.00.00

This function will read the oldest record stored in the logger when operating in mode 1 (see canLoggerSetup).

The function block must be initialized and manually updated in order to read any messages from the logger.
When a message is read from the logger, the ready flag will be set, the output variables will contain the message identifier (ID) information, and the buffer defined during initialization will contain the data.
The data is valid until the function block is updated again.
Note: this function will not affect the internal buffer.

The function block can only read CAN messages from the first logger.

Input:

data : PTR
Address of the buffer to receive the data in.
Note: the buffer must be 8 bytes long.

Output:

xtd : BOOL
TRUE: Message uses the extended identifier (29 bits).
FALSE: Message uses the standard identifier (11 bits).

id : DINT
The identifier of the message.

datasize : SINT
Number of bytes received.

ready : BOOL
True if a message has been received.

Declaration:

```
FUNCTION_BLOCK canLoggerRead;
VAR_INPUT
   data     : PTR;
END_VAR;
VAR_OUTPUT
   xtd      : BOOL;
   ID       : DINT;
   datasize : SINT;
   ready    : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   LogBuf : ARRAY [1..1300] OF SINT;
   buf    : ARRAY [1..8] OF SINT;
   reader : canLoggerRead;
END_VAR;

PROGRAM Example;
VAR
   FilterID : SINT;
END_VAR;

// Initialize CAN
canOpen(baud := 250);
FilterID := canFilterCreate(xtd := TRUE, startID := 16#0EFDD600, length := 6);
canLoggerSetup(buffer := ADDR(LogBuf), size := SIZEOF(LogBuf));
reader(data := ADDR(buf));
...

BEGIN
   ...
   reader();
   IF reader.ready THEN
      DebugMsg(message:="Message received!");
      DebugFmt(message:="-reader.xtd= \1", v1 := INT(reader.xtd));
      DebugFmt(message:="-reader.ID= \4", v4 := reader.ID);
      DebugFmt(message:="-reader.DataSize= \1", v1 := INT(reader.DataSize));
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.9.20 canLoggerRead2 (Functionblock)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, NX-200, NX-400, LX2, LX5
Firmware version: 4.69 / 1.11.00

This function will read the oldest record stored in the logger when operating in mode 2 (see canLoggerSetup).

The function block must be initialized and manually updated in order to read any messages from the logger.

When a message is read from the logger, the ready flag will be set, the output variables will contain the message identifier (ID) information, and the buffer defined during initialization will contain the data.
The data is valid until the function block is updated again.
Note: this function will not affect the internal buffer.

Input:

port : SINT (1/2) *(default 1)*
The port of the logger.

data : PTR
Address of the buffer to receive the data in.
Note: the buffer must be 8 bytes long.

Output:

xtd : BOOL
TRUE: Message uses the extended identifier (29 bits).
FALSE: Message uses the standard identifier (11 bits).

id : DINT
The identifier of the message.

datasize : SINT
Number of bytes received.

time : INT
The time-stamp. (Look here for information).

ready : BOOL
True if a message has been received.

Declaration:

```
FUNCTION_BLOCK canLoggerRead2;
VAR_INPUT
   port     : SINT := 1;
   data     : PTR;
END_VAR;
VAR_OUTPUT
   xtd      : BOOL;
   ID       : DINT;
   time     : INT;
   datasize : SINT;
   ready    : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   LogBuf : ARRAY [1..1300] OF SINT;
   buf    : ARRAY [1..8] OF SINT;
   reader : canLoggerRead2;
END_VAR;

PROGRAM Example;
```

```
VAR
   FilterID : SINT;
END_VAR;

// Initialize CAN
canOpen(baud := 250);
FilterID := canFilterCreate(xtd := TRUE, startID := 16#0EFDD600, length := 6);
canLoggerSetup(mode:=2,buffer := ADDR(LogBuf), size := SIZEOF(LogBuf));
reader(data := ADDR(buf));
...

BEGIN
   ...
   reader();
   IF reader.ready THEN
      DebugMsg(message:="Message received!");
      DebugFmt(message:="-reader.time= \1", v1 := reader.time);
      DebugFmt(message:="-reader.xtd= \1", v1 := INT(reader.xtd));
      DebugFmt(message:="-reader.ID= \4", v4 := reader.ID);
      DebugFmt(message:="-reader.DataSize= \1", v1 := INT(reader.DataSize));
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.9.21 canLoggerToMemory (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2 pro, DX4 pro, CX1 pro-c/warp-c, MX2 turbo/encore/warp, NX-200, NX-400, LX2, LX5
Firmware version: 2.50 / 1.00.00

This function will move the contents of the logger to memory.

In mode 1, the CAN messages will be stored in memory it the following format:
<xtd/size><ID><data>

In mode 2, the CAN messages will be stored in memory it the following format:
<xtd/size><ID><time><data>

Where:

xtd	4 bits	- Located in the upper nibble of the first byte; zero = standard identifier, other = extended identifier.
size	4 bits	- Located in the lower nibble of the first byte; the number of valid data bytes.
ID	4 Bytes	- The message identifier.
time	2 Bytes	- The timestamp. (Look here for information)
data	8 Bytes	- The message data.

Input:

port : SINT (1/2) *(default 1)*
The port of the logger.

dst : PTR;
Address of destination data area.

size : DINT;
The size of the memory in bytes.

Output:

None.

Returns: **INT**

>0	- Number of bytes written
-1	- Logger is not configured.
-2	- Illegal data found in logger.

Declaration:

```
FUNCTION canLoggerToMemory : INT;
VAR_INPUT
   port   : SINT := 1;
   dst    : PTR;
   size   : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   LogBuf : ARRAY [1..2600] OF SINT;
   read   : ARRAY [1..480] OF SINT;
END_VAR;

PROGRAM example;
VAR
   FilterID : SINT;
   level    : INT;
   rc       : INT;
END_VAR;

// Initialize CAN
canOpen(baud := 250);
FilterID := canFilterCreate(xtd := TRUE, startID := 16#0EFDD600, length := 6);
canLoggerSetup(buffer := ADDR(LogBuf), size := SIZEOF(LogBuf));
canLoggerStart();
...

BEGIN
   ...
   canLoggerStart();
   ...
   // List the number of received CAN messages
   level := canLoggerLevel();
   IF level >= 900 THEN
      // Logger level high mark
      rc := canLoggerToMemory(dst := ADDR(read), size := SIZEOF(read));
      IF rc > 0 THEN
         ...  // Handle data here
      END_IF;
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.9.22 canFMSFilterCreate (Function)

Architecture:	**X32 / NX32 / NX32L**
Device support:	**MX2 pro, DX4 pro, CX1 pro-c/warp-c, MX2 turbo/encore/warp, NX-200, NX-400, LX2, LX5**
Firmware version:	**2.60 / 1.00.00**

This creates a receive filter similar to canFilterCreateX, except that the filter will be for a single FMS/J1939 message.
When a filter has been successfully created, the function returns an ID for the filter.
This ID can be used to check the status of the filter with canFilterStatus.

Please see the canFilterCreate function for more information on working with receive filters and canFilterCreateX for more information on the advanced filter options.

The J1939 standard is a vehicle bus standard used for communication and diagnostics among vehicle components.
The FMS (Fleet Management System interface) is a subset of the J1939 standard and is defined as a standard interface to vehicle data of commercial vehicles.
All FMS/J1939 messages contain 8 bytes of data and a standard header containing an index called PGN (Parameter Group Number) which is embedded in the CAN message's 29-bit identifier (extended identifier).
(The full definition of the header is: <priority:3 bit><reserve:1 bit><PGN:17 bit><source address:8 bit>)

Input:

port : SINT (1/2) *(default 1)*
The port of the CAN bus.

PGN : DINT (16#0...16#1_FFFF)
The PGN number of the FMS/J1939 message.

downsample : SINT (1..127, *Default 1*)
Only accepts each x incoming message - thus allowing logging for a longer period.

changed : BOOL
TRUE - Only accepts an incoming message if contents are different from the last received message.
FALSE - All incoming messages are accepted.

change_hi_mask : PTR
Address of a bit mask that is used to determine which parts of the message data are used when checking for low state to high state differences.
If change_hi_mask is not used, all parts of the message data are used. This parameter is ignored if changed is FALSE.
Note: the bit mask must be 8 bytes long.

change_lo_mask : PTR
Address of a bit mask that is used to determine which parts of the message data are used when checking for high state to low state differences.
If change_lo_mask is not used, all parts of the message data are used. This parameter is ignored if changed is FALSE.
Note: the bit mask must be 8 bytes long.

destination : SINT (1...3, *default 3*)
1 - The received messages are forwarded to the application.
2 - The received messages are forwarded to the logger.
3 - The received messages are forwarded to both the application and the logger.

limit : SINT (0...127)
The number of received messages before the filter is disabled. There is no limit if set to 0 (zero).

timeout : SINT (0...127)
The number of seconds without receiving a message before the next incoming message is automatically accepted.

The timeout is disabled if set to 0 (zero).

group : SINT (0,2...127, *default 0*) *(only supported from on NX32/NX32L firmware V5.04 / 1.31.00)*
The number of messages in sequence that are part of the group.
The group is disabled if set to 0 (zero).

time_interval : INT (0,100...32767, *default 0*) *(only supported from on NX32/NX32L firmware V5.04 / 1.31.00)*
The number of milliseconds after receiving a message, where incoming messages are ignored.
The time_interval is disabled if set to 0 (zero).

Returns: SINT

>0	- ID of the new filter.
-1	- Illegal PGN.
-3	- No free filters.
-4	- Illegal downsample.
-5	- Illegal limit.
-6	- Illegal timeout.
-7	- Illegal destination.
-8	- The CAN bus is not open.
-10	- Illegal group.
-11	- Both the group and change is enabled.
-12	- Illegal time_interval.
-13	- Both the time_interval and timeout is enabled.

Declaration:

```
FUNCTION canFMSFilterCreate : SINT;
VAR_INPUT
   port           : SINT := 1;
   PGN            : DINT;
   downsample     : SINT := 1;
   changed        : BOOL;
   destination    : SINT := 3;
   limit          : SINT;
   timeout        : SINT;
   group          : SINT := 0;
   time_interval  : INT  := 0;
   change_hi_mask : PTR;
   change_lo_mask : PTR;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   can    : canReceiveMessage;
   buf    : ARRAY[1..8] OF SINT;
END_VAR;

FUNCTION_BLOCK ParseCruiseControl;
VAR_INPUT
   data   : PTR;
END_VAR;
VAR_OUTPUT
   speed  : DINT;      // Wheel based speed (SPN84)
   clutch : BOOL;      // Clutch switch (SPN 598), TRUE=pressed,
FALSE=released
   break  : BOOL;      // Break switch (SPN 597), TRUE=pressed, FALSE=released
```

```
   cruise : BOOL;      // Cruise control active (SPN 595), TRUE=switched on,
FALSE=switched off
   val    : SINT;      // Unknown data in byte 7. (SPN 976)
END_VAR;
VAR
   can    : ARRAY [1..8] OF SINT;
   temp   : SINT;
   high   : DINT;
END_VAR;
   // Copy CAN data to local buffer
   memcpy(dst := ADDR(can), src := data, len := 8);

   // Extract Wheel based speed.
   speed := shl32(in := can[2], n := 8) + can[3];

   // Extract Clutch switch
   IF shr8(in := can[4], n := 6) = 1 THEN
      clutch := TRUE;
   ELSE
      clutch := FALSE;
   END_IF;

   // Extract Break switch
   temp := shr8(in := can[4], n := 4) AND 16#03;
   IF temp = 1 THEN
      break := TRUE;
   ELSE
      break := FALSE;
   END_IF;

   // Extract Cruise control active
   IF (can[4] AND 16#03) = 1 THEN
      cruise := TRUE;
   ELSE
      cruise := FALSE;
   END_IF;

   // Extract Unknown data
   val := can[7] AND 16#1F;
END_FUNCTION_BLOCK;

FUNCTION ParseEngineSpeed : DINT;
VAR_INPUT
   data : PTR;
END_VAR;
VAR
   can  : ARRAY [1..8] OF SINT;
   tick : DINT;      // Engine speed (SPN190)
END_VAR;
   // Copy CAN data to local buffer
   memcpy(dst := ADDR(can), src := data, len := 8);

   // Extract engine speed
   tick             := shl32(in := can[4], n := 8) + can[5];
   ParseEngineSpeed := tick * 125/1000; // Convert from ticks to RPM's
END_FUNCTION;

PROGRAM example;
VAR
   PGN  : DINT;
   CCVS : ParseCruiseControl;
END_VAR;

   // Initializing
```

```
   canOpen(baud := 250);
   can.data := ADDR(buf);
   canFMSFilterCreate(PGN := 65265);
   canFMSFilterCreate(PGN := 61444);

BEGIN
   can();
   IF can.ready THEN
      PGN := canFMSGetPGN(id := can.id);
      IF PGN = 65265 THEN
         CCVS(data := ADDR(buf));
         DebugFmt(message := "speed= \4", v4 := CCVS.speed);
         DebugFmt(message := "clutch= \1", v1 := INT(CCVS.clutch));
         DebugFmt(message := "break= \1", v1 := INT(CCVS.break));
         DebugFmt(message := "cruise= \1", v1 := INT(CCVS.cruise));
         DebugFmt(message := "unknown= \1", v1 := CCVS.val);
      ELSIF PGN = 61444 THEN
         DebugFmt(message := "rpm= \4", v4 := ParseEngineSpeed(data :=
ADDR(buf)));
      END_IF;
   END_IF;
END;

END_PROGRAM;
```

4.2.9.23 canFMSGetPGN (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2 pro, DX4 pro, CX1 pro-c/warp-c, MX2 turbo/encore/warp, NX-200, NX-400, LX2, LX5
Firmware version: 2.60 / 1.00.00

This function can be used to extract the FMS/J1939 PGN from a CAN message received with canReceiveMessage.
To create a CAN filter from a PGN, please see canFMSFilterCreate.

Input:

Id : DINT (16#0...16#FFFFFFFF) (*Default 0*)
The CAN message ID to extract PGN from.

Returns: DINT

>0 - The extracted PGN.

Declaration:

```
FUNCTION canFMSGetPGN : DINT;
VAR_INPUT
   Id    : DINT := 0;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   can   : canReceiveMessage;
   buf   : ARRAY[1..8] OF SINT;
END_VAR;
```

```
FUNCTION_BLOCK ParseCruiseControl;
VAR_INPUT
   data   : PTR;
END_VAR;
VAR_OUTPUT
   speed  : DINT;      // Wheel based speed (SPN84)
   clutch : BOOL;      // Clutch switch (SPN 598), TRUE=pressed,
FALSE=released
   break  : BOOL;      // Break switch (SPN 597), TRUE=pressed, FALSE=released
   cruise : BOOL;      // Cruise control active (SPN 595), TRUE=switched on,
FALSE=switched off
   val    : SINT;      // Unknown data in byte 7. (SPN 976)
END_VAR;
VAR
   can    : ARRAY [1..8] OF SINT;
   temp   : SINT;
   high   : DINT;
END_VAR;
   // Copy CAN data to local buffer
   memcpy(dst := ADDR(can), src := data, len := 8);

   // Extract Wheel based speed.
   speed := shl32(in := can[2], n := 8) + can[3];

   // Extract Clutch switch
   IF shr8(in := can[4], n := 6) = 1 THEN
      clutch := TRUE;
   ELSE
      clutch := FALSE;
   END_IF;

   // Extract Break switch
   temp := shr8(in := can[4], n := 4) AND 16#03;
   IF temp = 1 THEN
      break := TRUE;
   ELSE
      break := FALSE;
   END_IF;

   // Extract Cruise control active
   IF (can[4] AND 16#03) = 1 THEN
      cruise := TRUE;
   ELSE
      cruise := FALSE;
   END_IF;

   // Extract Unknown data
   val := can[7] AND 16#1F;
END_FUNCTION_BLOCK;

FUNCTION ParseEngineSpeed : DINT;
VAR_INPUT
   data : PTR;
END_VAR;
VAR
   can  : ARRAY [1..8] OF SINT;
   tick : DINT;       // Engine speed (SPN190)
END_VAR;
   // Copy CAN data to local buffer
   memcpy(dst := ADDR(can), src := data, len := 8);

   // Extract engine speed
   tick            := shl32(in := can[4], n := 8) + can[5];
   ParseEngineSpeed := tick * 125/1000; // Convert from ticks to RPM's
```

```
END_FUNCTION;

PROGRAM example;
VAR
   PGN  : DINT;
   CCVS : ParseCruiseControl;
END_VAR;

   // Initializing
   canOpen(baud := 250);
   can.data := ADDR(buf);
   canFMSFilterCreate(PGN := 65265);
   canFMSFilterCreate(PGN := 61444);

BEGIN
   can();
   IF can.ready THEN
      PGN := canFMSGetPGN(id := can.id);
      IF PGN = 65265 THEN
         CCVS(data := ADDR(buf));
         DebugFmt(message := "speed= \4", v4 := CCVS.speed);
         DebugFmt(message := "clutch= \1", v1 := INT(CCVS.clutch));
         DebugFmt(message := "break= \1", v1 := INT(CCVS.break));
         DebugFmt(message := "cruise= \1", v1 := INT(CCVS.cruise));
         DebugFmt(message := "unknown= \1", v1 := CCVS.val);
      ELSIF PGN = 61444 THEN
         DebugFmt(message := "rpm= \4", v4 := ParseEngineSpeed(data :=
ADDR(buf)));
      END_IF;
   END_IF;
END;

END_PROGRAM;
```

4.2.9.24 canSetWakeup (Function)

Architecture: NX32L
Device support: NX-200, NX-400, LX2, LX5
Firmware version: 1.36.00

This function is used to configure the system to wake from suspend when a CAN message is received.
It is similar to using the CAN parameter on pmWaitEvent.

pmSuspend return codes for this wake-up source:

Error	Type	Value	Description
True	9	-1	The CAN bus is not open.
True	9	-2	The CAN bus is open but not enabled as a wake-up source.
True	9	-3	The baud rate is not supported.
False	9	1	Wake-up on CAN.

Input:

enable : BOOL (default: TRUE)
Set to true to enable the wake-up, set to false to disable it.

port : SINT (1/2) ***(default 1)***
The port of the CAN bus.

Returns: INT

1 - The system has been configured to wake.
0 - This function is not supported.
-1 - The CAN bus is not present.
-2 - The CAN bus is not open.

Declaration:

```
FUNCTION ALIGN canSetWakeup : INT;
VAR_INPUT
   enable : BOOL := TRUE;
   port   : SINT := 1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Open CAN 1
   canOpen(baud:=500);
   ...
   // Enable wake on CAN 1
   canSetWakeup(port := 1, enable := ON);
   ...
END;
END_PROGRAM;
```

4.2.9.25 canitpOpen (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, NX-200, NX-400, LX2, LX5
Firmware version: 5.13 / 1.80.00

Open an ISO-TP connection using an already opened CAN port.

A callback function is used to notify the application that a message is received. (See CALLBACK for more information)

The ISO-TP protocol uses two way communication so the CAN write capability must be enabled. If CAN write is not enabled, or monitor mode is used, then messages cannot be received or send.

canitpOpen can only be used to create a single connection, to create multiple connections, see canitpSessionCreate.

Input:

port : SINT (1/2) ***(default 1)***
The port of the CAN bus to use for ISO-TP.

src : DINT
The CAN ID to filter for incoming ISO-TP messages.

dst : DINT
The CAN ID where ISO-TP messages are sent to.

xtd : BOOL ***(default TRUE)***

Set to FALSE to use standard identifiers (11 bits), leave at TRUE for extended identifiers (29 bits).

cb_receive : CALLBACK
The callback function for received messages. (See the canitpRecv callback declaration below)

cb_data : PTR
The buffer where the received message is stored.

cb_size : INT *(1..4095)*
The size of the receive buffer.
If an incoming ISO-TP message is too large for the receive buffer it will be discarded.

Returns: INT

1	- Successful.
0	- Not available.
-1	- ISO-TP is open.
-2	- The CAN port is not open
-3	- Illegal source address
-4	- Illegal destination address
-5	- Illegal function pointer
-6	- Illegal buffer or size
-7	- Failed to create CAN filter.

Declaration:

```
FUNCTION canitpOpen  : BOOL;
VAR_INPUT
   port     : SINT := 1;
   src      : DINT;
   dst      : DINT;
   cb_receive : CALLBACK;
   cb_data : PTR;
   cb_size : INT;
   xtd      : BOOL := TRUE;
END_VAR;
```

Callback declaration:

```
FUNCTION CALLBACK canitpRecv
VAR_INPUT
   size     : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   buf    : ARRAY [1..4096] OF SINT;
END_VAR;

FUNCTION CALLBACK canitpRecv
VAR_INPUT
   size   : INT; //Size of the data received.
END_VAR;
   DebugMsg(message := strFromMemory(src := ADDR(buf), len := size));
END_FUNCTION;

PROGRAM example;
VAR
   rc     : INT;
```

```
   str    : STRING;
   out    : ARRAY [1..4095] OF SINT;
   len    : INT;
END_VAR;

   // Open can
   canOpen(baud := 250, monitor := FALSE);

   // Open ISO-TP
   rc := canitpOpen(port     := 1,
                    src      := 16#12345678,
                    dst      := 16#12345679,
                    cb_receive := @canitpRecv,
                    cb_data    := ADDR(buf),
                    cb_size    := SIZEOF(buf));
   IF rc < 1 THEN
      DebugFmt(message := "canitpOpen=\1", v1 := rc);
      RETURN;
   END_IF;

   // Build message
   str := "Hello from RTCU device";

   // Send message
   len := strLen(str := str);
   strToMemory(str := str, dst := ADDR(out), len := len);
   rc  := canitpSend(data := ADDR(out), size := len);
   IF rc < 1 THEN
      DebugFmt(message := "canitpSend=\1", v1 := rc);
      RETURN;
   END_IF;

BEGIN
END;
END_PROGRAM;
```

4.2.9.26 canitpClose (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, NX-200, NX-400, LX2, LX5
Firmware version: 5.13 / 1.80.00

Close an ISO-TP connection previously opened by the canitpOpen function

Input:
None.

Returns: **INT**
1 - Successful.
0 - Not available.
-1 - ISO-TP is not open.

Declaration:
```
FUNCTION canitpClose : INT;
```

Example:

```
INCLUDE rtcu.inc

VAR
```

```
   buf    : ARRAY [1..4096] OF SINT;
END_VAR;

FUNCTION CALLBACK canitpRecv
VAR_INPUT
   size   : INT;
END_VAR;
   DebugMsg(message := strFromMemory(src := ADDR(buf), len := size));
END_FUNCTION;

PROGRAM example;
VAR
   rc     : INT;
END_VAR;

   // Open can
   canOpen(baud := 250, monitor := FALSE);

   // Open ISO-TP
   rc := canitpOpen(port     := 1,
                    src      := 16#12345678,
                    dst      := 16#12345679,
                    received := @canitpRecv,
                    cb_data  := ADDR(buf),
                    cb_size  := SIZEOF(buf));
   IF rc < 1 THEN
      DebugFmt(message := "canitpOpen=\1", v1 := rc);
      RETURN;
   END_IF;

   // ...

   // Close ISO-TP
   rc := canitpClose();
   IF rc < 1 THEN
      DebugFmt(message := "canitpClose=\1", v1 := rc);
      RETURN;
   END_IF;

BEGIN
END;
END_PROGRAM;
```

4.2.9.27 canitpSend (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, NX-200, NX-400, LX2, LX5
Firmware version: 5.13 / 1.80.00

Send an ISO-TP message.
The destination of the message is set in the canitpOpen function, using the dst parameter.

Input:

data : PTR
The buffer with the message to send.

size : INT ***(1..4095)***
The size of the message in bytes.

Returns: INT

1	- Successful.
0	- Not available.
-1	- ISO-TP is not open.
-2	- Illegal data or size.
-3	- Error sending the message.
-11	- The CAN bus is not open.
-14	- Timeout sending the messge.
-15	- The CAN bus is in monitor mode.
-16	- An error is detected on the can bus.

Declaration:

```
FUNCTION canitpSend : INT;
VAR_INPUT
   data : PTR;
   size : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   buf   : ARRAY [1..4096] OF SINT;
END_VAR;

FUNCTION CALLBACK canitpRecv
VAR_INPUT
   size  : INT;
END_VAR;
   DebugMsg(message := strFromMemory(src := ADDR(buf), len := size));
END_FUNCTION;

PROGRAM example;
VAR
   rc    : INT;
   str   : STRING;
   out   : ARRAY [1..4095] OF SINT;
   len   : INT;
END_VAR;

   // Open can
   canOpen(baud := 250, monitor := FALSE);

   // Open ISO-TP
   rc := canitpOpen(port     := 1,
                    src      := 16#12345678,
                    dst      := 16#12345679,
                    received := @canitpRecv,
                    cb_data  := ADDR(buf),
                    cb_size  := SIZEOF(buf));
   IF rc < 1 THEN
      DebugFmt(message := "canitpOpen=\1", v1 := rc);
      RETURN;
   END_IF;

   // Build message
   str := "Hello from RTCU device";

   // Send message
   len := strLen(str := str);
   strToMemory(str := str, dst := ADDR(out), len := len);
   rc  := canitpSend(data := ADDR(out), size := len);
```

```
   IF rc < 1 THEN
      DebugFmt(message := "canitpSend=\1", v1 := rc);
      RETURN;
   END_IF;

BEGIN
END;
END_PROGRAM;
```

4.2.9.28 canitpSessionCreate (Function)

Architecture: NX32L
Device support: NX-200, NX-400, LX2, LX5
Firmware version: 1.96.00

Create an ISO-TP session using an already opened CAN port.

A callback function is used to notify the application that a message is received. (See CALLBACK for more information)

The ISO-TP protocol uses two way communication so the CAN write capability must be enabled. If CAN write is not enabled, or monitor mode is used, then messages cannot be received or send.

A maximum of 30 sessions can be created on a CAN port, if none of its filters are in use by the other functions.
To create more than a total of 16 connections, it is necessary to reduce the size of the internal transmit and receive buffers. If e.g. only sending small messages, the transmit buffer can be reduced significantly, increasing the total number of connections.

Input:

port : SINT (1/2) ***(default 1)***
The port of the CAN bus to use for ISO-TP.

src : DINT
The CAN ID to filter for incoming ISO-TP messages.

dst : DINT
The CAN ID where ISO-TP messages are sent to.

xtd : BOOL ***(default TRUE)***
Set to FALSE to use standard identifiers (11 bits), leave at TRUE for extended identifiers (29 bits).

cb_receive : CALLBACK
The callback function for received messages. (See the canitpRecv callback declaration below)

cb_data : PTR
The buffer where the received message will be stored.

cb_arg : DINT
User argument to pass on to the callback. Can e.g. be used to select the buffer to read from.

cb_size : INT ***(1..4095)***
The size of the buffer to place the received data in.
If an incoming ISO-TP message is too large for the receive buffer it will be discarded.

rx_size : INT ***(6..4095, default 4095)***

The size of the internal receive buffer. Can be reduced to increase the number of simultaneous sessions.

tx_size : INT ***(6..4095, default 4095)***
The size of the internal transmit buffer. Can be reduced to increase the number of simultaneous sessions.

Output:

handle : *SYSHANDLE*
The handle to this ISO-TP session.

Returns: INT

1	- Successful.
0	- Not available.
-2	- The CAN port is not open
-3	- Invalid source address
-4	- Invalid destination address
-5	- Invalid function pointer
-6	- Invalid buffer or size
-7	- Failed to create CAN filter for session.
-8	- Not enough resources for connection, try reducing internal buffer size or closing some of the sessions.

Declaration:

```
FUNCTION canitpSessionCreate  : INT;
VAR_INPUT
   handle  : ACCESS SYSHANDLE;
   src     : DINT;
   dst     : DINT;
   cb_receive : CALLBACK;
   cb_data : PTR;
   cb_arg  : DINT;
   cb_size : INT;
   rx_size : INT;
   tx_size : INT;
   xtd     : BOOL := TRUE;
   port    : SINT := 1;
END_VAR;
```

Callback declaration:

```
FUNCTION CALLBACK canitpRecv
VAR_INPUT
   handle  : SYSHANDLE;
   arg     : DINT;
   size    : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
VAR
   buf   : ARRAY [1..4096] OF SINT;
END_VAR;

FUNCTION CALLBACK canitpRecv
VAR_INPUT
   handle : SYSHANDLE; //Handle to session
   arg    : DINT; //User data
```

```
   size   : INT; //Size of the data received.
END_VAR;
   DebugMsg(message := strFromMemory(src := PTR(arg), len := size));
END_FUNCTION;


PROGRAM example;
VAR
   handle: SYSHANDLE;
   rc    : INT;
   str   : STRING;
   out   : ARRAY [1..4095] OF SINT;
   len   : INT;
END_VAR;

   // Open can
   canOpen(baud := 250, monitor := FALSE);

   // Create ISO-TP session
   rc := canitpSessionCreate(handle:=handle,
                  port     := 1,
                  src      := 16#12345678,
                  dst      := 16#12345679,
                  cb_receive := @canitpRecv,
                  cb_data    := ADDR(buf),
                  cb_arg     := DINT(ADDR(buf)),
                  cb_size    := SIZEOF(buf));
   IF rc < 1 THEN
      DebugFmt(message := "canitpSessionCreate=\1", v1 := rc);
      RETURN;
   END_IF;

   // Build message
   str := "Hello from RTCU device";

   // Send message
   len := strLen(str := str);
   strToMemory(str := str, dst := ADDR(out), len := len);
   rc  := canitpSessionSend(handle:=handle, data := ADDR(out), size := len);
   IF rc < 1 THEN
      DebugFmt(message := "canitpSessionSend=\1", v1 := rc);
      RETURN;
   END_IF;

BEGIN

END;
END_PROGRAM;
```

4.2.9.29 canitpSessionDestroy (Function)

Architecture: NX32L
Device support: NX-200, NX-400, LX2, LX5
Firmware version: 1.96.00

Close an ISO-TP session previously created by the canitpSessionCreate function

Input:

handle : *SYSHANDLE*
The handle to the ISO-TP session to close.

Returns: INT

1 - Successful.
0 - Not available.
-1 - ISO-TP session is not open.
-2 - Invalid handle.

Declaration:

```
FUNCTION canitpSessionDestroy : INT;
VAR_INPUT
   handle  : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
VAR
   buf    : ARRAY [1..4096] OF SINT;
END_VAR;

FUNCTION CALLBACK canitpRecv
VAR_INPUT
   handle : SYSHANDLE; //Handle to session
   arg    : DINT; //User data
   size   : INT; //Size of the data received.
END_VAR;
   DebugMsg(message := strFromMemory(src := PTR(arg), len := size));
END_FUNCTION;


PROGRAM example;
VAR
   handle: SYSHANDLE;
   rc    : INT;
   str   : STRING;
   out   : ARRAY [1..4095] OF SINT;
   len   : INT;
END_VAR;

   // Open can
   canOpen(baud := 250, monitor := FALSE);

   // Create ISO-TP session
   rc := canitpSessionCreate(handle:=handle,
                     port     := 1,
                     src      := 16#12345678,
                     dst      := 16#12345679,
                     cb_receive := @canitpRecv,
                     cb_data    := ADDR(buf),
                     cb_arg     := DINT(ADDR(buf)),
                     cb_size    := SIZEOF(buf));
   IF rc < 1 THEN
      DebugFmt(message := "canitpSessionCreate=\1", v1 := rc);
      RETURN;
   END_IF;

   // ...

   // Close ISO-TP
   rc := canitpSessionDestroy(handle:=handle);
   IF rc < 1 THEN
      DebugFmt(message := "canitpSessionDestroy=\1", v1 := rc);
```

```
      RETURN;
   END_IF;

BEGIN

END;
END_PROGRAM;
```

4.2.9.30 canitpSessionSend (Function)

Architecture: NX32L
Device support: NX-200, NX-400, LX2, LX5
Firmware version: 1.96.00

Send an ISO-TP message.
The destination of the message is set in the canitpSessionCreate function, using the dst parameter.

Input:

handle : SYSHANDLE
The handle of the ISO-TP session to send the data on.

data : PTR
The buffer with the message to send.

size : INT *(1..4095)*
The size of the message in bytes.

Returns: INT

1	- Successful.
0	- Not available.
-1	- Session is not open.
-2	- Invalid parameter.
-3	- Error sending the message.
-11	- The CAN bus is not open.
-14	- Timeout sending the messge.
-15	- The CAN bus is in monitor mode.
-16	- An error is detected on the can bus.

Declaration:

```
FUNCTION canitpSessionSend : INT;
VAR_INPUT
   handle : SYSHANDLE;
   data   : PTR;
   size   : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
VAR
   buf    : ARRAY [1..4096] OF SINT;
END_VAR;

FUNCTION CALLBACK canitpRecv
VAR_INPUT
```

```
   handle : SYSHANDLE; //Handle to session
   arg    : DINT; //User data
   size   : INT; //Size of the data received.
END_VAR;
   DebugMsg(message := strFromMemory(src := PTR(arg), len := size));
END_FUNCTION;


PROGRAM example;
VAR
   handle: SYSHANDLE;
   rc    : INT;
   str   : STRING;
   out   : ARRAY [1..4095] OF SINT;
   len   : INT;
END_VAR;

   // Open can
   canOpen(baud := 250, monitor := FALSE);

   // Create ISO-TP session
   rc := canitpSessionCreate(handle:=handle,
                    port     := 1,
                    src      := 16#12345678,
                    dst      := 16#12345679,
                    cb_receive := @canitpRecv,
                    cb_data    := ADDR(buf),
                    cb_arg     := DINT(ADDR(buf)),
                    cb_size    := SIZEOF(buf));
   IF rc < 1 THEN
      DebugFmt(message := "canitpSessionCreate=\1", v1 := rc);
      RETURN;
   END_IF;

   // Build message
   str := "Hello from RTCU device";

   // Send message
   len := strLen(str := str);
   strToMemory(str := str, dst := ADDR(out), len := len);
   rc  := canitpSessionSend(handle:=handle, data := ADDR(out), size := len);
   IF rc < 1 THEN
      DebugFmt(message := "canitpSessionSend=\1", v1 := rc);
      RETURN;
   END_IF;

BEGIN

END;
END_PROGRAM;
```

4.2.10 display: Display Functions

4.2.10.1 LCD: Display functions

The display functions allows the program to interact with the LCD display found on some of the RTCU devices.
The NX-400 can also use the GUI functions as an alternate, more advanced interface.

- displayBacklight — Controls the backlight of the display.
- displayBacklightWake — Restore the backlight after it has turned off due to inactivity.
- displayBox — Draw a box in the display
- displayCircle — Draw a circle in the display
- displayClear — Clears the display
- displayDefineChar — Define a character.
- displayGetKey — Waits for a key press from the display.
- displayImage — Draw an image in display
- displayLine — Draw a line in the display
- displayNumber — Writes a number at the current position
- displayPoint — Draw point in the display
- displayPower — Control display power
- displayStop — Stop the Display API and restore the IO status screen.
- displayString — Writes text at the current position
- displayXY — Set the current text position
- displayKeySetWakeup — Configures the system to wake from pmSuspend on key press.
- displaySysMenuEnable — Enable or disable the system menu.
- displaySysMenuSetPassword — Control password access to the system menu.

4.2.10.2 displayBacklight (Function)

Architecture: NX32L
Device support: NX-400, LX4
Firmware version: 1.00.00

This function is used to control the backlight intensity of the LCD.
Note: the backlight intensity is retained across power down.
Note: LX4 only has a single intensity, so intensity 1-100 will give the same result.

Input:

intensity : INT (0..100)
The wanted backlight intensity, as an percentage of the full intensity. The intensity will be mapped to the supported backlight intensity levels on the device.

Returns:

None.

Declaration:

```
FUNCTION displayBacklight;
VAR_INPUT
   intensity : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Set backlight to 50%
   displayBacklight(intensity := 50);
   ...
END;

END_PROGRAM;
```

4.2.10.3 displayBacklightWake (Function)

Architecture: NX32L
Device support: NX-400, LX4
Firmware version: 1.93.00

This function is used to restore the backlight after it has turned off due to inactivity when autoOff is enabled on displayPower.

Input:
None.

Returns:
None.

Declaration:
```
FUNCTION displayBacklightWake;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Restore backlight
   displayBacklightWake();
   ...
END;

END_PROGRAM;
```

4.2.10.4 displayBox (Function)

Architecture: X32 / NX32 / NX32L
Device support: DX4 pro, NX-400, LX4
Firmware version: 1.30 / 1.00.00

This function is used to draw a square box in the display.

Input:
x1 : INT (1..136 for DX4, 1..190 for NX-400)
x position (column) of the start corner, 1 is the leftmost.

y1 : INT (1..32 for DX4, 1..99 for NX-400)
y position (row) of the start corner, 1 is the topmost.

x2 : INT (1..136 for DX4, 1..190 for NX-400)
x position (column) of the end corner, 1 is the leftmost.

y2 : INT (1..32 for DX4, 1..99 for NX-400)
y position (row) of the end corner, 1 is the topmost.

fill : BOOL
If true, the inside of the box is filled.

color : INT *(Default 1)*
0 (zero) - Background (OFF).
1 - Black (ON).

Returns:

None.

Declaration:

```
FUNCTION displayBox;
VAR_INPUT
   x1    : INT;
   y1    : INT;
   x2    : INT;
   y2    : INT;
   fill  : BOOL;
   color : INT := 1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   sym1 : ARRAY[1..22] OF SINT := 16#1F, 16#00, 16#20, 16#80, 16#40, 16#40,
16#80, 16#20, 16#80, 16#20, 16#80, 16#20, 16#80, 16#20, 16#80, 16#20, 16#40,
16#40, 16#20, 16#80, 16#1F, 16#00;
   sym2 : ARRAY[1..22] OF SINT := 16#1F, 16#00, 16#24, 16#80, 16#44, 16#40,
16#84, 16#20, 16#84, 16#20, 16#FF, 16#E0, 16#84, 16#20, 16#84, 16#20, 16#44,
16#40, 16#24, 16#80, 16#1F, 16#00;
   sym3 : ARRAY[1..22] OF SINT := 16#1F, 16#00, 16#20, 16#80, 16#60, 16#C0,
16#91, 16#20, 16#8A, 16#20, 16#84, 16#20, 16#8A, 16#20, 16#91, 16#20, 16#60,
16#C0, 16#20, 16#80, 16#1F, 16#00;
END_VAR;

   // Turn on the display
   displayPower(power := ON);

   // Define smile
   displayDefineChar(index := 0, map := "0000180C6666060666660C1800000000");

   // Draw line 1 symbol
   displayCircle(x := 8, y := 8, rad := 5);
   displayLine(x1 := 8, y1 := 3, x2 := 8,  y2 := 13);
   displayLine(x1 := 3, y1 := 8, x2 := 13, y2 := 8);

   // Draw line 2 symbol
   displayImage(x := 3,  y := 19, width := 11, height := 11, data :=
```

```
ADDR(sym3));

   // Show text
   displayXY(x := 3, y := 1);
   displayString(message := "Hello World $80");
   displayXY(x := 3, y := 2);
   displayString(message := "Cookie: ");
   displayXY(x := 11, y := 2);
   displayNumber(number := 4711);

   // Draw bounding
   displayBox(x1 := 1, y1 := 1, x2 := 125, y2 := 31);

BEGIN
   ...
END;
END_PROGRAM;
```

4.2.10.5 displayCircle (Function)

Architecture: X32 / NX32 / NX32L
Device support: DX4 pro, NX-400, LX4
Firmware version: 1.30 / 1.00.00

This function is used to draw a circle in the display.

Input:

x : INT (1..136 for DX4, 1..190 for NX-400)
x position (column) of the center of the circle, 1 is the leftmost.

y : INT (1..32 for DX4, 1..99 for NX-400)
y position (row) of the center of the circle, 1 is the topmost.

rad : INT
The radius of the circle.

color : INT *(Default 1)*
0 (zero) - Background (OFF).
1 - Black (ON).

Returns:

None.

Declaration:

```
FUNCTION displayCircle;
VAR_INPUT
   x     : INT;
   y     : INT;
   rad   : INT;
   color : INT := 1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   sym1 : ARRAY[1..22] OF SINT := 16#1F, 16#00, 16#20, 16#80, 16#40, 16#40,
16#80, 16#20, 16#80, 16#20, 16#80, 16#20, 16#80, 16#20, 16#80, 16#20, 16#40,
```

```
16#40, 16#20, 16#80, 16#1F, 16#00;
   sym2 : ARRAY[1..22] OF SINT := 16#1F, 16#00, 16#24, 16#80, 16#44, 16#40,
16#84, 16#20, 16#84, 16#20, 16#FF, 16#E0, 16#84, 16#20, 16#84, 16#20, 16#44,
16#40, 16#24, 16#80, 16#1F, 16#00;
   sym3 : ARRAY[1..22] OF SINT := 16#1F, 16#00, 16#20, 16#80, 16#60, 16#C0,
16#91, 16#20, 16#8A, 16#20, 16#84, 16#20, 16#8A, 16#20, 16#91, 16#20, 16#60,
16#C0, 16#20, 16#80, 16#1F, 16#00;
END_VAR;

   // Turn on the display
   displayPower(power := ON);

   // Define smile
   displayDefineChar(index := 0, map := "0000180C6666060666660C1800000000");

   // Draw line 1 symbol
   displayCircle(x := 8, y := 8, rad := 5);
   displayLine(x1 := 8, y1 := 3, x2 := 8,  y2 := 13);
   displayLine(x1 := 3, y1 := 8, x2 := 13, y2 := 8);

   // Draw line 2 symbol
   displayImage(x := 3,  y := 19, width := 11, height := 11, data :=
ADDR(sym3));

   // Show text
   displayXY(x := 3, y := 1);
   displayString(message := "Hello World $80");
   displayXY(x := 3, y := 2);
   displayString(message := "Cookie: ");
   displayXY(x := 11, y := 2);
   displayNumber(number := 4711);

   // Draw bounding
   displayBox(x1 := 1, y1 := 1, x2 := 125, y2 := 31);

BEGIN
   ...
END;
END_PROGRAM;
```

4.2.10.6 displayClear (Function)

Architecture: X32 / NX32 / NX32L
Device support: DX4 pro, NX-400, LX4
Firmware version: 1.30 / 1.00.00

This clears the contents of the display. After this call, the current write position will be set to the upper left corner (1,1).

Input:

None.

Returns:

None.

Declaration:

```
FUNCTION displayClear;
```

Example:

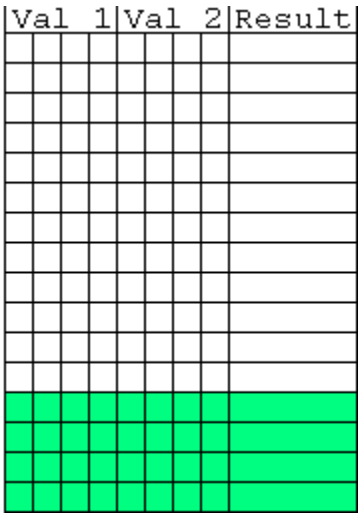
```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Clear the display
   displayClear();
   ...
END;

END_PROGRAM;
```

4.2.10.7 displayDefineChar (Function)

Architecture: X32 / NX32 / NX32L
Device support: DX4 pro, NX-400, LX4
Firmware version: 1.30 / 1.00.00

This function is used to define a character in the display.

The display uses characters 8 pixels wide by 16 pixels high.
To help visualise and calculate the character map, use a table with 8 * 16 cells, where each cell represents a pixel.



(Figure 1. Empty character pixel map)

The last 4 rows (with green background in *Figure 1*) are below the writing line and should only be used if a character similar to "p", "g" or "q" is defined.

Draw the character in the table by using 1's (1's show a pixel) and 0's (0's hide a pixel) as shown in *Figure 2*.

Val 1				Val 2				Result
0	0	0	0	0	0	0	0	00
0	0	0	0	0	0	0	0	00
0	0	0	1	1	0	0	0	18
0	0	0	0	1	1	0	0	0C
0	1	1	0	0	1	1	0	66
0	1	1	0	0	1	1	0	66
0	0	0	0	0	1	1	0	06
0	0	0	0	0	1	1	0	06
0	1	1	0	0	1	1	0	66
0	1	1	0	0	1	1	0	66
0	0	0	0	1	1	0	0	0C
0	0	0	1	1	0	0	0	18
0	0	0	0	0	0	0	0	00
0	0	0	0	0	0	0	0	00
0	0	0	0	0	0	0	0	00
0	0	0	0	0	0	0	0	00

*(Figure 2. Sample character: **Smile**)*

Once the table has been filled, calculate the 2 Hex digits for each line.
The map string in the function contains the Hex digits from the top left and down.
For the sample character in *Figure 2.*, the map string would look like this:
"0000180C6666060666660C1800000000"

Note: only upper case characters can be used and no spacing characters can be included.

Input:

index : INT (0..31)
The index of the character to define.

map : STRING
The calculated map of the character.

Returns: INT

0 - Success.
-1 - The character map is illegal.
-2 - Display not ready.

Declaration:

```
FUNCTION displayDefineChar : INT;
VAR_INPUT
   index : INT;
   map   : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   sym1 : ARRAY[1..22] OF SINT
   := 16#1F, 16#00, 16#20, 16#80, 16#40, 16#40, 16#80, 16#20, 16#80, 16#20,
16#80, 16#20, 16#80, 16#20, 16#80, 16#20, 16#40, 16#40, 16#20, 16#80, 16#1F,
16#00;
   sym2 : ARRAY[1..22] OF SINT
   := 16#1F, 16#00, 16#24, 16#80, 16#44, 16#40, 16#84, 16#20, 16#84, 16#20,
16#FF, 16#E0, 16#84, 16#20, 16#84, 16#20, 16#44, 16#40, 16#24, 16#80, 16#1F,
16#00;
```

```
   sym3 : ARRAY[1..22] OF SINT
   := 16#1F, 16#00, 16#20, 16#80, 16#60, 16#C0, 16#91, 16#20, 16#8A, 16#20,
16#84, 16#20, 16#8A, 16#20, 16#91, 16#20, 16#60, 16#C0, 16#20, 16#80, 16#1F,
16#00;
END_VAR;

   // Turn on the display
   displayPower(power := ON);

   // Define smile
   displayDefineChar(index := 0, map := "0000180C6666060666660C1800000000");

   // Draw line 1 symbol
   displayCircle(x := 8, y := 8, rad := 5);
   displayLine(x1 := 8, y1 := 3, x2 := 8,  y2 := 13);
   displayLine(x1 := 3, y1 := 8, x2 := 13, y2 := 8);

   // Draw line 2 symbol
   displayImage(x := 3,  y := 19, width := 11, height := 11, data :=
ADDR(sym3));

   // Show text
   displayXY(x := 3, y := 1);
   displayString(message := "Hello World $80");
   displayXY(x := 3, y := 2);
   displayString(message := "Cookie: ");
   displayXY(x := 11, y := 2);
   displayNumber(number := 4711);

   // Draw bounding
   displayBox(x1 := 1, y1 := 1, x2 := 125, y2 := 31);

BEGIN
   ...
END;
END_PROGRAM;
```

4.2.10.8 displayGetKey (Function)

Architecture: X32 / NX32 / NX32L
Device support: DX4 pro, NX-400, LX4
Firmware version: 1.30 / 1.00.00

This waits for a key press from the display. The function waits for a maximum number of seconds, and if no key is pressed within the timeout period, the function returns a 0 which indicates the timeout. If a key press has been detected before the timeout period, a number identifying the actual key press is returned.
A small buffer is present in the system so that all key presses are received by the application.

Input:

timeout : INT

-1	- Waits forever until a key is pressed.
0	- Does not wait.
> 0	- Waits the specified number of second (max. 60).

Returns:

-2	- Invalid parameter.
-1	- Not supported.
0	- Timeout.
120	- Up arrow key pressed.
121	- Down arrow key pressed.

122 - Left arrow key pressed.
123 - Right arrow key pressed.
124 - OK key pressed.
125 - ESC key pressed.
126 - DEL key pressed.
127 - SEL key pressed.
200 - NX-400 only: Click happened on screen where there was no control to handle it.
201 - NX-400 only: Click happened on status bar where there was no control to handle it.
202 - NX-400/LX4 only: Backlight reduced due to inactivity. See displayPower for more information.
203 - NX-400/LX4 only: Activity has occurred; the backlight is restored.

Declaration:

```
FUNCTION displayGetKey : INT;
VAR_INPUT
   timeout : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   key : INT;
END_VAR;

BEGIN
   ...
   // Wait for key
   key := displayGetKey(timeout := 45);
   ...
END;

END_PROGRAM;
```

4.2.10.9 displayImage (Function)

Architecture: X32 / NX32 / NX32L
Device support: DX4 pro, NX-400, LX4
Firmware version: 1.30 / 1.00.00

This function is used to draw an image in the display.

Input:

x : INT (1..136 for DX4, 1..190 for NX-400)
x position (column) where the image is drawn from, 1 is the leftmost

y : INT (1..32 for DX4, 1..99 for NX-400)
y position (row) where the image is drawn from, 1 is the topmost

width : INT (1..136 for DX4, 1..190 for NX-400)
The width of the image in pixels.

height : INT (1..32 for DX4, 1..99 for NX-400)
The height of the image in pixels.

data : PTR
The image to draw.

Returns:
None.

Declaration:
```
FUNCTION displayImage;
VAR_INPUT
   x      : INT;
   y      : INT;
   width  : INT;
   height : INT;
   data   : PTR;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   sym1 : ARRAY[1..22] OF SINT := 16#1F, 16#00, 16#20, 16#80, 16#40, 16#40,
16#80, 16#20, 16#80, 16#20, 16#80, 16#20, 16#80, 16#20, 16#80, 16#20, 16#40,
16#40, 16#20, 16#80, 16#1F, 16#00;
   sym2 : ARRAY[1..22] OF SINT := 16#1F, 16#00, 16#24, 16#80, 16#44, 16#40,
16#84, 16#20, 16#84, 16#20, 16#FF, 16#E0, 16#84, 16#20, 16#84, 16#20, 16#44,
16#40, 16#24, 16#80, 16#1F, 16#00;
   sym3 : ARRAY[1..22] OF SINT := 16#1F, 16#00, 16#20, 16#80, 16#60, 16#C0,
16#91, 16#20, 16#8A, 16#20, 16#84, 16#20, 16#8A, 16#20, 16#91, 16#20, 16#60,
16#C0, 16#20, 16#80, 16#1F, 16#00;
END_VAR;

   // Turn on the display
   displayPower(power := ON);

   // Define smile
   displayDefineChar(index := 0, map := "0000180C6666060666660C1800000000");

   // Draw line 1 symbol
   displayCircle(x := 8, y := 8, rad := 5);
   displayLine(x1 := 8, y1 := 3, x2 := 8,  y2 := 13);
   displayLine(x1 := 3, y1 := 8, x2 := 13, y2 := 8);

   // Draw line 2 symbol
   displayImage(x := 3,  y := 19, width := 11, height := 11, data :=
ADDR(sym3));

   // Show text
   displayXY(x := 3, y := 1);
   displayString(message := "Hello World $80");
   displayXY(x := 3, y := 2);
   displayString(message := "Cookie: ");
   displayXY(x := 11, y := 2);
   displayNumber(number := 4711);

   // Draw bounding
   displayBox(x1 := 1, y1 := 1, x2 := 125, y2 := 31);

BEGIN
   ...
END;
END_PROGRAM;
```

4.2.10.10 displayLine (Function)

Architecture: X32 / NX32 / NX32L
Device support: DX4 pro, NX-400, LX4
Firmware version: 1.30 / 1.00.00

This function is used to draw a line in the display.

Input:

x1 : INT (1..136 for DX4, 1..190 for NX-400)
x position (column) of the start point, 1 is the leftmost

y1 : INT (1..32 for DX4, 1..99 for NX-400)
y position (row) of the start point, 1 is the topmost

x2 : INT (1..136 for DX4, 1..190 for NX-400)
x position (column) of the end point, 1 is the leftmost

y2 : INT (1..32 for DX4, 1..99 for NX-400)
y position (row) of the end point, 1 is the topmost

color : INT *(Default 1)*
0 (zero) - Background (OFF).
1 - Black (ON).

Returns:

None.

Declaration:

```
FUNCTION displayLine;
VAR_INPUT
   x1    : INT;
   y1    : INT;
   x2    : INT;
   y2    : INT;
   color : INT := 1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   sym1 : ARRAY[1..22] OF SINT := 16#1F, 16#00, 16#20, 16#80, 16#40, 16#40,
16#80, 16#20, 16#80, 16#20, 16#80, 16#20, 16#80, 16#20, 16#80, 16#20, 16#40,
16#40, 16#20, 16#80, 16#1F, 16#00;
   sym2 : ARRAY[1..22] OF SINT := 16#1F, 16#00, 16#24, 16#80, 16#44, 16#40,
16#84, 16#20, 16#84, 16#20, 16#FF, 16#E0, 16#84, 16#20, 16#84, 16#20, 16#44,
16#40, 16#24, 16#80, 16#1F, 16#00;
   sym3 : ARRAY[1..22] OF SINT := 16#1F, 16#00, 16#20, 16#80, 16#60, 16#C0,
16#91, 16#20, 16#8A, 16#20, 16#84, 16#20, 16#8A, 16#20, 16#91, 16#20, 16#60,
16#C0, 16#20, 16#80, 16#1F, 16#00;
END_VAR;

   // Turn on the display
   displayPower(power := ON);
```

```
   // Define smile
   displayDefineChar(index := 0, map := "0000180C6666060666660C1800000000");

   // Draw line 1 symbol
   displayCircle(x := 8, y := 8, rad := 5);
   displayLine(x1 := 8, y1 := 3, x2 := 8,  y2 := 13);
   displayLine(x1 := 3, y1 := 8, x2 := 13, y2 := 8);

   // Draw line 2 symbol
   displayImage(x := 3,  y := 19, width := 11, height := 11, data :=
ADDR(sym3));

   // Show text
   displayXY(x := 3, y := 1);
   displayString(message := "Hello World $80");
   displayXY(x := 3, y := 2);
   displayString(message := "Cookie: ");
   displayXY(x := 11, y := 2);
   displayNumber(number := 4711);

   // Draw bounding
   displayBox(x1 := 1, y1 := 1, x2 := 125, y2 := 31);

BEGIN
   ...
END;
END_PROGRAM;
```

4.2.10.11 displayNumber (Function)

Architecture: X32 / NX32 / NX32L
Device support: DX4 pro, NX-400, LX4
Firmware version: 1.30 / 1.00.00

This prints a number on the display at the current write position.

Input:

number : DINT
The number to print.

Returns:

None.

Declaration:

```
FUNCTION displayNumber;
VAR_INPUT
   number : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   sym1 : ARRAY[1..22] OF SINT := 16#1F, 16#00, 16#20, 16#80, 16#40, 16#40,
16#80, 16#20, 16#80, 16#20, 16#80, 16#20, 16#80, 16#20, 16#80, 16#20, 16#40,
16#40, 16#20, 16#80, 16#1F, 16#00;
   sym2 : ARRAY[1..22] OF SINT := 16#1F, 16#00, 16#24, 16#80, 16#44, 16#40,
```

```
16#84, 16#20, 16#84, 16#20, 16#FF, 16#E0, 16#84, 16#20, 16#84, 16#20, 16#44,
16#40, 16#24, 16#80, 16#1F, 16#00;
   sym3 : ARRAY[1..22] OF SINT := 16#1F, 16#00, 16#20, 16#80, 16#60, 16#C0,
16#91, 16#20, 16#8A, 16#20, 16#84, 16#20, 16#8A, 16#20, 16#91, 16#20, 16#60,
16#C0, 16#20, 16#80, 16#1F, 16#00;
END_VAR;

   // Turn on the display
   displayPower(power := ON);

   // Define smile
   displayDefineChar(index := 0, map := "0000180C6666060666660C1800000000");

   // Draw line 1 symbol
   displayCircle(x := 8, y := 8, rad := 5);
   displayLine(x1 := 8, y1 := 3, x2 := 8,  y2 := 13);
   displayLine(x1 := 3, y1 := 8, x2 := 13, y2 := 8);

   // Draw line 2 symbol
   displayImage(x := 3,  y := 19, width := 11, height := 11, data :=
ADDR(sym3));

   // Show text
   displayXY(x := 3, y := 1);
   displayString(message := "Hello World $80");
   displayXY(x := 3, y := 2);
   displayString(message := "Cookie: ");
   displayXY(x := 11, y := 2);
   displayNumber(number := 4711);

   // Draw bounding
   displayBox(x1 := 1, y1 := 1, x2 := 125, y2 := 31);

BEGIN
   ...
END;
END_PROGRAM;
```

4.2.10.12 displayPoint (Function)

Architecture: X32 / NX32 / NX32L
Device support: DX4 pro, NX-400, LX4
Firmware version: 1.30 / 1.00.00

This function is used to draw a point in the display.

Input:

x : INT (1..136 for DX4, 1..190 for NX-400)
x position (column) of the point, 1 is the leftmost.

y : INT (1..32 for DX4, 1..99 for NX-400)
y position (row) of the point, 1 is the topmost.

color : INT *(Default 1)*
0 (zero) - Background (OFF).
1 - Black (ON).

Returns:

None.

Declaration:

```
FUNCTION displayPoint;
VAR_INPUT
   x     : INT;
   y     : INT;
   color : INT := 1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Set point
   displayPoint(x := 15, y := 15);
   ...
END;

END_PROGRAM;
```

4.2.10.13 displayPower (Function)

Architecture: X32 / NX32 / NX32L
Device support: DX4 pro, NX-400, LX4
Firmware version: 1.30 / 1.00.00

This sets whether the LCD displays will turn on when external power is present. This has the same effect as setting the LCD display power in Device: Configuration: Power / Battery.

Note: The power state is retained across power down.

On the NX-400 and the LX4, the display can be configured to automatically reduce the intensity of the backlight, when the display is not being actively used. When a key or the display is clicked, a dialog or form is shown or displayBacklightWake is called, it will restore the backlight to the previous level. The backlight changes are available via displayGetKey.
The values for autoOff and offLevel are kept across power down and will be used as the new default values.
Note: offLevel is not used on LX4 as it does not have variable backlight intensity and it will always turn it completely off if autooff is enabled.

Input:

power : BOOL
TRUE: The display will turn ON when external power is present.
FALSE: The display will remain OFF when external power is present.

autoOff : INT *default -1*
>0 Auto off is enabled, the backlight will turn off when the specified number of seconds has elapsed without the display being used.
0: Auto off is disabled, the backlight will stay on as long as the display is powered.
-1: Leave it at the default value.

offLevel : SINT(1..100) *default -1*
0..100: The backlight intensity level to switch to when turning the backlight off. Use 0 to turn the backlight completely off, use a larger value to keep it powered. Only used on NX-400.
-1: Leave it at the default value.

Returns:
None.

Declaration:
```
FUNCTION displayPower;
VAR_INPUT
   power    : BOOL;
   offLevel : SINT := -1;
   autoOff  : INT  := -1;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;

// Turn on the display
displayPower(power := ON);

BEGIN
   ...
END;
END_PROGRAM;
```

4.2.10.114 displayStop (Function)

Architecture: NX32L
Device support: NX-400, LX4
Firmware version: 1.93.00

This function stops showing the custom content on the display and returns it to the IO status screen.

On NX-400 this can be used to allow switching to the GUI functions with guiOpen.

Input:
None.

Returns:
None.

Declaration:
```
FUNCTION displayStop;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   displayString(message := "Hello World");
   ...
   // Restore IO display
```

```
   displayStop();
   // Switch to the GUI forms
   guiOpen();
   ...
END;

END_PROGRAM;
```

4.2.10.15 displayString (Function)

Architecture: X32 / NX32 / NX32L
Device support: DX4 pro, NX-400, LX4
Firmware version: 1.30 / 1.00.00

This prints a text message on the display at the current write position.

Input:

message : STRING
Text string to print.

Returns:

None.

Declaration:

```
FUNCTION displayString;
VAR_INPUT
   message : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   sym1 : ARRAY[1..22] OF SINT := 16#1F, 16#00, 16#20, 16#80, 16#40, 16#40,
16#80, 16#20, 16#80, 16#20, 16#80, 16#20, 16#80, 16#20, 16#80, 16#20, 16#40,
16#40, 16#20, 16#80, 16#1F, 16#00;
   sym2 : ARRAY[1..22] OF SINT := 16#1F, 16#00, 16#24, 16#80, 16#44, 16#40,
16#84, 16#20, 16#84, 16#20, 16#FF, 16#E0, 16#84, 16#20, 16#84, 16#20, 16#44,
16#40, 16#24, 16#80, 16#1F, 16#00;
   sym3 : ARRAY[1..22] OF SINT := 16#1F, 16#00, 16#20, 16#80, 16#60, 16#C0,
16#91, 16#20, 16#8A, 16#20, 16#84, 16#20, 16#8A, 16#20, 16#91, 16#20, 16#60,
16#C0, 16#20, 16#80, 16#1F, 16#00;
END_VAR;

   // Turn on the display
   displayPower(power := ON);

   // Define smile
   displayDefineChar(index := 0, map := "0000180C6666060666660C1800000000");

   // Draw line 1 symbol
   displayCircle(x := 8, y := 8, rad := 5);
   displayLine(x1 := 8, y1 := 3, x2 := 8,  y2 := 13);
   displayLine(x1 := 3, y1 := 8, x2 := 13, y2 := 8);

   // Draw line 2 symbol
   displayImage(x := 3,  y := 19, width := 11, height := 11, data :=
```

```
ADDR(sym3));

   // Show text
   displayXY(x := 3, y := 1);
   displayString(message := "Hello World $80");
   displayXY(x := 3, y := 2);
   displayString(message := "Cookie: ");
   displayXY(x := 11, y := 2);
   displayNumber(number := 4711);

   // Draw bounding
   displayBox(x1 := 1, y1 := 1, x2 := 125, y2 := 31);

BEGIN
   ...
END;
END_PROGRAM;
```

4.2.10.16 displayXY (Function)

Architecture: X32 / NX32 / NX32L
Device support: DX4 pro, NX-400, LX4
Firmware version: 1.30 / 1.00.00

This sets the current write position. This is the position where the next write to the display will take place.

Input:

x : INT (1..18 for DX4, 1..23 for NX-400)
x position (column) 1 is the leftmost.

y : INT (1..2 for DX4, 1..6 for NX-400)
y position (row) 1 is the topmost.

Returns:

None.

Declaration:

```
FUNCTION displayXY;
VAR_INPUT
   x : INT;
   y : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   sym1 : ARRAY[1..22] OF SINT := 16#1F, 16#00, 16#20, 16#80, 16#40, 16#40,
16#80, 16#20, 16#80, 16#20, 16#80, 16#20, 16#80, 16#20, 16#80, 16#20, 16#40,
16#40, 16#20, 16#80, 16#1F, 16#00;
   sym2 : ARRAY[1..22] OF SINT := 16#1F, 16#00, 16#24, 16#80, 16#44, 16#40,
16#84, 16#20, 16#84, 16#20, 16#FF, 16#E0, 16#84, 16#20, 16#84, 16#20, 16#44,
16#40, 16#24, 16#80, 16#1F, 16#00;
   sym3 : ARRAY[1..22] OF SINT := 16#1F, 16#00, 16#20, 16#80, 16#60, 16#C0,
16#91, 16#20, 16#8A, 16#20, 16#84, 16#20, 16#8A, 16#20, 16#91, 16#20, 16#60,
16#C0, 16#20, 16#80, 16#1F, 16#00;
```

```
END_VAR;

   // Turn on the display
   displayPower(power := ON);

   // Define smile
   displayDefineChar(index := 0, map := "0000180C6666060666660C1800000000");

   // Draw line 1 symbol
   displayCircle(x := 8, y := 8, rad := 5);
   displayLine(x1 := 8, y1 := 3, x2 := 8,  y2 := 13);
   displayLine(x1 := 3, y1 := 8, x2 := 13, y2 := 8);

   // Draw line 2 symbol
   displayImage(x := 3,  y := 19, width := 11, height := 11, data :=
ADDR(sym3));

   // Show text
   displayXY(x := 3, y := 1);
   displayString(message := "Hello World $80");
   displayXY(x := 3, y := 2);
   displayString(message := "Cookie: ");
   displayXY(x := 11, y := 2);
   displayNumber(number := 4711);

   // Draw bounding
   displayBox(x1 := 1, y1 := 1, x2 := 125, y2 := 31);

BEGIN
   ...
END;
END_PROGRAM;
```

4.2.10.17 displayKeySetWakeup (Function)

Architecture: NX32L
Device support: NX-400, LX4
Firmware version: 1.36.00

This function is used to configure the system to wake from suspend when the touch screen is pressed.
It is similar to using the key parameter on pmWaitEvent.

pmSuspend return codes for this wake-up source:

Error	Type	ID	Value	Description
False	8	0	1	Wake-up on screen press.

Input:

Enable : BOOL (default: TRUE)
Set to true to enable the wake-up, set to false to disable it.

Returns:

1 - The system has been configured to wake.
0 - This function is not supported.

Declaration:

```
FUNCTION ALIGN displayKeySetWakeup:INT;
VAR_INPUT
   enable: BOOL := TRUE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Enable wake on touch screen press
   displayKeySetWakeup();
   ...
END;

END_PROGRAM;
```

4.2.10.18 displaySysMenuEnable (Function)

Architecture: NX32L
Device support: NX-400, LX4
Firmware version: 1.94.00

This function enables or disables the system menu.
This function is the same as guiSysMenuEnable.
This works similarly to logging in using a password, enabled with displaySysMenuSetPassword.
The system menu can be used to show the status for the device, including network status and RCH status.
On the NX-400, the system menu can also be used to change some settings including calibration of the touch screen and setting the state of the virtual jumpers.

On the NX-400, guiWaitEvent can be used to detect when the menu is enabled:
- If the system menu is enabled, the login succeeded event is sent.
- When logging out, either by clicking "Log out" in the system menu, or when logging out due to the timeout, the logged out event is sent.

On the NX-400, the enabled menu can be access by pressing on the clock shown in the top right corner of the display.
On the LX4, the menu will be shown on the display immediately and can be navigated using the arrow keys and the OK and ESC keys.
When the menu is disabled or exited on the LX4, it will close the menu and switch back to the normal contents of the display.

Input:

enable : BOOL
If true, the system menu is enabled. If false, the system menu is disabled.

timeout: INT (-1..900) ***default -1***
Number of seconds without activity before disabling the menu automatically. 0 disables the timeout, -1 leaves it at its current value.

Returns: **INT**

```
   0    - Success.
  -3    - Invalid timeout
  -5    - Internal error
 -11    - The function is not supported.
```

Declaration:

```
FUNCTION displaySysMenuEnable : INT;
VAR_INPUT
   enable    : BOOL;
   timeout   : INT := -1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Enable system menu with 20 second timeout.
   rc := displaySysMenuEnable(enable := TRUE, timeout := 20);
   ...
END;
END_PROGRAM;
```

4.2.10.119 displaySysMenuSetPassword (Function)

Architecture: NX32L
Device support: NX-400, LX4
Firmware version: 1.94.00

This function configures the password for entering the system menu. If it is disabled, only guiSysMenuEnable and displaySysMenuEnable can be used to enable the system menu.
This function is the same as guiSysMenuSetPassword.
To open the dialog to enter the password on the NX-400, press on the clock in the status bar for about 3 seconds and release.
To open the dialog to enter the password on the LX4, keep the OK key pressed for about 10 seconds.

On the NX-400, guiWaitEvent can be used to detect login attempts:
- If an invalid password is entered, the failed login event is sent, otherwise the login succeeded event is sent.
- When logging out, either by clicking "Log out" in the system menu, or when logging out due to the timeout set with displaySysMenuEnable, the logged out event is sent.

Input:

enable : BOOL
If password based access to the system menu should be allowed.

password : STRING
The password that must be entered on the on screen keyboard, to enable the system menu. Minimum length is 3 characters.
For the LX4, the password must only contain the numbers 0-9 and has a maximum length of 16 characters.

On the LX4, the password can also be empty, which disables the password requirement while still allowing access to the system menu by pressing OK for 10 seconds.

Returns: INT

0	- Success.
-3	- Invalid password
-5	- Internal error
-11	- The function is not supported.

Declaration:

```
FUNCTION displaySysMenuSetPassword : INT;
VAR_INPUT
   enable    : BOOL;
   password  : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Enable login with password "12345"
   rc := displaySysMenuSetPassword(enable := TRUE, password := "12345");
   ...
END;
END_PROGRAM;
```

4.2.11 dtmf: Functions for DTMF interaction

4.2.11.1 DTMF: Functions for DTMF interaction

DTMF functions for access to the DTMF functionality available on certain RTCU variants. Please consult the technical documentation of the actual RTCU device in question.. The DTMF functions can be used to implement interactive voice response systems together with the Voice features.

- dtmfGetKey — Waits for a key press.
- dtmfGetNumber — Waits for a number.

4.2.11.2 dtmfGetKey (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2 pro, DX4 pro, AX9, MX2 turbo/encore/warp, AX9 turbo/encore, NX-200, NX-400, NX-900, NX-910, LX2, LX4, LX5
Firmware version: 1.00 / 1.00.00

This functions waits for a DTMF key press during an active GSM voice session. The function waits for a maximum number of milliseconds, and if no key is pressed within the timeout period, the function returns with -1 which indicates the timeout. If a key press has been detected before the timeout period, a number identifying the actual key press is returned.

Any voice messages being played (or in the queue) with voiceTalk will be canceled when a DTMF key is pressed during a call to dtmfGetKey. Also see voiceStop.

Input:

timeout : INT (0..32767) ***Default 3000***
Timeout period in milliseconds to wait for a DTMF tone.

Returns: INT (-1, 0..15)

-1:	Timeout in waiting for the DTMF tone.
0..9:	DTMF key "0".."9".
10:	DTMF key "*" (star).
11:	DTMF key "#" (hash).
12:	DTMF "A".
13:	DTMF "B".
14:	DTMF "C".
15:	DTMF "D".

Declaration:

```
FUNCTION dtmfGetKey : INT;
VAR_INPUT
   timeout : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Wait for a DTMF key press for maximum 5 seconds
```

```
   CASE dtmfGetKey(timeout:=5000) OF
      // Timeout in waiting for a key press
      -1 :
      ...
      // The key "5" has been pressed
      5  :
      ...
   END_CASE;
END;

END_PROGRAM;
```

4.2.11.3 dtmfGetNumber (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2 pro, DX4 pro, AX9, MX2 turbo/encore/warp, AX9 turbo/encore, NX-200, NX-400, NX-900, NX-910, LX2, LX4, LX5
Firmware version: 1.00 / 1.00.00

This waits for a number typed in via DTMF key during an active GSM voice session. The function waits for a maximum number of milliseconds between each key press, and if no key is pressed within the timeout period, the function returns with -1 which indicates the timeout. If a complete number has been received, this number will be returned to the caller. The input sequence must be terminated by pressing the "#" (hash) key.

Any voice messages being played (or in the queue) with voiceTalk will be canceled when a DTMF key is pressed during a call to dtmfGetNumber. Also see voiceStop.

Input:

timeout : INT (0..32767) ***Default 3000***
Timeout period in milliseconds between DTMF key-press.

Returns: DINT (0..999999999)

-1 : Timeout in waiting for a DTMF key-press or an invalid entry (too long).
Otherwise the number typed.

Declaration:

```
FUNCTION dtmfGetNumber : DINT;
VAR_INPUT
   timeout : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
VAR
   tmp : DINT;
END_VAR;

PROGRAM test;

BEGIN
   ...
   // Wait for a DTMF number, timeout between key presses is set to 5 seconds
   tmp := dtmfGetNumber(timeout:=5000);

   // Check if timeout in waiting for a number
   IF tmp = -1 THEN
```

```
        ...
    // Number received without timeout
    ELSE
        ...
    END_IF;
END;

END_PROGRAM;
```

4.2.12 emu: Emulator functions

4.2.12.1 emu: Emulator functions

The Emulator functions are special functions that can be used to control the operation of the RTCU Emulator.

The following functions are available:

- emuTerminate — Terminates the RTCU Emulator.

4.2.12.2 emuTerminate (Function)

Architecture: ALL
Device support: All emulated devices
Emulator version: 2.12

This function requests that the RTCU Emulator terminates. If successful, it does not return.
It is useful when e.g. using it for running automated tests, where the RTCU Emulator can then be terminated when the test is completed or has failed.
When called on physical devices, it just returns 0 without doing anything.

Input:

exitCode : SINT
The exit code for the RTCU Emulator process to return when terminating.
Note that the RTCU Emulator may decide to return 0 or -1 when terminating by itself.

Returns: INT

0 - Not supported.
-1 - Error

Declaration:

```
FUNCTION emuTerminate : INT;
VAR_INPUT
   exitCode   : SINT;  // Exit code to return from process when terminated.
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc      : INT;
END_VAR;

BEGIN
   ...
   // Terminate the RTCU Emulator with an exit code of 1
   rc := emuTerminate(exitCode := 1);
   ...
END;
END_PROGRAM;
```

4.2.13 ext: Platform Extension functions (NX32L)

4.2.13.1 ext: Extension functions

The **RTCU M2M Platform** is designed to be as open, adaptable and programmable as possible. For most applications the power and flexibility offered with the RTCU IDE is sufficient to realize the most advanced M2M applications, but the need for a deeper customization may arise in certain specialized vertical applications.

Under the **NX32L Architecture** a mechanism to expand the platform on the Linux system level is offered named **Platform Extensions**.
With Platform Extensions it is possible to develop two types of extensions to complement the RTCU IDE application:

Program
A program is an application that operates autonomously in a process separate from the RTCU run-time process. Basically such a program has access to most resources and functionality offered by the Linux operating system.
Specialized applications needed to meet the requirement for advanced protocols, specialized gateways or simple optimized calculation services can be realized.
The recommended way to communicate with a program is to use standard TCP/IP sockets.

Module
Modules are extension libraries that are loaded by the RTCU runtime to natively expand the API with any new functionality that may not be part of the standard API as supplied by Logic IO.
A module follows a rigid and well documented structure as it needs to interface with the MODCALL C binding interface.

Tag
A tag is a set of strings, name and text, which holds extra information about a program or module extension.
For example, the version of the extension or a short description.
An extension can hold several tags.

Module licensing
Some extension modules may require that a license is present on the RTCU device before some or all of the features can be used.
A license is added to a RTCU device by applying a device specific license key. The license key must be acquired from the makers of the module.
For details about how to add support for licensing to an extension module, please see the **RTCU Platform SDK.**

All platform extensions must be installed on the device, before they can be used.
The platform extensions are installed as part of the project transfer, when included in the 'Platform Extension' part of the project view.

The following functions are available to control extension programs:

- extProgramStart — Starts an external program.
- extProgramSignal — Sends a signal to a running program.
- extProgramStatus — Queries the status of a program.

The following functions are available to control extension modules:

- extModuleLoad — Load an extension module.

The following functions are available to handle tags in extension files:

- extTagEnumerate — Read the tag name.
- extTagRead — Read the text of a tag

The following functions are available to handle license keys:

- extLicenseApply — Apply a license key.
- extLicenseCheck — Check if a license is present.
- extLicenseRemove — Remove a license.

For detailed information how to develop Platform Extensions please refer to the dedicated **RTCU Platform SDK**.

4.2.13.2 extProgramStart (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.00.00

This function starts an external program.
The extension program must be installed by adding it to the Platform Extension in the project view, and transferring the project.

The program will start with the working directory set to the root of the internal drive.

Input:

path : STRING
The full path to the program to execute, as shown in the Platform Extension dialog.

args : STRING
A string with the arguments to the program.
The arguments are separated by spaces, use double quotes(") to include a space in an argument.
Currently a maximum of 9 parameters are supported.

Output:

handle : SYSHANDLE
A handle to the started program.

Returns: INT

- 0 - The program was started. The handle parameter now contains a handle to the program.
- -1 - A handle could not be created. Too many program handles may have been created. Use extProgramStatus to release exited programs.
- -2 - Invalid path. Make sure that the path is correct and that the drive is open.
- -3 - Could not start program.
- -4 - Invalid program.
- -6 - Program was build for a different architecture.

Declaration:

```
FUNCTION extProgramStart : INT;
VAR_INPUT
   path   : STRING;
   args   : STRING;
   handle : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc     : INT;
   signal : DINT;
   handle : SYSHANDLE;
END_VAR;

BEGIN
   ...
   fsMediaOpen(media := 1);
   // Start the program SERVER.RPX with four arguments: "-port" "3490" "-name"
and "test server".
   rc := extProgramStart(handle := handle, path := "B:
\SYSTEM\EXT\SERVER.RPX",
      args := "-port 3490 -name $"test server$"");
   DebugFmt(message := "Server started: \1", v1 := rc);
   ...
   // Stop the program by sending the Terminate signal
   rc := extProgramSignal(handle := handle, signal := 9);
   DebugFmt(message := "Send kill: \1", v1 := rc);
   // Request status
   rc := extProgramStatus(handle := handle, code := signal);
   IF rc = 2 THEN
      DebugFmt(message := "Program was terminated with the \4 signal", v4 :=
signal);
   END_IF;
END;
END_PROGRAM;
```

4.2.13.3 extProgramSignal (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.00.00

This function sends a signal to an external program.
This can both be used to stop a running program, and to communicate with the program.
When the program is stopped, extProgramStatus must be called to release the handle.

Input:

handle : SYSHANDLE
A handle to the program to send the signal to.

signal : DINT (default 15)
The signal to send. Common values are:

9	- Kill signal. Kills the program immediately.
10	- User signal 1
12	- User signal 2
15	- Terminate signal. Requests that the program terminates.
18	- Continue signal. Requests that a stopped program continues.
19	- Stop signal. Requests the program to stop.

Returns: INT

0	- The signal was sent to the program.
-1	- Invalid handle.

-2 - Invalid signal.
-3 - Could not find program
-4 - Permission failed.
-5 - Failed to send signal

Declaration:

```
FUNCTION extProgramSignal : INT;
VAR_INPUT
   handle : SYSHANDLE;
   signal : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc     : INT;
   signal : DINT;
   handle : SYSHANDLE;
END_VAR;

BEGIN
   ...
   fsMediaOpen(media := 1);
   // Start the program SERVER.RPX with four arguments: "-port" "3490" "-name"
and "test server".
   rc := extProgramStart(handle := handle, path := "B:
\SYSTEM\EXT\SERVER.RPX",
      args := "-port 3490 -name $"test server$"");
   DebugFmt(message := "Server started: \1", v1 := rc);
   ...
   // Stop the program by sending the Terminate signal
   rc := extProgramSignal(handle := handle, signal := 9);
   DebugFmt(message := "Send kill: \1", v1 := rc);
   // Request status
   rc := extProgramStatus(handle := handle, code := signal);
   IF rc = 2 THEN
      DebugFmt(message := "Program was terminated with the \4 signal", v4 :=
signal);
   END_IF;
END;
END_PROGRAM;
```

4.2.13.4 extProgramStatus (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.00.00

This function checks the status of an external program and to release the handle once the program exits.

Input:

handle : SYSHANDLE
A handle to the program to check the status of.

Output:

handle : SYSHANDLE
If the program is stopped, the handle is released and made invalid.

code : DINT
The status code of an exited program or the signal that caused the status of the program to change.

Returns: INT

3 - The program is stopped. Code contains the signal that made it stop.
2 - The program is terminated because of a signal. Code contains the signal that made it terminate. Handle has been made invalid.
1 - The program has exited normally. Code contains the status code from the program. Handle has been made invalid.
0 - The program is still running.
-1 - Handle is not valid.
-2 - Error checking the status. A likely cause is that the process has ended and the status has already been checked.

Declaration:

```
FUNCTION extProgramStatus : INT;
VAR_INPUT
   handle : ACCESS SYSHANDLE;
   code   : ACCESS DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc     : INT;
   signal : DINT;
   handle : SYSHANDLE;
END_VAR;

BEGIN
   ...
   fsMediaOpen(media := 1);
   // Start the program SERVER.RPX with four arguments: "-port" "3490" "-name"
and "test server".
   rc := extProgramStart(handle := handle, path := "B:
\SYSTEM\EXT\SERVER.RPX",
      args := "-port 3490 -name $"test server$"");
   DebugFmt(message := "Server started: \1", v1 := rc);
   ...
   // Stop the program by sending the Terminate signal
   rc := extProgramSignal(handle := handle, signal := 9);
   DebugFmt(message := "Send kill: \1", v1 := rc);
   // Request status
   rc := extProgramStatus(handle := handle, code := signal);
   IF rc = 2 THEN
      DebugFmt(message := "Program was terminated with the \4 signal", v4 :=
signal);
   END_IF;
END;
END_PROGRAM;
```

4.2.13.5 extModuleLoad (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.04.00

This function will load and initialize an extension module.
The extension module must be installed by adding it to the Platform Extension in the project view, and transferring the project.

Input:

path : STRING
The full path to the module, as shown in the Platform Extension dialog.

Returns: INT

1	- The module is loaded.
0	- The function is not available.
-1	- Invalid path. Make sure that the path is correct and that the drive is open.
-2	- The file in path is not a valid module.
-3	- No more modules can be loaded.
-4	- The module could not be loaded.
-5	- Failed to initialize module.
-6	- Module was build for a different architecture.

Declaration:

```
FUNCTION extModuleLoad : INT;
VAR_INPUT
   path   : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc     : INT;
   status : DINT;
   handle : SYSHANDLE;
END_VAR;

   // Initialize
   fsMediaOpen(media := 1);

   // Load modules
   rc := extModuleLoad(path := "b:\system\ext\example.rmx");
   IF rc < 1 THEN
      DebugFmt(message := "extModuleLoad=\1", v1 := rc);
   END_IF;

BEGIN
END;
END_PROGRAM;
```

4.2.13.6 extTagEnumerate (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.00.00

This function is used to retrieve the name of the tags in a platform extension program or module.

Input:

path : STRING
The full path to the program or module to enumerate, as shown in the Platform Extension dialog.

index : INT
The index of the tag to get. (One based)

Output:

tag : STRING
The name of the tag.

Returns: INT

1	- Success.
0	- Not available.
-1	- Invalid path. Make sure that the path is correct and that the drive is open.
-2	- Invalid program or module.
-3	- No more tags.

Declaration:

```
FUNCTION extTagEnumerate : INT;
VAR_INPUT
   path   : STRING;
   index  : INT;
   tag    : ACCESS STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

FUNCTION ShowTag;
VAR_INPUT
   path     : STRING;
END_VAR;
VAR
   tag      : STRING;
   text     : STRING;
   i, rc    : INT;
END_VAR;

   i := 0;

   DebugMsg(message := "Extension:     " + path);
   REPEAT
      // Setup
      i := i + 1;

      // Get tag
      rc := extTagEnumerate(path := path, index := i, tag := tag);
      IF rc > 0 THEN
```

```
            DebugMsg(message := "Tag:");
            DebugMsg(message := tag);
            rc := extTagRead(path := path, tag := tag, text := text);
            IF rc > 0 THEN
               DebugMsg(message := "Text:");
               DebugMsg(message := text);
            ELSE
               DebugFmt(message := "extReadTag=\1", v1 := rc);
            END_IF;
         ELSE
            DebugFmt(message := "extTagEnumerate=\1", v1 := rc);
         END_IF;
      UNTIL rc < 1
      END_REPEAT;
      DebugMsg(message := "-------------------------------------------------");

END_FUNCTION;

PROGRAM test;

   // Initialize
   fsMediaOpen(media := 1);

   // Show all the tags from the extension
   ShowTag(path := "B:\SYSTEM\EXT\SERVER.RPX");


END_PROGRAM;
```

4.2.13.7 extTagRead (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.00.00

This function is used to read the value of a tag in a platform extension program or module.

Input:

path : STRING
The full path to the program or module, as shown in the Platform Extension dialog.

tag : STRING
The name of the tag.

Output:

text : STRING
The value of the tag.

Returns: INT

1 - Success.
0 - Not available.
-1 - Invalid path. Make sure that the path is correct and that the drive is open.
-2 - Invalid program or module.
-3 - The tag is not found.

Declaration:

```
FUNCTION extTagRead : INT;
VAR_INPUT
   path   : STRING;
   tag    : STRING;
```

```
   text    : ACCESS STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

FUNCTION ShowTag;
VAR_INPUT
   path     : STRING;
END_VAR;
VAR
   tag      : STRING;
   text     : STRING;
   i, rc    : INT;
END_VAR;

   i := 0;

   DebugMsg(message := "Extension:     " + path);
   REPEAT
      // Setup
      i := i + 1;

      // Get tag
      rc := extTagEnumerate(path := path, index := i, tag := tag);
      IF rc > 0 THEN
         DebugMsg(message := "Tag:");
         DebugMsg(message := tag);
         rc := extTagRead(path := path, tag := tag, text := text);
         IF rc > 0 THEN
            DebugMsg(message := "Text:");
            DebugMsg(message := text);
         ELSE
            DebugFmt(message := "extReadTag=\1", v1 := rc);
         END_IF;
      ELSE
         DebugFmt(message := "extTagEnumerate=\1", v1 := rc);
      END_IF;
   UNTIL rc < 1
   END_REPEAT;
   DebugMsg(message := "--------------------------------------------------");

END_FUNCTION;

PROGRAM test;

   // Initialize
   fsMediaOpen(media := 1);

   // Show all the tags from the extension
   ShowTag(path := "B:\SYSTEM\EXT\SERVER.RPX");


END_PROGRAM;
```

4.2.13.8 extLicenseApply (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.80.00

The extLicenseApply function is used to apply a license key for an extension module.
Instead of using this function, the license key can also be applied via the Apply upgrade key dialog.

Input:
key : STRING
The license key that contains the license to apply.
The license key is a text containing at least 358 characters.
Because of this, Large String Support must be enabled to use the function.

Returns: INT
1 - The license key is applied
0 - The function is not available.
-1 - Invalid license key.
-2 - Failed to apply.
-3 - Generic error.

Declaration:
```
FUNCTION extLicenseApply : INT;
VAR_INPUT
   key : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   key : STRING;
END_VAR;

   // Get the license key
   key :=
"@0ARQBkRm6q2V4YW1wbGUubGljZW5zZQCi/rdpCtubIgas0VW0VgpvuR1AJYfWeqWNX3O5wv6rwIK
d4EGGJRB0wjgZGj2wjhH/FnH85DVhZ8hcJicZzTckl5iZz811aooAwis4/ulhrJvMiMDMHIJoYMQFr
awrgRAs7fWsdT0jjV/Y80+m6n6mncok0BYicXIninpdZDqDW9DplYMHqZftbyjJX3d7XIEKOkA+
+7cGOJzTqwSMRaPwqiZqUKDiZtsU3rzpGUnJtnyX4X2pwSz3kA0MmWoy70l3XqYGhJudI2AuiSwht4
LFPGj0h8otXkZsDhjY1YzlX+9PHQptHBvsTCKr1GHKllMDdjThA3YEU+DibijGrYhV";

   // Apply the license key
   extLicenseApply(key := key);


END_PROGRAM
```

4.2.13.9 extLicenseCheck (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.80.00

This function will check if a license is present.

Input:
id : STRING
The license id to check for.

The license id is a text of up to 50 characters, which identifies the license.

Returns: INT

1	- The license is present
0	- The function is not available.
-1	- The license is not present
-3	- Generic error.

Declaration:

```
FUNCTION extLicenseCheck : INT;
VAR_INPUT
   id : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

FUNCTION ShowLicense;
VAR_INPUT
   id       : STRING;
END_VAR;
VAR
   text     : STRING;
   rc       : INT;
END_VAR;

   // Check license
   rc := extLicenseCheck(id := id);

   // Build
   CASE rc OF
      1:  str := "Present";
      0:  str := "Not available";
      -1: str := "Not present";
      -3: str := "Generic error";
   ELSE
      str := strFormat(format := "Unknown \1", v1 := rc);
   END_CASE;

   DebugMsg(message := "+------------------------------+");
   DebugMsg(message := "license        : " + str);

END_FUNCTION;

PROGRAM test;


   // Show if the license is present
   ShowLicense(id := "example.license");


END_PROGRAM;
```

4.2.13.10 extLicenseRemove (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.80.00

This function will remove a license.

Input:

id : STRING
The license id string to remove.
The license id is a text of up to 50 characters, which identifies the license.
To remove all licenses from the device, use `"*"` as the id.

Returns: INT

1 - The license is removed
0 - The function is not available.

Declaration:

```
FUNCTION extLicenseRemove : INT;
VAR_INPUT
   id : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;


   // Remove a license
   extLicenseRemove(id := "example.license");


END_PROGRAM;
```

4.2.14 fs: File system

4.2.14.1 fs: File system functions

The file system functions offers support for standard Windows FAT16/32 format 8.3 file systems. This provides applications with the opportunity to have an almost unlimited storage capacity for extensive data logging, databases, user configurations, etc.
By using standard SD-CARD technology or USB drives, it is straightforward to exchange data with any Windows PC (or any other FAT16/32-compatible platform).

This table show how the medias are identified in VPL:

Media	ID	Drive
SD-CARD	0	A:
Internal drive	1	B:
Project drive	2	P:
USB drive	3	U:

The drives are addressed by using the drive name A:, B:, P: or U:, for example "B:\DOC\MYLOG.TXT".

Functions operating on the media:

- fsMediaPresent — Determines if the media is present.
- fsMediaWriteProtected — Determines if the media is write-protected.
- fsMediaOpen — Opens a media.
- fsMediaClose — Closes a media.
- fsMediaQuickFormat — Formats a media.
- fsMediaEject — Ejects a media.
- fsMediaSize — Retrieves the size of the media.
- fsMediaFree — Retrieves the amount of the free space on the media.
- fsStatusLEDEnable — Enables or disables status-LED.

Functions operating on directories:

- fsDirCreate — Create a new directory.
- fsDirChange — Change directory.
- fsDirCurrent — Retrieves current directory path.
- fsDirCatalog — Lists all directory items (files and directories).
- fsDirDelete — Deletes a directory.

Functions operating on files:

- fsFileCreate — Creates a new file.
- fsFileOpen — Opens an existing file.
- fsFileExists — Determines if a file exists.
- fsFileRename — Renames a file.
- fsFileMove — Moves a file.
- fsFileDelete — Deletes a file.
- fsFileStatus — Retrieves status of an opened file.

• fsFileGetCreateTime	Retrieves the time of file creation.
• fsFileGetSize	Retrieves the size of a file.
• fsFileSeek	Moves the file pointer.
• fsFilePosition	Gets the current file pointer position.
• fsFileRead	Reads a block of data from the file.
• fsFileReadString	Reads a string from the file.
• fsFileWrite	Writes a block of data to the file.
• fsFileWriteString	Writes a string to a file.
• fsFileWriteStringNL	Writes a string that includes a line break at the end to a file.
• fsFileClose	Closes a file.
• fsFileFlush	Flushes cached write operations to the media.

Definitions:

Absolute path	A path to a file or directory that includes all sub-directories from the root directory, e.g. "\dir1\dir2\file.txt".
Relative path	A path to a file or directory relative to the current directory, e.g. "dir1\dir2\file.bin" or "..\dir1\file.dat".

File system Consistency

The firmware does everything possible to maintain the consistency of the file system. It is ensured that shutting down an operation or performing a controlled reset will automatically be handled by the firmware so that the file system is shut down in a consistent state.
Media removal without the use of fsMediaEject or hardware reset should be avoided as this can result in an inconsistent state of the file system and, in a worst case scenario, even loss of data.
In case of an uncontrolled shutdown, or reinserting a non-ejected media, the firmware will automatically check and repair the file system. As this phase can take some time, the file system will be unavailable until the completion of the operation. The file system API will return a -22 (media is busy) error code during this phase.
In case of a repair operation, the fault entry "Drive ***X***: has been repaired due to unexpected shutdown." will be present in the fault log after the repair has completed, where ***X*** is the letter of the media.

The implementation of the FAT format 8.3 file system has the following limitations:

- The absolute path is limited to 60 characters.
- It is possible to have 16 files open simultaneously.
- The file name is restricted to a maximum length of 8 characters plus the 3 file type characters. The "." separator is not included.
- The file name must not contain spaces or special characters. "-" or "_" are both allowed, though.
- A file with a long file name can be accessed with the "~" character, e.g. "Long Filename.txt" is accessed as "LONGFI~1.TXT".
- A directory name must not contain "." or any other special characters as stated above.
- "\" (backslash) is used as a directory separator.
- It supports standard SD-CARDs (non-SDHC) with a capacity of up to 2 GB and SD-CARDs (SDHC) with a capacity of up to 32 GB.
- If more than one partition is present on the media, the file system will use the first.
- There is a separate working directory for each thread. When the thread is first started, the working directory is the root of the drive.

Using the Internal Flash drive.

NX32L devices:

The capacity of the internal drive of an NX32L devices ranges from 61 MB to 4 GB and utilizes different types of flash technology.

The total number of writable bytes are limited for any flash technology and is calculated as follows:

TBW = (CAP * PE) / WAF

TBW is the Total Bytes Written before exceeding the guaranteed endurance of the device.
CAP is the size of the partition where the wear-leveling is performed by the system or media controller. This size will often be different than the size of the internal drive due to compression and other data at the same partition.
PE is the Program/Erase factory that depends on the NAND flash technology used.
WAF (Write Amplification Factor) is between 4 and 8, that depends on the applications write behavior.
For example, large sequential writes produce a lower WAF, while random writes of small data blocks produce a higher WAF. Typically WAF is set to 8.

The values for CAP and PE are as follows:

Device	Capacity (CAP)	Program/Erase (PE)
RTCU NX-200 / NX-400 / NX-900	270 MB	100000
RTCU LX5 pro / LX5 eco	180 MB	80000
RTCU LX4 pro	61 MB	80000
RTCU LX2 pro	4000 MB	3000

Assuming that WAF=8 then TBW can be calculated for each device:

Device	Calculation	TBW (MB)
RTCU NX-200 / NX-400 / NX-900	(270*100000) / 8	3375000
RTCU LX5 pro / LX5 eco	(180*80000) / 8	1800000
RTCU LX4 pro	(61*80000) / 8	610000
RTCU LX2 pro	(4000*3000) / 8	1500000

For example, a workload featuring 500 MB of daily write usage cycle will reach end of life as follows:

Device	Calculation	Duration
RTCU NX-200 / NX-400 / NX-900	3375000 / 500	6750 days = ~18 years
RTCU LX5 eco / LX5 pro	1800000 / 500	3600 days = ~9 years
RTCU LX4 pro	610000 / 500	1220 days = ~3 years
RTCU LX2 pro	1500000 / 500	3000 days = ~8 years

The above table shows that care must be taken for write data intensive applications and on the RTCU LX4 pro it is recommended to use an SD-CARD for such applications.

X32/NX32 devices:
The size of the internal flash drive is specified in the data sheet and/or technical manual of the device in question.
When using the internal flash drive, it must be observed that the flash memory technology used is the same as used for Persistent Memory and Datalogger storage. For this reason, there is a limited number of write operations possible, and the following must therefore be observed:

- The file system will automatically perform wear leveling - thus swapping heavily used sectors in the flash with lesser used sectors.
- The size of a sector is 512 bytes.

Using a device with 4 MB or 8 MB internal flash drive the wear leveling, combined with the effective write throughput, basically means that there is no write endurance limitations to be observed.

For devices with a small internal flash drive of 512 KBytes the write endurance limitation must however be observed.
512 KBytes translates into initially 1020 usable sectors, each sector can be rewritten a minimum 100,000 times before wear-out occurs.

The following number of write operations can be expected on the Internal Flash drive:

- With a file OPEN, WRITE and CLOSE operation on the same 512 bytes file, a total number of approx. 51 million write operations can be made.
- With a file CREATE, WRITE and CLOSE operation on a 512 bytes file, a total number of approx. 25.5 million write operations can be made.

- With a file OPEN, WRITE and CLOSE operation on the same 1024 bytes file, a total number of approx. 34 million write operations can be made.
- With a file CREATE, WRITE and CLOSE operation on a 1024 bytes file, a total number of approx. 20.4 million write operations can be made.

Writing every 10 seconds, 24 hours a day, will allow CREATE, WRITE and CLOSE operations on a 1024 bytes file for more than 6 years!

Using the Intellisync Project Drive.
The Intellisync Project Drive is read-only from VPL, and does not support directories. To write to the project drive, it must be synchronized using e.g. the RTCU IDE, using User files. It is not possible to read VSX files from the project drive.

4.2.14.2 fsMediaPresent (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This queries if the media is inserted (present) in the RTCU device.

Input:

media : INT (0..3)
The media to query.

Returns: BOOL

TRUE: Media is present.
FALSE: Media is not present.

Declaration:

```
FUNCTION fsMediaPresent : BOOL;
VAR_INPUT
   media : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM FileExample;
VAR
```

```
   mediaPresent : BOOL;
   rc           : INT;
END_VAR;

fsMediaOpen(media := 0);

BEGIN
   // Update media status
   IF mediaPresent <> fsMediaPresent(media := 0) THEN
      IF fsMediaPresent(media := 0) THEN
         IF fsMediaWriteProtected(media := 0) THEN
            DebugMsg(message := "Media present, media write protected!");
         END_IF;
         // Open files
         ...
      ELSE
         DebugMsg(message := "Media has been removed!");
      END_IF;
      // Save current status
      mediaPresent := fsMediaPresent(media := 0);
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.14.3 fsMediaWriteProtected (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This queries if a media is write protected. If the media is write protected, it is not possible to write any information to it (read-only mode), and it is not possible to create, modify, or delete any files or directories.

The write protection on an SD-CARD is determined by the small "tab" on the edge of the card. This "tab" is only present on the full-size SD-CARD - therefore devices with a micro-SD-CARD reader will never report the media as write protected.

USB medias can be write protected for multiple reasons, including by setting physical write protection switches or, especially in the case of flash medias, in the case of errors on the file system.

Input:

media : INT (0..3)
The media to query.

Returns: BOOL

TRUE: Media is write protected (files and directories are read only).
FALSE: Media is not write protected.

Declaration:

```
FUNCTION fsMediaWriteProtected : BOOL;
VAR_INPUT
   media : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM FileExample;
VAR
   mediaPresent : BOOL;
   rc           : INT;
END_VAR;

fsMediaOpen(media := 0);

BEGIN
   // Update media status
   IF mediaPresent <> fsMediaPresent(media := 0) THEN
      IF fsMediaPresent(media := 0) THEN
         IF fsMediaWriteProtected(media := 0) THEN
            DebugMsg(message := "Media present, media write protected!");
         END_IF;
         // Open files
         ...
      ELSE
         DebugMsg(message := "Media has been removed!");
      END_IF;
      // Save current status
      mediaPresent := fsMediaPresent(media := 0);
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.14.4 fsMediaOpen (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This opens the media for use by the file system. Calling this function is required before the file system can be used.
Once a media is opened with this function, that media is available for all threads.

This table show how the medias are identified in VPL:

Media	ID	Drive
SD-CARD	0	A:
Internal drive	1	B:
Project drive	2	P:
USB drive	3	U:

Please note that the default media at device start-up is always the SD-CARD (A:) drive. This means that if only the internal drive is used, the current drive must be changed from A: to B: by using fsDirChange or absolute path specifications must be used.

Also see fsMediaClose for closing the media when finished with the operations.

Input:

media : INT (0..3)
The media to open.

Returns: INT

0	Media is opened.
-1	Invalid media ID.
-15	The specified media is not available.

Declaration:

```
FUNCTION fsMediaOpen : INT;
VAR_INPUT
   media : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   fd : FILE;
END_VAR;

fsMediaOpen(media := 0);
BEGIN
   ...
   IF fsFileExists(name := "gpslog.dat") THEN
      fd := fsFileOpen(name := "gpslog.dat");
   ELSE
      fd := fsFileCreate(name := "gpslog.dat");
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.14.5 fsMediaClose (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This closes the media. Use this function to disable communication to the media. After the media is closed, the file system cannot be used until the media is opened again.

Please also see fsMediaOpen (Function).

Input:

media : INT (0..3)
The media to close.

Returns:

None

Declaration:

```
FUNCTION fsMediaClose;
VAR_INPUT
   media : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   filesys : BOOL;
END_VAR;

PROGRAM test;
VAR
   fs_open : BOOL;
END_VAR;

BEGIN
   ...
   IF filesys THEN
      IF NOT fs_open THEN
         IF fsMediaOpen(media := 0) <> 0 THEN
            DebugMsg(message := "fsMediaOpen - Failed!");
         ELSE
            fsStatusLEDEnable(enable := ON);
            fs_open := TRUE;
         END_IF;
      END_IF;
   ELSE
      IF fs_open THEN
         fsMediaClose(media := 0);
         fsStatusLEDEnable(enable := OFF);
         fs_open := FALSE;
      END_IF;
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.14.6 fsMediaQuickFormat (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This performs a quick format of the media.

The formatting performed is a Quick Format, which means that all files and directories are erased but a full format with full sector initialization will not be performed.

Input:

media : INT (0..3)
The media to format.

full : BOOL *(default: FALSE)*
When TRUE a full format of the drive is performed.
Important: This parameter is only used for the internal drive of NX32L devices.

Returns: INT

0	- Media is formatted.
-1	- Media is not open.
-16	- Media not present.

-17 - Media communication error (card not supported).
-22 - Media is busy.
-23 - Media is write protected.
-24 - Unknown file system.
-39 - The media is busy because a platform extension is used.

Declaration:

```
FUNCTION fsMediaQuickFormat : INT;
VAR_INPUT
   media : INT;
   full  : BOOL := FLASE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   cmd : STRING;
END_VAR;

PROGRAM FileExample;

// Open media
fsMediaOpen(media := 0);

BEGIN
   ...
   IF cmd = "format" THEN
      rc := fsMediaQuickFormat(media := 0);
      IF rc = 0 THEN
         DebugMsg(message := "Media formatted!");
      ELSE
         DebugFmt(message := "Media formatting failed! (\1)", v1 := rc);
      END_IF;
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.14.7 fsMediaEject (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.02 / 1.00.00

This function ejects the media.

After this function is called, all files are closed, and the media is seen as not being present.

Input:

media : INT (0..3)
The media to eject.

Returns: INT

0 - Media is ejected.
-1 - Media cannot be ejected.
-16 - Media not present.

Declaration:

```
FUNCTION fsMediaEject : INT;
VAR_INPUT
   media : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   eject : BOOL;
END_VAR;

PROGRAM FileExample;
VAR
   eject_old : BOOL;
   rc : INT;
END_VAR;

   ...
   IF eject THEN
      IF NOT eject_old THEN
         rc := fsMediaEject(media := 0);
         IF rc = 0 THEN
            DebugMsg(message := "Media ejected!");
         ELSE
            DebugMsg(message := "Media not present");
         END_IF;
         eject_old := TRUE;
      END_IF;
   ELSE
      IF eject_old THEN
         eject_old := FALSE;
      END_IF;
   END_IF;
   ...

END_PROGRAM;
```

4.2.14.8 fsMediaSize (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 4.10 / 1.00.00

This function returns the size of the media.
It may take some time for large media.

Input:

media : INT (0..3)
The media to examine.

Returns: DINT

Size of the media in KB
-1 - Media not found
-16 - Media not present.
-24 - Unknown file system.

Declaration:

```
FUNCTION fsMediaSize : DINT;
VAR_INPUT
   media : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

VAR
   a  : DINT;
END_VAR;

BEGIN
   ...
   // Get the size of the internal drive
   a := fsMediaSize(media := 1);
   ...
END;

END_PROGRAM;
```

4.2.14.9 fsMediaFree (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 4.10 / 1.00.00

This function returns the amount of free space on the media.
It may take some time for large media.

Input:

media : INT (0..3)
The media to examine.

Returns: DINT

Size of the free space in KB
-1 - Media not found
-16 - Media not present.
-24 - Unknown file system.

Declaration:

```
FUNCTION fsMediaFree : DINT;
VAR_INPUT
   media : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

VAR
   a  : DINT;
END_VAR;
```

```
BEGIN
   ...
   // Get the amount of free space on the internal drive
   a := fsMediaFree(media := 1);
   ...
END;

END_PROGRAM;
```

4.2.14.10 fsDirCreate (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This creates a new directory.

The fsDirCreate function does not work recursively and only tries to create the directory. If any of the directories in the path (if absolute) does not exist, the function will fail.

Input:

name : STRING
Name of the directory to create. Both absolute and relative paths are allowed.

Returns: INT

0	- Directory created.
-1	- Media not open.
-2	- Media is not formatted.
-3	- Path of the new directory is invalid.
-4	- Name of the new directory is invalid.
-6	- Directory already exists.
-7	- Media is full.
-16	- Media not present.
-17	- Media communication error (card not supported).
-22	- Media is busy.
-23	- Media is write protected.
-24	- Unknown file system.

Declaration:

```
FUNCTION fsDirCreate : INT;
VAR_INPUT
   name : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc : INT;
END_VAR;

BEGIN
   ...
   // Create new directory
   rc := fsDirCreate(name := "\data\gps");
```

```
   IF rc = 0 THEN
      ...
   ELSE
      // Failed to create directory
      ...
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.14.11 fsDirChange (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This function changes the working directory and/or the current media.
To change the media, insert the drive in the path.

Note: the change will only take effect for the calling thread. The working directory for all other threads is not changed.

Input:

path : STRING
Path of the new directory. Both absolute and relative paths are allowed.

Returns: INT

0	- Directory changed.
-1	- Media not open.
-2	- Media is not formatted.
-4	- Name of directory is invalid.
-5	- Directory does not exists.
-16	- Media not present.
-17	- Media communication error (card not supported).
-22	- Media is busy.
-24	- Unknown file system.

Declaration:

```
FUNCTION fsDirChange : INT;
VAR_INPUT
   path : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc : INT;
END_VAR;

BEGIN
   ...
   // Change directory
   rc := fsDirChange(path := "..\Docs\logs");
```

```
   IF rc = 0 THEN
      ...
   ELSE
      // Failed to change directory
      ...
   END_IF;
   ...
   // Change media
   rc := fsDirChange(path := "B:\");
   ...
END;

END_PROGRAM;
```

4.2.14.112 fsDirCurrent (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This returns the absolute path to the current working directory of the calling thread - including the media.

Input:
None.

Returns: STRING
The current directory or empty if an error occurred (media is not open or has been removed without closing it).

Declaration:
```
FUNCTION fsDirCurrent : STRING;
```

Example:
```
INCLUDE rtcu.inc

FUNCTION directory_dump;
VAR
   i   : INT;
   str : STRING;
END_VAR;
   // Print current directory
   DebugMsg(message := "Directory: " + fsDirCurrent());
   // Iterate directory
   REPEAT
      str := fsDirCatalog(index := i);
      IF strLen(str := str) > 0 THEN
         DebugMsg(message := str);
      END_IF;
      i := i + 1;
   UNTIL strLen(str := str) = 0
   END_REPEAT;
   // If directory is empty, print it
   IF i = 1 THEN
      DebugMsg(message := "<EMPTY>");
   END_IF;
END_FUNCTION;
```

4.2.14.13 fsDirCatalog (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

---

fsDirCatalog is used to return the contents of the current directory - including sub-directories and files.
When a sub-directory or file is created it is registered in the file system. The file system does not sort the contents by type (file or directory) or alphabetically but it assigns an index number to the file or directory as it is created. This index number is the input of the fsDirCatalog function, and it will return the name of the file or directory the index refers to.

The index number of a file or sub-directory may change when the content of a directory is modified, for example when files or sub-directories are created, deleted or moved.

Note: The working directory is on a per thread basis.

Input:

index : INT (1..n)
Index entry to return. The first entry is 1.

Returns: STRING

Name of file or directory at the index specified. Empty if none.

Declaration:

```
FUNCTION fsDirCatalog : STRING;
VAR_INPUT
   index : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

FUNCTION directory_dump;
VAR
   i   : INT;
   str : STRING;
END_VAR;
   // Print current directory
   DebugMsg(message := "Directory: " + fsDirCurrent());
   // Iterate directory
   REPEAT
      str := fsDirCatalog(index := i);
      IF strLen(str := str) > 0 THEN
         DebugMsg(message := str);
      END_IF;
      i := i + 1;
   UNTIL strLen(str := str) = 0
   END_REPEAT;
   // If directory is empty, print it
   IF i = 1 THEN
      DebugMsg(message := "<EMPTY>");
   END_IF;
END_FUNCTION;
```

4.2.14.14 fsDirDelete (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This function will delete the specified directory.

Input:

name : STRING
Name of the directory to delete. Both absolute and relative paths are allowed.

Returns: INT

0	- Directory deleted.
-1	- Media not open.
-2	- Media is not formatted.
-3	- Path to directory invalid.
-4	- Name of directory invalid.
-5	- Directory does not exist.
-14	- Directory is not empty.
-16	- Media not present.
-17	- Media communication error (card not supported).
-22	- Media is busy.
-23	- Media is write protected.
-24	- Unknown file system.

Declaration:

```
FUNCTION fsDirDelete : INT;
VAR_INPUT
   name : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc : INT;
END_VAR;

BEGIN
   ...
   // Delete directory
   rc := fsDirDelete(name := "\Logs");
   IF rc = 0 THEN
      ...
   ELSE
      // Delete failed
      ...
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.14.15 fsFileCreate (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This function will create a new file and return a FILE descriptor (ID) for the new file. When the function returns, it is necessary to verify that the file has been successfully created with the fsFileStatus function. The data pointer used when reading and writing data to the file will be initialized to point at the beginning of the file.

Note: if a file with the name already exists, it will be overwritten without warning; use fsFileExists to check if the file already exists.

Input:

name : STRING
Name of the file to create. Both absolute and relative paths are allowed.

Returns: FILE

FILE descriptor. Use the fsFileStatus to verify the operation.

Declaration:

```
FUNCTION fsFileCreate : FILE;
VAR_INPUT
   name : STRING;
END_VAR;
```

Example:

```
...
IF fsFileExists(name := "gpslog.dat") THEN
   fd := fsFileOpen(name := "gpslog.dat");
ELSE
   fd := fsFileCreate(name := "gpslog.dat");
END_IF;
IF fsFileStatus(fd := fd) = 0 THEN
   // File open, write data
   ...
   fsFileWrite(fd := fd, buffer := ADDR(gps.latitude), length :=
SIZEOF(gps.latitude));
   fsFileWrite(fd := fd, buffer := ADDR(gps.longitude), length :=
SIZEOF(gps.longitude));
   ...
   fsFileClose(fd := fd);
ELSE
   // Error when opening/creating file
   ...
END_IF;
...
```

4.2.14.16 fsFileOpen (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This will open an existing file for reading and writing. The function returns a FILE descriptor (ID) for the file, and it is necessary to verify that the file has been successfully opened with the fsFileStatus function. The data pointer used when reading and writing data will always point at the end of file when opened.
It is possible to have 16 files open simultaneously - each with a unique FILE descriptor.

Input:
name : STRING
Name of the file to open. Both absolute and relative paths are allowed.

Returns: FILE
FILE descriptor. Use the fsFileStatus to verify the operation.

Declaration:
```
FUNCTION fsFileOpen : FILE;
VAR_INPUT
   name : STRING;
END_VAR;
```

Example:

```
...
IF fsFileExists(name := "gpslog.dat") THEN
   fd := fsFileOpen(name := "gpslog.dat");
ELSE
   fd := fsFileCreate(name := "gpslog.dat");
END_IF;
IF fsFileStatus(fd:=fd) = 0 THEN
   // File open, write data
   ...
   fsFileWrite(fd := fd, buffer := ADDR(gps.latitude), length :=
SIZEOF(gps.latitude));
   fsFileWrite(fd := fd, buffer := ADDR(gps.longitude), length :=
SIZEOF(gps.longitude));
   ...
   fsFileClose(fd := fd);
ELSE
   // Error when opening/creating file
   ...
END_IF;
...
```

4.2.14.17 fsFileExists (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This will query if a file already exists.

Input:
name : STRING
Name of the file to query. Both absolute and relative paths are allowed.

Returns: BOOL
TRUE: File does exist.

FALSE: File does not exist.

Declaration:

```
FUNCTION fsFileExists : BOOL;
VAR_INPUT
   name : STRING;
END_VAR;
```

Example:

```
...
IF fsFileExists(name := "gpslog.dat") THEN
   fd := fsFileOpen(name := "gpslog.dat");
ELSE
   fd := fsFileCreate(name := "gpslog.dat");
END_IF;
IF fsFileStatus(fd := fd) = 0 THEN
   // File open, write data
   ...
   fsFileWrite(fd := fd, buffer := ADDR(gps.latitude), length :=
SIZEOF(gps.latitude));
   fsFileWrite(fd := fd, buffer := ADDR(gps.longitude), length :=
SIZEOF(gps.longitude));
   ...
   fsFileClose(fd := fd);
ELSE
   // Error when opening/creating file
   ...
END_IF;
...
```

4.2.14.18 fsFileRename (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This will rename a file.

Input:

name_old : STRING
Name of the file to rename. Both absolute and relative paths are allowed.

name_new : STRING
The new name of the file. No path information can be included in the new name as the file remains at the same path.

Returns: INT

0	- File is renamed.
-1	- Media not open.
-2	- Media is not formatted.
-3	- Path to file invalid.
-4	- Name of file (new or old) is illegal.
-5	- File not found.
-6	- A file with this name already exist.
-12	- File is open.
-16	- Media not present.
-17	- Media communication error (card not supported)

-22 - Media is busy
-23 - Media is write protected.
-24 - Unknown file system.

Declaration:

```
FUNCTION fsFileRename : INT;
VAR_INPUT
   name_old : STRING;
   name_new : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc : INT;
END_VAR;

BEGIN
   ...
   // Rename file
   rc := fsFileRename(name_old := "\prg1.log", name_new := "prg2.log");
   IF rc = 0 THEN
      ...
   ELSE
      // Rename failed
      ...
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.14.19 fsFileDelete (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This deletes a file on the media.

Input:

name : STRING
Name of the file to delete. Both absolute and relative paths are allowed.

Returns: INT

0 - File deleted.
-1 - Media not open.
-2 - Media is not formatted.
-3 - Path to file is not found.
-4 - Name of file is illegal.
-5 - File not found.
-12 - File is open.
-16 - Media not present.
-17 - Media communication error (card not supported).
-22 - Media is busy.

-23 - Media is Write protected.
-24 - Unknown file system.

Declaration:

```
FUNCTION fsFileDelete : INT;
VAR_INPUT
   name : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc : INT;
END_VAR;

BEGIN
   ...
   // Delete file
   rc := fsFileDelete(name := "\prg1.log");
   IF rc = 0 THEN
      ...
   ELSE
      // Delete failed
      ...
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.14.20 fsFileStatus (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This function will return the status on the specified FILE after it has been opened or created.

Input:

fd : FILE
FILE descriptor for the file retrieved from fsFileOpen or fsFileCreate .

Returns: INT

0 - File is open and ready for use.
-1 - Media not open.
-7 - Media is full.
-8 - File is not open.
-16 - Media not present.
-17 - Media communication error (card not supported).
-22 - Media is busy.
-23 - Media is write protected.
-24 - Unknown file system.
-33 - Too many files open.
-38 - File is no longer accessible and must be closed.

Declaration:

```
FUNCTION fsFileStatus : INT;
VAR_INPUT
   fd : FILE;
END_VAR;
```

Example:

```
...
IF fsFileExists(name := "gpslog.dat") THEN
   fd := fsFileOpen(name := "gpslog.dat");
ELSE
   fd := fsFileCreate(name := "gpslog.dat");
END_IF;
IF fsFileStatus(fd := fd) = 0 THEN
   // File open, write data
   ...
   fsFileWrite(fd := fd, buffer := ADDR(gps.latitude), length :=
SIZEOF(gps.latitude));
   fsFileWrite(fd := fd, buffer := ADDR(gps.longitude), length :=
SIZEOF(gps.longitude));
   ...
   fsFileClose(fd := fd);
ELSE
   // Error when opening/creating file
   ...
END_IF;
...
```

4.2.14.21 fsFileGetCreateTime (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This function will return the file creation time as a LINSEC value.

Input:

name : STRING
The name of the file.

Returns: DINT

The time of creation for the file in seconds since 1980-1-1 00:00:00 (LINSEC).

Declaration:

```
FUNCTION fsFileGetCreateTime : DINT;
VAR_INPUT
   name : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

VAR
```

```
   a  : DINT;
END_VAR;

BEGIN
   ...
   // Get the time of creation as number of seconds since 1980-1-1 00:00:00
   a := fsFileGetCreateTime(name := "test.txt");
   ...
END;

END_PROGRAM;
```

4.2.14.22 fsFileGetSize (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.07 / 1.00.00

This function will return the size of the file specified.

Input:

name : STRING
The name of the file.

Returns: DINT

The size of the file in bytes or the following error codes:

0	- File is empty or not present.
-1	- Media not open.
-2	- Media is not formatted.
-5	- Could not find file.
-16	- Media not present.
-17	- Media communication error (card not supported).
-22	- Media is busy.
-24	- Unknown file system.

Declaration:

```
FUNCTION fsFileGetSize : DINT;
VAR_INPUT
   name : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

VAR
   a  : DINT;
END_VAR;

BEGIN
   ...
   // Get the size of the file
   a := fsFileGetSize(name := "test.txt");
   ...
```

```
END;

END_PROGRAM;
```

4.2.14.23 fsFileSeek (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

---

fsFileSeek will move the FILE data pointer to an address offset. The function is used for data addressing in the file before read or write operations. All data is addressed as 8 bit, and after each successful read or write operation, the file data pointer is automatically incremented.

Input:

fd : FILE
FILE descriptor for the file retrieved from fsFileOpen or fsFileCreate.

offset : DINT
Offset from start of file to move the data pointer to.
>= 1 - Offset value.
0 - Start of file.
-1 - End of file.

Returns: INT

0 - File pointer moved.
-8 - File not open.
-17 - Media communication error (card not supported).
-22 - Media is busy.
-34 - Offset value too large.
-38 - File is no longer accessible and must be closed.

Declaration:

```
FUNCTION fsFileSeek : INT;
VAR_INPUT
   fd     : FILE;
   offset : DINT;
END_VAR;
```

Example:

```
...
IF fsFileExists(name := "\gpslog.dat") THEN
   fdGpslog := fsFileOpen(name := "\gpslog.dat");
   IF fsFileStatus(fd := fdGpslog) = 0 THEN
      // Is file empty?
      IF fsFilePosition(fd := fdGpslog) > 0 THEN
         rc := fsFileSeek(fd := fsGpslog, offset := 0);
         IF rc = 0 THEN
            // Iterate file
            REPEAT
               // Read data fields
               len1 := fsFileRead(fd := fdGpslog, buffer := ADDR(linsec),
length := SIZEOF(linsec));
               len2 := fsFileRead(fd := fdGpslog, buffer := ADDR(lat), length
:= SIZEOF(lat));
               len3 := fsFileRead(fd := fdGpslog, buffer := ADDR(lon), length
:= SIZEOF(lon));
```

```
               // Is data valid?
               IF len1 = SIZEOF(linsec) AND len2 = SIZEOF(lat) AND len3 =
SIZEOF(lon) THEN
                  // Data is read
                  ...
               END_IF;
            UNTIL len1 <> SIZEOF(linsec) OR len2 <> SIZEOF(lat) OR len3 <>
SIZEOF(lon)
            END_REPEAT;
         ELSE
            // Error
         END_IF;
      ELSE
         // No data
      END_IF;
      fsFileClose(fd := fdGpslog);
   ELSE
      // Could not open file
   END_IF;
ELSE
   // File dont exist
END_IF;
...
```

4.2.14.24 fsFilePosition (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This will return the position of the FILE data pointer. The function can be used to determine the size of the file in conjunction with fsFileSeek (with offset set to -1). All data is addressed as 8 bit, and after each successful read or write operation, the file data pointer is automatically incremented.

Input:

fd : FILE
FILE descriptor for the file retrieved from fsFileOpen or fsFileCreate.

Returns: DINT

Position of file pointer.

Declaration:

```
FUNCTION fsFilePosition : DINT;
VAR_INPUT
  fd : FILE;
END_VAR;
```

Example:

```
...
IF fsFileExists(name := "\gpslog.dat") THEN
   fdGpslog := fsFileOpen(name := "\gpslog.dat");
   IF fsFileStatus(fd := fdGpslog) = 0 THEN
      // Is file empty?
      IF fsFilePosition(fd := fdGpslog) > 0 THEN
         rc := fsFileSeek(fd := fsGpslog, offset := 0);
         IF rc = 0 THEN
```

```
            // Iterate file
            REPEAT
               // Read data fields
               len1 := fsFileRead(fd := fdGpslog, buffer := ADDR(linsec),
length := SIZEOF(linsec));
               len2 := fsFileRead(fd := fdGpslog, buffer := ADDR(lat), length
:= SIZEOF(lat));
               len3 := fsFileRead(fd := fdGpslog, buffer := ADDR(lon), length
:= SIZEOF(lon));
               // Is data valid?
               IF len1 = SIZEOF(linsec) AND len2 = SIZEOF(lat) AND len3 =
SIZEOF(lon) THEN
                  // Data is read
                  ...
               END_IF;
            UNTIL len1 <> SIZEOF(linsec) OR len2 <> SIZEOF(lat) OR len3 <>
SIZEOF(lon)
            END_REPEAT;
         ELSE
            // Error
         END_IF;
      ELSE
         // No data
      END_IF;
      fsFileClose(fd := fdGpslog);
   ELSE
      // Could not open file
   END_IF;
ELSE
   // File dont exist
END_IF;
...
```

4.2.14.25 fsFileRead (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This function reads a block of data from a FILE and returns the number of bytes read. All data is addressed as 8 bit, and after each successful read operation, the file data pointer is automatically incremented.

Please see fsFileReadString for string read.

Input:

fd : FILE
FILE descriptor for the file retrieved from fsFileOpen or fsFileCreate.

buffer : PTR
Address of the buffer to receive the data in.

length : INT
Number of bytes to read.

Returns: INT

Number of bytes read from file.

Declaration:

```
FUNCTION fsFileRead : INT;
VAR_INPUT
   fd     : FILE;
   buffer : PTR;
   length : INT;
END_VAR;
```

Example:

```
...
IF fsFileExists(name := "\gpslog.dat") THEN
   fdGpslog := fsFileOpen(name := "\gpslog.dat");
   IF fsFileStatus(fd := fdGpslog) = 0 THEN
      // Is file empty?
      IF fsFilePosition(fd := fdGpslog) > 0 THEN
         rc := fsFileSeek(fd := fsGpslog, offset := 0);
         IF rc = 0 THEN
            // Iterate file
            REPEAT
               // Read data fields
               len1 := fsFileRead(fd := fdGpslog, buffer := ADDR(linsec),
length := SIZEOF(linsec));
               len2 := fsFileRead(fd := fdGpslog, buffer := ADDR(lat), length
:= SIZEOF(lat));
               len3 := fsFileRead(fd := fdGpslog, buffer := ADDR(lon), length
:= SIZEOF(lon));
               // Is data valid?
               IF len1 = SIZEOF(linsec) AND len2 = SIZEOF(lat) AND len3 =
SIZEOF(lon) THEN
                  // Data is read
                  ...
               END_IF;
            UNTIL len1 <> SIZEOF(linsec) OR len2 <> SIZEOF(lat) OR len3 <>
SIZEOF(lon)
            END_REPEAT;
         ELSE
            // Error
         END_IF;
      ELSE
         // No data
      END_IF;
      fsFileClose(fd := fdGpslog);
   ELSE
      // Could not open file
   END_IF;
ELSE
   // File dont exist
END_IF;
...
```

4.2.14.26 fsFileReadString (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This reads a string from a FILE - starting from the location set by the data pointer (fsFileSeek). The length of the string will either be the maximum string length as defined by the STRING variable or a carriage return value (0D hex or 13 decimal) followed by a new line value (0A hex or 10 decimal). After each successful read operation, the file data pointer is automatically incremented

Input:
fd : FILE
FILE descriptor for the file retrieved from fsFileOpen or fsFileCreate.

Returns: STRING
The string read. Or an empty string if none is present.

Declaration:
```
FUNCTION fsFileReadString : STRING;
VAR_INPUT
   fd : FILE;
END_VAR;
```

Example:

```
...
IF fsFileExists(name := "\prg1.log") THEN
   fdLog := fsFileOpen(name := "\prg1.log");
   IF fsFileStatus(fd := fdLog) = 0 THEN
      // Is file empty?
      IF fsFilePosition(fd := fdLog) > 0 THEN
         rc := fsFileSeek(fd := fdLog, offset := 0);
         IF rc = 0 THEN
            // Iterate file
            REPEAT
               str := fsFileReadString(fd := fdLog);
               IF strLen(str := str) > 0 THEN
                  // Data is read
                  ...
               END_IF;
            UNTIL strLen(str := str) = 0
            END_REPEAT;
         ELSE
            // Error
         END_IF;
      ELSE
         // No data
      END_IF;
      fsFileClose(fd := fdLog);
   ELSE
      // Could not open file
   END_IF;
ELSE
   // File dont exist
END_IF;
...
```

4.2.14.27 fsFileWrite (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This function will write a buffer to a FILE - starting from the location set by the file data pointer (changed with fsFileSeek). After each successful write operation the file datapointer is automatically incremented.

Please also see fsFileWriteString for string writing.

Input:

fd : FILE
FILE descriptor for the file retrieved from fsFileOpen or fsFileCreate.

buffer : PTR
Address of the buffer that contains the data.

length : INT
Number of bytes to write.

Returns: INT

Number of bytes written to file.

Declaration:

```
FUNCTION fsFileWrite : INT;
VAR_INPUT
   fd     : FILE;
   buffer : PTR;
   length : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   fdLog    : FILE;
   doLog    : BOOL := TRUE;
END_VAR;

FUNCTION WriteLog;
VAR_INPUT
   text : STRING;
END_VAR;
VAR
   rc   : INT;
END_VAR;

   IF fsFileStatus(fd := fdLog) = 0 THEN
      // Write text to log
      rc := fsFileWriteStringNL(fd := fdLog, str := text);
      IF rc <> strLen(str := text) THEN
         DebugFmt(message := "error \1 when writing string to prg1.log", v1 :=
rc);
         doLog := FALSE;
         fsFileClose(fd := fdLog);
      END_IF;
   END_IF;

END_FUNCTION;

PROGRAM FileExample;
VAR
   fdGpslog : FILE;
   gps      : gpsFix;
   doGpsLog : BOOL := TRUE;
   iter     : DINT := 1;
   str      : STRING;
   rc       : INT;
```

```
END_VAR;

// Media
fsMediaOpen(media := 0);
gpsPower(power := ON);

// Open file for program log
IF fsFileExists(name := "prg1.log") THEN
   fdLog := fsFileOpen(name := "prg1.log");
ELSE
   fdLog := fsFileCreate(name := "prg1.log");
END_IF;
IF fsFileStatus(fd := fdLog) <> 0 THEN
   DebugFmt(message := "File $"prg1.log$" not open, error code=\1", v1 :=
fsFileStatus(fd := fdLog));
   doGpsLog := FALSE;
END_IF;

// Open file for log of GPS positions
IF fsFileExists(name := "gpslog.dat") THEN
   fdGpslog := fsFileOpen(name := "gpslog.dat");
ELSE
   fdGpslog := fsFileCreate(name := "gpslog.dat");
END_IF;
IF fsFileStatus(fd := fdGpslog) <> 0 THEN
   DebugFmt(message := "File $"gpslog.dat$" not open, error code=\1", v1 :=
fsFileStatus(fd := fdGpslog));
END_IF;

BEGIN
   gps();
   IF gps.mode > 1 THEN
      // Log
      IF doLog THEN
         WriteLog(text := "GPS position calculated");
      END_IF;
      // log position
      IF doGpsLog THEN
         fsFileWrite(fd := fdGpslog, buffer := ADDR(gps.linsec), length :=
SIZEOF(gps.linsec));
         fsFileWrite(fd := fdGpslog, buffer := ADDR(gps.latitude), length :=
SIZEOF(gps.latitude));
         fsFileWrite(fd := fdGpslog, buffer := ADDR(gps.longitude), length :=
SIZEOF(gps.longitude));
      END_IF;
   END_IF;

   // Log
   IF doLog THEN
      str := strFormat(format := "Iteration \4", v4 := iter);
      WriteLog(text := str);
   END_IF;

   Sleep(delay := 5000);
   iter := iter + 1;
END;

END_PROGRAM;
```

4.2.14.28 fsFileWriteString (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This will write a string to the FILE - starting from the location set by the file data pointer (changed with fsFileSeek). After each successful write operation, the file data pointer is automatically incremented.

For automatic insertion of new lines after each string write, see fsFileWriteStringNL.

Input:

fd : FILE
FILE descriptor for the file retrieved from fsFileOpen or fsFileCreate.

str : STRING
The string to be written to the file.

Returns: INT

Number of bytes written to file.

Declaration:

```
FUNCTION fsFileWriteString : INT;
VAR_INPUT
   fd  : FILE;
   str : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   LogReader : logRead;
END_VAR;

FUNCTION logExport;
VAR_INPUT
   filename : STRING;
END_VAR;
VAR
   fd  : FILE;
   str : STRING;
   i   : INT;
   col : INT;
END_VAR;

   // Create file
   fd := fsFileCreate(name := filename);
   IF fsFileStatus(fd := fd) <> 0 THEN
      DebugFmt(message := "Could not open file, error code=\1", v1 :=
fsFileStatus(fd := fd));
      RETURN;
   END_IF;

   // Goto first entry
   logFirst();
```

```
   LogReader();
   col := logValuesPerRecord();

   // Iterate log
   WHILE LogReader.status = 0 DO
      // Format Date
      str := strFormat(format := "\1-\2-\3,", v1 := LogReader.day, v2 :=
LogReader.month, v3 := LogReader.year);
      fsFileWriteString(fd := fd, str := str);
      // Format Time
      str := strFormat(format := "\1:\2:\3,", v1 := LogReader.hour, v2 :=
LogReader.minute, v3 := LogReader.second);
      fsFileWriteString(fd := fd, str := str);
      // Format Tag
      str := sintToStr(v := LogReader.tag);
      fsFileWriteString(fd := fd, str := str);
      // Iterate values
      FOR i := 1 TO col DO
         // Format value
         str := strFormat(format := ",\4", v4 := LogReader.value[i]);
         fsFileWriteString(fd := fd, str := str);
      END_FOR;
      // Terminate entry
      fsFileWriteString(fd := fd, str := "$N");
      // Next entry
      IF NOT logNext() THEN EXIT; END_IF;
      LogReader();
   END_WHILE;

   // Close file
   fsFileClose(fd := fd);

END_FUNCTION;
```

4.2.14.29 fsFileWriteStringNL (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This is the same as the fsFileWriteString, but it will add a carriage return value (0D hex or 13 decimal) followed by a new line value (0A hex or 10 decimal) in the end of the string. Each write will start from the location set by the file data pointer (changed with fsFileSeek). After each successful write operation, the file data pointer is automatically incremented.

Input:

fd : FILE
FILE descriptor for the file retrieved from fsFileOpen or fsFileCreate.

str : STRING
The string to be written to the file.

Returns: INT

Number of bytes written to file.

Declaration:

```
FUNCTION fsFileWriteStringNL : INT;
VAR_INPUT
   fd  : FILE;
```

```
   str : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   fdLog    : FILE;
   doLog    : BOOL := TRUE;
END_VAR;

FUNCTION WriteLog;
VAR_INPUT
   text : STRING;
END_VAR;
VAR
   rc   : INT;
END_VAR;

   IF fsFileStatus(fd := fdLog) = 0 THEN
      // Write text to log
      rc := fsFileWriteStringNL(fd := fdLog, str := text);
      IF rc <> (strLen(str := text) + 2) THEN
         DebugFmt(message := "error \1 when writing string to prg1.log", v1 :=
rc);
         doLog := FALSE;
         fsFileClose(fd := fdLog);
      END_IF;
   END_IF;

END_FUNCTION;

PROGRAM FileExample;
VAR
   fdGpslog : FILE;
   gps      : gpsFix;
   doGpsLog : BOOL := TRUE;
   iter     : DINT := 1;
   str      : STRING;
   rc       : INT;
END_VAR;

// Media
fsMediaOpen(media := 0);
gpsPower(power := ON);

// Open file for program log
IF fsFileExists(name := "prg1.log") THEN
   fdLog := fsFileOpen(name := "prg1.log");
ELSE
   fdLog := fsFileCreate(name := "prg1.log");
END_IF;
IF fsFileStatus(fd := fdLog) <> 0 THEN
   DebugFmt(message := "File $"prg1.log$" not open, error code=\1", v1 :=
fsFileStatus(fd := fdLog));
   doGpsLog := FALSE;
END_IF;

// Open file for log of GPS positions
IF fsFileExists(name := "gpslog.dat") THEN
   fdGpslog := fsFileOpen(name := "gpslog.dat");
ELSE
```

```
   fdGpslog := fsFileCreate(name := "gpslog.dat");
END_IF;
IF fsFileStatus(fd := fdGpslog) <> 0 THEN
   DebugFmt(message := "File $"gpslog.dat$" not open, error code=\1", v1 :=
fsFileStatus(fd := fdGpslog));
END_IF;

BEGIN
   gps();
   IF gps.mode > 1 THEN
      // Log
      IF doLog THEN
         WriteLog(text := "GPS position calculated");
      END_IF;
      // log position
      IF doGpsLog THEN
         fsFileWrite(fd := fdGpslog, buffer := ADDR(gps.linsec), length :=
SIZEOF(gps.linsec));
         fsFileWrite(fd := fdGpslog, buffer := ADDR(gps.latitude), length :=
SIZEOF(gps.latitude));
         fsFileWrite(fd := fdGpslog, buffer := ADDR(gps.longitude), length :=
SIZEOF(gps.longitude));
      END_IF;
   END_IF;

   // Log
   IF doLog THEN
      str := strFormat(format := "Iteration \4", v4 := iter);
      WriteLog(text := str);
   END_IF;

   Sleep(delay := 5000);
   iter := iter + 1;
END;

END_PROGRAM;
```

4.2.14.30 fsFileClose (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This closes a file. When all read and write operations are done, close the file to avoid accidental write operations if the media is removed or an error occurs in the program to avoid data corruption.

Input:

fd : FILE
FILE descriptor for the file retrieved from fsFileOpen or fsFileCreate.

Returns: INT

0	- File closed.
-8	- File not open.
-16	- Media not present.
-22	- Media is busy.

Declaration:

```
FUNCTION fsFileClose : INT;
VAR_INPUT
   fd : FILE;
END_VAR;
```

Example:

```
...
IF fsFileExists(name:="gpslog.dat") THEN
   fd := fsFileOpen(name:="gpslog.dat");
ELSE
   fd := fsFileCreate(name:="gpslog.dat");
END_IF;
IF fsFileStatus(fd:=fd) = 0 THEN
   // File open, write data
   ...
   fsFileWrite
(fd:=fd,buffer:=ADDR(gps.latitude),length:=SIZEOF(gps.latitude));
   fsFileWrite
(fd:=fd,buffer:=ADDR(gps.longitude),length:=SIZEOF(gps.longitude));
   ...
   fsFileClose(fd:=fd);
ELSE
   // Error when opening/creating file
   ...
END_IF;
...
```

4.2.14.3 fsFileFlush (Function)

1

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.05 / 1.00.00

This function writes any cached data to the media.
To improve performance, the file system caches data written by the application and only physically writes data to the media after several writes operations or when the file is closed. Calling fsFileFlush is equal to closing the file and opening it again - except that the current position in the file is maintained.

Input:

fd : FILE

FILE descriptor for the file retrieved from fsFileOpen or fsFileCreate.

Returns: **INT**

0	- File flushed.
-8	- File not open.
-16	- Media not present.
-22	- Media is busy.
-38	- File is no longer accessible and must be closed.

Declaration:

```
FUNCTION fsFileflush : INT;
VAR_INPUT
   fd : FILE;
END_VAR;
```

Example:

```
...
IF fsFileStatus(fd:=fd) = 0 THEN
   // File open, write data
   ...
   fsFileWrite
(fd:=fd,buffer:=ADDR(gps.latitude),length:=SIZEOF(gps.latitude));
   fsFileWrite
(fd:=fd,buffer:=ADDR(gps.longitude),length:=SIZEOF(gps.longitude));
   fsFileFlush(fd:=fd);
   ...
END_IF;
...
```

4.2.14.32 fsStatusLEDEnable (Function)

Architecture: X32 / NX32
Device support: All
Firmware version: 1.02

This function controls the use of a LED3 and LED4 on the RTCU device as status indicator for the file system.
When the status is enabled, LED 3 and LED 4 cannot be used from the application (no output variable can be configured to use the LEDs).

The use of LED3/4 has the following interpretation:

Off	SD-CARD is not present.
Green, Constant	SD-CARD is present.
Green, Blinking	SD-CARD is ejected and can be removed.
Orange	Read/write operation on the SD-CARD.

Note: on devices where LED3 and LED4 are not present, LED1 and LED2 are used instead.
Note: This function is not supported on NX32L.

Input:

enable : BOOL
TRUE: Use the LED for status.
FALSE: Do not use the LED for status.

Returns: INT

0 - Success.

Declaration:

```
FUNCTION fsStatusLEDEnable : INT;
VAR_INPUT
   enable : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

   fsStatusLEDEnable(enable:=ON);
BEGIN
   ...
```

```
END;
END_PROGRAM;
```

4.2.14.33 fsFileMove (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.21 / 1.00.00

This function moves a file to another directory. The destination must be on the same drive as the source.

Input:

src : STRING
Name of the file to move. Both absolute and relative paths are allowed.

dst : STRING
The destination path of the file. Both absolute and relative paths are allowed.

Returns: INT

0	- File is renamed.
-1	- Media not open.
-2	- Media is not formatted.
-3	- Path to file is invalid.
-4	- Name of file (new or old) is invalid.
-5	- File not found.
-6	- A file with this name already exist.
-12	- File is open.
-16	- Media not present.
-17	- Media communication error (card not supported).
-22	- Media is busy.
-23	- Media is write protected.
-24	- Unknown file system.

Declaration:

```
FUNCTION fsFileMove : INT;
VAR_INPUT
   src : STRING;
   dst : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc : INT;
END_VAR;

BEGIN
   ...
   // Move file
   rc := fsFileMove(src:="\prg1.log",dst:="\folder\prg1.log");
   IF rc = 0 THEN
      ...
   ELSE
      // Move failed
```

```
        ...
    END_IF;
    ...
END;

END_PROGRAM;
```

4.2.15 ftp: File Transfer Protocol

4.2.15.1 ftp: FTP Functions

FTP functions are used to send and receive files by using FTP (File Transfer Protocol).
FTP is an internet standard (RFC 959) for file transmission across IP networks.

The RTCU devices exclusively uses Passive (PASV) mode for file transfer.

Note:
On many FTP servers, the name of files and folders are case sensitive.
The handling of error situations like lost connection may vary from server to server.

Secure connection
For NX32L devices, support for FTPS (FTP secure) is available using TLS (TLS v1.2, v1.1 or 1.0).

To use secure connection:
1. Ensure the root certificate needed to verify the FTP server is present.
2. When calling ftpConnect set the 'security' to the required type of security. E.g. 'security := 1' (use FTPS if the server supports this).

Functions used for server communication:

- ftpOpen — Opens the FTP interface.
- ftpClose — Closes the FTP interface.
- ftpConnect — Connects to server.
- ftpConnected — Checks connection to server.
- ftpDisconnect — Disconnects from server.
- ftpCancel — Cancel last file transfer.
- ftpLastResponse — Gets last server response.
- ftpSendOpts — Send options to the server.

Functions operating on directories:

- ftpDirChange — Changes working directory.
- ftpDirCreate — Creates a directory.
- ftpDirCurrent — Gets working directory.
- ftpDirDelete — Deletes a directory.
- ftpDirRename — Renames a directory.
- ftpDirCatalogGet — Fetches catalog.
- ftpDirCatalog — Gets an entry from a fetched catalog.

Functions operating on files:

- ftpFileDelete — Deletes a file.
- ftpFileReceive — Receives a file.
- ftpFileRename — Renames a file.
- ftpFileSend — Sends a file.
- ftpFileSize — Gets the remote size of a file.

Receiving / sending a file over FTP

The following sequence is required to send or receive a file:

1.	ftpOpen	Initialize the FTP functions.
2.	ftpConnect	Connect to server.
3.	ftpFileSend / ftpFileReceive	Send/receive a file.
4.	ftpDisconnect	Disconnect from server.
5.	ftpClose	Finish working with FTP.

The implementation of FTP has the following limitations:

- Only passive FTP is allowed.
- An IP-network connection must be available.
- File system must be present and open for sending and receiving files.
- When fetching catalogs, only the first 64 files in 8.3 format are listed.

4.2.15.2 ftpOpen (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 2.10 / 1.00.00

This function will open the FTP interface. Calling this function is required before the FTP interface can be used.
Also see ftpClose for closing the FTP interface when finished with the operations.

This function must be called before any communication with the server by using FTP can occur. The actual connection has not been established when this function returns. The ftpConnect function must be used to establish the connection.

Prior to communicating with the server, the network interface connection must be established. For example by using gsmPower and gprsOpen for Mobile network.

Note:
Timeouts may occurs on action if they are interrupted by incoming or outgoing voice calls.

Input:

None.

Returns: INT

0	- Success.
- 1	- General error.
- 6	- FTP not supported on device.
-10	- Communication with server not possible.

Declaration:

```
FUNCTION ftpOpen : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   open      : BOOL;
END_VAR;

   gsmPower(power := ON);
   gprsOpen();
```

```
BEGIN
  open := (ftpOpen() = 0);
  ...
  ftpClose();
END;
END_PROGRAM;
```

4.2.15.3 ftpClose (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 2.10 / 1.00.00

This function will close the FTP interface. After the FTP interface is closed, it cannot be used until opened again.
Please also see ftpOpen.

When closing the FTP interface, all sessions created with ftpConnect will automatically be closed.

Input:
None.

Returns: INT
0 - Success.
- 5 - FTP not open, use ftpOpen to open interface.

Declaration:
```
FUNCTION ftpClose : INT;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   open      : BOOL;
END_VAR;

   gsmPower(power := ON);
   gprsOpen();

BEGIN
  open := (ftpOpen() = 0);
  ...
  ftpClose();
END;
END_PROGRAM;
```

4.2.15.4 ftpCancel (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 2.10 / 1.00.00

This function will notify ftpFileSend or ftpFileReceive functions to abort the current file transfer and return.

Note: how the server handles the abort request is individual and partial files may still exists.

Input:
ID : INT
The ID returned from ftpConnect.

Returns: INT
0 - Success.
- 1 - General error.
- 2 - Invalid ID.
- 5 - FTP not open, use ftpOpen to open interface.

Declaration:
```
FUNCTION ftpCancel : INT;
VAR_INPUT
   ID     : INT;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

VAR
   host   : STRING := "ftp.domain.com";
   usr    : STRING := "ftpuser";
   pwd    : STRING := "user pwd";

   id     : INT;
END_VAR;

THREAD_BLOCK tbSend;
   ftpOpen();
   ...
   id := ftpConnect(host := host, username := usr, password := pwd);
   ftpFileReceive(id := id, name := name, local := local);
   id := 0;
   ftpClose();
END_THREAD_BLOCK;

PROGRAM example;
VAR
   send   : tbSend;
END_VAR;

   gsmPower(power := ON);
   gprsOpen();
BEGIN
   send();
   ...
   IF id <> 0 THEN
      ftpCancel(id := id);
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.15.5 ftpConnect (Function)

```
Architecture:       X32 / NX32 / NX32L
Device support:     All
Firmware version:   2.10 / 1.00.00
```

This initiates a FTP connection to the specified IP address/port number. When this function returns, the connection has been established.
For each ftpConnect(), there must be a corresponding ftpDisconnect() even if the connection is closed from the server side (see example below).

It is possible to establish a maximum of 2 simultaneous FTP connections.

Note:
A connection error is returned if a successful login request cannot be obtained. This includes invalid IP or a non-FTP response from server.

Using secure connection
For NX32L devices, a secure TLS (TLS v1.2, v1.1 or 1.0) connection can be established, assuming that the matching X509 certificate is present.
There are 2 ways of implementing secure connections:
1. Explicit, where the secure connection handshake is performed after the server sends the welcome message.
2. Implicit, where the secure connection handshake is performed before the server sends the welcome message.
The second one is an older (non standard) method which requires a different port (the default is port 990).

With both methods, the secure connection is established before sending the User name and Password during login.

Input:

Host : STRING
IP address (in dotted format "aaa.bbb.ccc.ddd" or symbolic "domain.xyz".

Port : DINT (1..65535, default 21)
Port to connect to on server.

iface : SINT (default 1)
The network interface to use. 0 = Default network, 1 = Mobile network, 2 = LAN network, etc. (See Network)
Note: For backwards compatibility the Mobile network is used as default network.

Username : STRING (default "anonymous")
Username to use during login.

Password : STRING
Password to use if requested by server.
Note: password is only used if requested by server.

Security : SINT
The required type of security used by the connection.
0 - Plain text session.
1 - Plain text session or Explicit secure session.
2 - Explicit secure session.
3 - Implicit secure session.

Returns: INT

ID of new connected session.

- 1 - All session is in use, see ftpDisconnect to free.
- 3 - Error in parameter.
- 5 - FTP not open, use ftpOpen to open interface.
- 8 - Connecting to server failed.
- 9 - Login failed, server has not replied as expected to login.
- 10 - Communication with server not possible.
- 11 - The server require secure connection.
- 12 - Failed to establish secure connection.

Declaration:

```
FUNCTION ftpConnect : INT;
VAR_INPUT
   Host     : STRING;
   Username : STRING := "anonymous";
   Password : STRING;
   Port     : DINT   := 21;
   iface    : SINT   := 1;
   security : SINT   := 0;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   open  : BOOL;
   host  : STRING := "ftp.domain.com";
   usr   : STRING := "ftpuser";
   pwd   : STRING := "user pwd";

   id    : INT;
END_VAR;

   gsmPower(power := ON);
   gprsOpen();

BEGIN
  open := (ftpOpen() = 0);
  IF open THEN
     id := ftpConnect(host := host, username := usr, password := pwd);
     ...
     ftpDisconnect(id := id);
  END_IF;
  ftpClose();
END;
END_PROGRAM;
```

4.2.15.6 ftpConnected (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 2.10 / 1.00.00

This function will test whether a connection to the FTP server has been established.
The connection state information is based on an expected response from the FTP server and does therefore not reflect the socket connection establishment itself.

Input:
ID : INT
The ID returned from ftpConnect.

Returns: **BOOL**
True if connected to server, false if not.

Declaration:
```
FUNCTION ftpConnected : BOOL;
VAR_INPUT
   ID     : INT;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   open  : BOOL;
   host  : STRING := "ftp.domain.com";
   usr   : STRING := "ftpuser";
   pwd   : STRING := "user pwd";

   id    : INT;
END_VAR;

   gsmPower(power := ON);
   gprsOpen();

BEGIN
  open := (ftpOpen() = 0);
  IF open THEN
     id := ftpConnect(host := host, username := usr, password := pwd);
     IF ftpConnected(id := id) THEN
     .
     ...
     END_IF;
     ftpDisconnect(id := id);
  END_IF;
  ftpClose();
END;
END_PROGRAM;
```

4.2.15.7 ftpDisconnect (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 2.10 / 1.00.00

This function will disconnect from the FTP server. All references to the connection identification (ID) after this call is illegal.
The connection is closed by quitting the session - thus making sure that the user is correctly logged out.
Any file transfer in progress will be canceled.

For each ftpConnect(), there must be a corresponding call to ftpDisconnect(). This is also the case when the connection is closed from the FTP server side (see example below).

Note:

Disconnection with no server communication capability will result in an unclosed session on the server.

Input:
ID : INT
The ID returned from ftpConnect.

Returns: INT
0 - Success.
- 1 - Failed to close connection.
- 2 - Invalid ID.
- 5 - FTP not open, use ftpOpen to open interface.

Declaration:
```
FUNCTION ftpDisconnect : INT;
VAR_INPUT
   ID     : INT;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   open  : BOOL;
   host  : STRING := "ftp.domain.com";
   usr   : STRING := "ftpuser";
   pwd   : STRING := "user pwd";

   id    : INT;
END_VAR;

   gsmPower(power := ON);
   gprsOpen();

BEGIN
  open := (ftpOpen() = 0);
  IF open THEN
     id := ftpConnect(host := host, username := usr, password := pwd);
     .
     ...
     ftpDisconnect(id := id);
  END_IF;
  ftpClose();
END;
END_PROGRAM;
```

4.2.15.8 ftpDirChange (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 2.10 / 1.00.00

This function changes the working directory on the FTP server.

Input:
ID : INT
The ID returned from ftpConnect.

Name : STRING
Name of the new working directory. Both absolute and relative paths are allowed.

Returns: INT

0 - Success.
- 1 - Failed to change working directory, see ftpLastResponse.
- 2 - Invalid ID.
- 3 - Invalid name string.
- 4 - Session is busy with another operation.
- 5 - FTP not open, use ftpOpen to open interface.
- 10 - Communication with server not possible.

Declaration:

```
FUNCTION ftpDirChange : INT;
VAR_INPUT
   ID   : INT;
   Name : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   open  : BOOL;
   host  : STRING := "ftp.domain.com";
   usr   : STRING := "ftpuser";
   pwd   : STRING := "user pwd";

   dir   : STRING := "test";
   id    : INT;
END_VAR;

   gsmPower(power := ON);
   gprsOpen();

BEGIN
  open := (ftpOpen() = 0);
  IF open THEN
     id := ftpConnect(host := host, username := usr, password := pwd);
     ftpDirChange(id := id, name := dir);
     ftpDisconnect(id := id);
  END_IF;
  ftpClose();
END;
END_PROGRAM;
```

4.2.15.9 ftpDirCreate (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 2.10 / 1.00.00

This function creates a directory on the FTP server.

The ftpDirCreate function does not work recursively and only tries to create the specified directory. If any of the directories in the path (if absolute) do not exist, the function may fail depending on how the server handles the request.

Input:

ID : INT
The ID returned from ftpConnect.

Name : STRING
Name of the directory to create. Both absolute and relative paths are allowed.

Returns: INT

0 - Success.
- 1 - Failed to create directory, see ftpLastResponse.
- 2 - Invalid ID.
- 4 - Session is busy with another operation.
- 5 - FTP not open, use ftpOpen to open interface.
- 10 - Communication with server not possible.

Declaration:

```
FUNCTION ftpDirCreate : INT;
VAR_INPUT
   ID   : INT;
   Name : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   open  : BOOL;
   host  : STRING := "ftp.domain.com";
   usr   : STRING := "ftpuser";
   pwd   : STRING := "user pwd";

   dir   : STRING := "test";
   id    : INT;
END_VAR;

   gsmPower(power := ON);
   gprsOpen();

BEGIN
  open := (ftpOpen() = 0);
  IF open THEN
     id := ftpConnect(host := host, username := usr, password := pwd);
     ftpDirCreate(id := id, name := dir);
     ftpDisconnect(id := id);
  END_IF;
  ftpClose();
END;
END_PROGRAM;
```

4.2.15.10 ftpDirCurrent (Function)

```
Architecture:       X32 / NX32 / NX32L
Device support:     All
Firmware version:   2.10 / 1.00.00
```

This function returns the current working directory on the FTP server.

Input:
ID : INT
The ID returned from ftpConnect.

Returns: STRING
The current directory or empty if an error has occurred (see ftpLastResponse for possible error information).

Declaration:
```
FUNCTION ftpDirCurrent : STRING;
VAR_INPUT
   ID    : INT;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   open  : BOOL;
   host  : STRING := "ftp.domain.com";
   usr   : STRING := "ftpuser";
   pwd   : STRING := "user pwd";

   id    : INT;
END_VAR;

   gsmPower(power := ON);
   gprsOpen();

BEGIN
  open := (ftpOpen() = 0);
  IF open THEN
     id := ftpConnect(host := host, username := usr, password := pwd);
     DebugMsg(message := "current = '" + ftpDirCurrent(id := id) + "'");
     ftpDisconnect(id := id);
  END_IF;
  ftpClose();
END;
END_PROGRAM;
```

4.2.15.11 ftpDirDelete (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 2.10 / 1.00.00

This function deletes a directory on the FTP server.

Input:
ID : INT
The ID returned from ftpConnect.

Name : STRING
Name of the directory to delete. Both absolute and relative paths are allowed.

Returns: INT
0 - Success.

- 1 - Delete failed, see ftpLastResponse.
- 2 - Invalid ID.
- 3 - Invalid or missing name.
- 4 - Session is busy with another operation.
- 5 - FTP not open, use ftpOpen to open interface.
- 10 - Communication with server not possible.

Declaration:

```
FUNCTION ftpDirDelete : INT;
VAR_INPUT
   ID   : INT;
   Name : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   open  : BOOL;
   host  : STRING := "ftp.domain.com";
   usr   : STRING := "ftpuser";
   pwd   : STRING := "user pwd";

   dir   : STRING := "test";
   id    : INT;
END_VAR;

   gsmPower(power := ON);
   gprsOpen();

BEGIN
  open := (ftpOpen() = 0);
  IF open THEN
     id := ftpConnect(host := host, username := usr, password := pwd);
     ftpDirDelete(id := id, name := dir);
     ftpDisconnect(id := id);
  END_IF;
  ftpClose();
END;
END_PROGRAM;
```

4.2.15.12 ftpDirRename (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 2.10 / 1.00.00

This function renames a directory on the FTP server.

Note:
If no path information is included in the new name, the directory remains at the same location. Otherwise the directory will be moved to the new location with the new name which can be the same as the old one.

Input:

ID : INT
The ID returned from ftpConnect.

Old_Name : STRING
Name of the directory to rename. Both absolute and relative paths are allowed.

New_Name : STRING
The new name of the directory.

Returns: INT

```
   0   - Success.
 - 1   - Rename failed, see ftpLastResponse.
 - 2   - Invalid ID.
 - 3   - Invalid or missing old or new name.
 - 4   - Session is busy with another operation.
 - 5   - FTP not open, use ftpOpen to open interface.
 - 10  - Communication with server not possible.
```

Declaration:

```
FUNCTION ftpDirRename : INT;
VAR_INPUT
   ID       : INT;
   Old_Name : STRING;
   New_Name : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   open  : BOOL;
   host  : STRING := "ftp.domain.com";
   usr   : STRING := "ftpuser";
   pwd   : STRING := "user pwd";

   dir   : STRING := "test";

   id    : INT;
END_VAR;

   gsmPower(power := ON);
   gprsOpen();

BEGIN
  open := (ftpOpen() = 0);
  IF open THEN
     id := ftpConnect(host := host, username := usr, password := pwd);
     ftpDirRename(id := id, old_name := dir, new_name := dir + "_o");
     ftpDisconnect(id := id);
  END_IF;
  ftpClose();
END;
END_PROGRAM;
```

4.2.15.13 ftpDirCatalogGet (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 2.10 / 1.00.00

This function will fetch the contents of the current directory on the FTP server

To read the actual entries in the list, use ftpDirCatalog.

The list will only contain files or sub-directories, and the order is as received by the server.

Note:
If "Name" is used, this is sent unchecked to the server as an option to the FTP command and can on most servers be used to filter the names listed e.g. "*.jpg" or "/log/*.log".
Only the first 64 names that match the 8.3 file format are fetched if using this function.

Input:

ID : INT
The ID returned from ftpConnect.

Dir : BOOL (default FALSE)
List directories or files.

Name : STRING (Optional)
If empty, all names in current directory are requested.

Returns: INT

Number of names in catalog.
- 1 - Failed to fetch list of files, see ftpLastResponse.
- 2 - Invalid ID.
- 4 - Session is busy with another operation.
- 5 - FTP not open, use ftpOpen to open interface.
- 10 - Communication with server not possible.

Declaration:

```
FUNCTION ftpDirCatalogGet : INT;
VAR_INPUT
   ID     : INT;
   Name   : STRING := "";
   Dir    : BOOL   := FALSE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   open  : BOOL;
   host  : STRING := "ftp.domain.com";
   usr   : STRING := "ftpuser";
   pwd   : STRING := "user pwd";

   id    : INT;

   count : INT;
   rc, i : INT;
   name  : STRING;
   size  : DINT;
END_VAR;

   gsmPower(power := ON);
   gprsOpen();

BEGIN
  open := (ftpOpen() = 0);
  IF open THEN
```

```
      id := ftpConnect(host := host, username := usr, password := pwd);
      count := ftpDirCatalogGet(id := id);

      FOR i := 1 TO count DO
        rc := ftpDirCatalog(id := id, index := i, name := name, size := size);
        CASE rc OF
          1 : DebugFmt(message := "File name = '" + name +"' size = \4", v4 :=
size);
          2 : DebugFmt(message := "Dir  name = '" + name +"' size = \4", v4 :=
size);
        END_CASE;
      END_FOR;

      ftpDisconnect(id := id);
  END_IF;
  ftpClose();
END;
END_PROGRAM;
```

4.2.15.14 ftpDirCatalog (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 2.10 / 1.00.00

This function will return an entry from the catalog fetched with ftpDirCatalogGet.
Names in the list are not sorted and are listed in the order received from the server.

Note:
The size of a directory is always 0.

Input:

ID : INT
The ID returned from ftpConnect.

Index : INT (1..64)
Index to be returned in the range of 1 to the value returned by ftpDirCatalogGet.

Output:

Name : STRING
File name.

Size : DINT
File size.

Returns: INT

2 - Index is a directory.
1 - Index is a file.

- 1 - Catalog has not been fetched, see ftpDirCatalogGet.
- 2 - Invalid ID.
- 3 - Invalid index.
- 4 - Session is busy with another operation.
- 5 - FTP not open, use ftpOpen to open interface.
- 10 - Communication with server not possible

Declaration:

```
FUNCTION ftpDirCatalog : INT;
VAR_INPUT
```

```
   ID     : INT;
   Index  : INT;
   Name   : ACCESS STRING;
   Size   : ACCESS DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   open  : BOOL;
   host  : STRING := "ftp.domain.com";
   usr   : STRING := "ftpuser";
   pwd   : STRING := "user pwd";

   id    : INT;

   count : INT;
   rc, i : INT;
   name  : STRING;
   size  : DINT;
END_VAR;

   gsmPower(power := ON);
   gprsOpen();

BEGIN
  open := (ftpOpen() = 0);
  IF open THEN
     id := ftpConnect(host := host, username := usr, password := pwd);
     count := ftpDirCatalogGet(id := id);

     FOR i := 1 TO count DO
       rc := ftpDirCatalog(id := id, index := i, name := name, size := size);
       CASE rc OF
         1 : DebugFmt(message := "File name = '" + name +"' size = \4", v4 :=
size);
         2 : DebugFmt(message := "Dir  name = '" + name +"' size = \4", v4 :=
size);
       END_CASE;
     END_FOR;

     ftpDisconnect(id := id);
  END_IF;
  ftpClose();
END;
END_PROGRAM;
```

4.2.15.15 ftpFileDelete (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 2.10 / 1.00.00

This function deletes a file on the FTP server.

Input:

ID : INT
The ID returned from ftpConnect.

Name : STRING
Name of the file to delete. Both absolute and relative paths are allowed.

Returns: INT

0 - Success.
-1 - Delete failed, see ftpLastResponse.
-2 - Invalid ID.
-3 - Invalid name
-4 - Session is busy with another operation.
-5 - FTP not open, use ftpOpen to open interface.
-10 - Communication with server not possible.

Declaration:

```
FUNCTION ftpFileDelete : INT;
VAR_INPUT
   ID   : INT;
   Name : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   open  : BOOL;
   host  : STRING := "ftp.domain.com";
   usr   : STRING := "ftpuser";
   pwd   : STRING := "user pwd";

   name  : STRING := "test.txt";
   id    : INT;
END_VAR;

   gsmPower(power := ON);
   gprsOpen();

BEGIN
  open := (ftpOpen() = 0);
  IF open THEN
     id := ftpConnect(host := host, username := usr, password := pwd);
     ftpFileDelete(id := id, name := name);
     ftpDisconnect(id := id);
  END_IF;
  ftpClose();
END;
END_PROGRAM;
```

4.2.15.16 ftpFileReceive (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 2.10 / 1.00.00

This function transfers a file from the FTP server to the device.
The file system must already be open with fsMediaOpen or the receive will fail.

Note:

The length of the remote name is restricted by the server but the local name must be in 8.3 format.

Note:
If the file already exists, it will be replaced without warnings.

Input:

ID : INT
The ID returned from ftpConnect.

Name : STRING
Name of the file to receive. Both absolute and relative paths are allowed.

Local : STRING (Optional)
Alternative name to save file as. Both absolute and relative paths are allowed.

Returns: INT

```
   0   - Success.
 - 1   - Failed to receive file, see ftpLastResponse.
 - 2   - Invalid ID.
 - 3   - Invalid local or remote name, e.g. missing remote or local not in 8.3 format.
 - 4   - Session is busy with another operation.
 - 5   - FTP not open, use ftpOpen to open interface.
 - 10  - Communication with server not possible.
```

Declaration:

```
FUNCTION ftpFileReceive : INT;
VAR_INPUT
   ID    : INT;
   Name  : STRING;
   Local : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   open  : BOOL;
   host  : STRING := "ftp.domain.com";
   usr   : STRING := "ftpuser";
   pwd   : STRING := "user pwd";

   name   : STRING := "test.txt";

   id    : INT;
END_VAR;

   gsmPower(power := ON);
   gprsOpen();
   fsMediaOpen(media := 0);

BEGIN
  open := (ftpOpen() = 0);
  IF open THEN
     id := ftpConnect(host := host, username := usr, password := pwd);
     ftpFileReceive(id := id, name := name, local := name);
     ftpDisconnect(id := id);
  END_IF;
  ftpClose();
```

```
END;
END_PROGRAM;
```

4.2.15.17 ftpFileRename (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 2.10 / 1.00.00

This function renames a file on the FTP server.

Note:
If no path information is included in the new name, the file remains at the same location. Otherwise the file will be moved to the new location with the new name which may be the same as the old.

Input:

ID : INT
The ID returned from ftpConnect.

Old_Name : STRING
Name of the file to rename. Both absolute and relative paths are allowed.

New_Name : STRING
The new location and name of the file.

Returns: INT

- 0 - Success.
- 1 - Rename failed, see ftpLastResponse.
- 2 - Invalid ID.
- 3 - Invalid old or new name.
- 4 - Session is busy with another operation.
- 5 - FTP not open, use ftpOpen to open interface.
- 10 - Communication with server not possible.

Declaration:

```
FUNCTION ftpFileRename : INT;
VAR_INPUT
   ID       : INT;
   Old_Name : STRING;
   New_Name : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   open  : BOOL;
   host  : STRING := "ftp.domain.com";
   usr   : STRING := "ftpuser";
   pwd   : STRING := "user pwd";

   name   : STRING := "test.txt";

   id    : INT;
END_VAR;
```

```
   gsmPower(power := ON);
   gprsOpen();

BEGIN
  open := (ftpOpen() = 0);
  IF open THEN
     id := ftpConnect(host := host, username := usr, password := pwd);
     ftpFileRename(id := id, old_name := name, new_name := "o_" + name);
     ftpDisconnect(id := id);
  END_IF;
  ftpClose();
END;
END_PROGRAM;
```

4.2.15.18 ftpFileSend (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 2.10 / 1.00.00

This function transfers a file from the device to the FTP server.
The file system must already be open with fsMediaOpen or the send will fail.

Note:
The length of the remote file name is restricted by the server.

Note:
If the file already exists and the user is allowed to overwrite it, the file will be replaced without warnings.

Input:

ID : INT
The ID returned from ftpConnect.

Name : STRING
Name to save the file as. Both absolute and relative paths are allowed.

Local : STRING (Optional)
Alternative local name of file to send if not the same name (without any path).
Both absolute and relative paths are allowed.

Returns: INT

0	- Success.
- 1	- Failed to send, see ftpLastResponse.
- 2	- Invalid ID.
- 3	- Invalid name, or it could not be used to generate a local name.
- 4	- Session is busy with another operation.
- 5	- FTP not open, use ftpOpen to open interface.
- 10	- Communication with server not possible

Declaration:

```
FUNCTION ftpFileSend : INT;
VAR_INPUT
   ID    : INT;
   Name  : STRING;
   Local : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   open  : BOOL;
   host  : STRING := "ftp.domain.com";
   usr   : STRING := "ftpuser";
   pwd   : STRING := "user pwd";

   name   : STRING := "test.txt";

   id    : INT;
END_VAR;

   gsmPower(power := ON);
   gprsOpen();
   fsMediaOpen(media := 0);

BEGIN
  open := (ftpOpen() = 0);
  IF open THEN
     id := ftpConnect(host := host, username := usr, password := pwd);
     ftpFileSend(id := id, name := name, local := name);
     ftpDisconnect(id := id);
  END_IF;
  ftpClose();
END;
END_PROGRAM;
```

4.2.15.19 ftpFileSize (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 2.10 / 1.00.00

This function will return the size of the file specified on the FTP server.

Note: this function may not be supported on some FTP servers.

Input:

ID : INT
The ID returned from ftpConnect.

Name : STRING
Name of the file to receive the size of. Both absolute and relative paths are allowed.

Returns: INT

The size of the file in bytes or the following error codes:
- 1 - Failed to get size, see ftpLastResponse.
- 2 - Invalid ID.
- 3 - Missing or invalid name.
- 4 - Session is busy with another operation.
- 5 - FTP not open, use ftpOpen to open interface.
- 10 - Communication with server not possible

Declaration:

```
FUNCTION ftpFileSize : DINT;
VAR_INPUT
```

```
   ID   : INT;
   Name : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   open  : BOOL;
   host  : STRING := "ftp.domain.com";
   usr   : STRING := "ftpuser";
   pwd   : STRING := "user pwd";

   name  : STRING := "test.txt";
   id    : INT;
   size  : DINT;
END_VAR;

   gsmPower(power := ON);
   gprsOpen();

BEGIN
  open := (ftpOpen() = 0);
  IF open THEN
     id := ftpConnect(host := host, username := usr, password := pwd);
     size := ftpFileSize(id := id, name := name);
     DebugFmt(message := "Size of " + name + " is \4", v4 := size);
     ftpDisconnect(id := id);
  END_IF;
  ftpClose();
END;
END_PROGRAM;
```

4.2.15.20 ftpSendOpts (Function)

Architecture: NX32 / NX32L
Device support: All
Firmware version: 5.18 / 2.12.00

This function will send an OPTS (options) command to the FTP server.
This can be used configure the server behavior, to e.g. enable or disable UTF-8 support.

Note: this function may not be supported on some FTP servers.
Note: Some options may prevent further communication on the current session.

Input:

ID : INT
The ID returned from ftpConnect.

opts : STRING
The options to send to the server.

Returns: INT

- 0 - Success.
- -1 - Failed to send option, see ftpLastResponse.
- -2 - Invalid ID.
- -3 - Missing or invalid opts.
- -4 - Session is busy with another operation.

- 5 - FTP not open, use ftpOpen to open interface.
- 10 - Communication with server not possible

Declaration:

```
FUNCTION ftpSendOpts : INT;
VAR_INPUT
   ID   : INT;
   opts : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   open  : BOOL;
   host  : STRING := "ftp.domain.com";
   usr   : STRING := "ftpuser";
   pwd   : STRING := "user pwd";

   name  : STRING := "$CF$86.txt"; // "phi".txt
   id    : INT;
   size  : DINT;
END_VAR;

   gsmPower(power := ON);
   gprsOpen();

BEGIN
  open := (ftpOpen() = 0);
  IF open THEN
     id := ftpConnect(host := host, username := usr, password := pwd);
     rc := ftpSendOpts(id := id, opts := "UTF8 ON");

     size := ftpFileSize(id := id, name := name);
     DebugFmt(message := "Size of " + name + " is \4", v4 := size);
     ftpDisconnect(id := id);
  END_IF;
  ftpClose();
END;
END_PROGRAM;
```

4.2.15.21 ftpLastResponse (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 2.10 / 1.00.00

This function will read the last reply from the FTP server.
Using this function may help diagnose general problems like user access rights and missing files.
Only the last line in the response from the server will be returned.

Input:

ID : INT
The ID returned from ftpConnect.

Output:

Response : STRING
The last reply from server.

Returns: INT

0 - Success.
- 1 - No response exists.
- 2 - Invalid ID.
- 4 - Session is busy with another operation.
- 5 - FTP not open, use ftpOpen to open interface.

Declaration:

```
FUNCTION ftpLastResponse : INT;
VAR_INPUT
   ID       : INT;
   Response : ACCESS STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   open  : BOOL;
   host  : STRING := "ftp.domain.com";
   usr   : STRING := "ftpuser";
   pwd   : STRING := "user pwd";

   str   : STRING := "";
   id    : INT;
END_VAR;

   gsmPower(power := ON);
   gprsOpen();

BEGIN
  open := (ftpOpen() = 0);
  IF open THEN
     id := ftpConnect(host := host, username := usr, password := pwd);
     IF id < 0 THEN
        ftpLastResponse(id := id, response := str);
        DebugMsg(message := "Server reply : " + str);
     END_IF;
     ftpDisconnect(id := id);
  END_IF;
  ftpClose();
END;
END_PROGRAM;
```

4.2.16 gnss: GNSS Receiver

4.2.16.1 GNSS: GNSS Receiver

The GNSS functions allow the program to interact with a GNSS module that supports multiple positioning systems, such as GPS, GLONASS, and GALILEO. The GNSS functionality is not available on all RTCU devices, so please consult the technical documentation for the actual RTCU device in question.

The GNSS functions are to be used together with the GPS functions, where the GNSS functions handle the functionality for multiple positioning systems, while the GPS functions implicitly use all positioning systems.

- gnssFix — Returns detailed information from the GNSS receiver.
- gnssSatellitesInView — Accesses information about the satellites in view
- gnssGetProfile — Returns information about GPS/GNSS features that are available on the RTCU device.
- gnssEnableType — Enables/disables the individual positioning systems.
- gnssGetEnabledSystems — Read a bit mask of enabled positioning systems.
- gnssDynamicModelSet — Configure the dynamic model used by the GNSS module.

Note: all values returned by the GNSS functions are in the WGS-84 datum.

To improve the performance in areas with poor or no signal conditions, some GNSS modules have support for Dead Reckoning using the following functions:

- gnssDRStatus — Return the status for Dead Reckoning.
- gnssDREnable — Enable or disable Dead Reckoning.
- gnssDRMountSetup — Configure mounting vectors for the GNSS module.
- gnssDRAlignSetup — Configure the alignment(rotation) of the GNSS module.
- gnssDRWheelTickSetup — Configure support for Wheel Tick for using sensors in the vehicle to improve the performance.
- gnssDRWheelTickPushTick — Send Wheel Tick data to the GNSS module.
- gnssDRWheelTickPushSpeed — Send vehicle speed to the GNSS module.

Dead Reckoning

Dead Reckoning allows for improved navigation performance in places with GNSS-denied conditions as well as during short GNSS outages.
It is based on Sensor Fusion Dead Reckoning (SFDR) technology, which integrates an Inertial Navigation System (INS) with GNSS measurements. The INS computes position, velocity and attitude changes and can, once initialized, provide accurate navigation information.

Two versions of the INS are used, one including WheelTick and one without WheelTick. These are traditionally called Automotive Dead Reckoning (ADR) and Untethered Dead Reckoning (UDR) respectively.

The GNSS uses some references to calculate the Dead Reckoning position:

The Vehicle Reference point (VRP) is the nominal point at which some motion constraints inherent to ground based vehicles are applied.
For automotive platforms, the VRP is defined as the center of the vehicle rear axle.

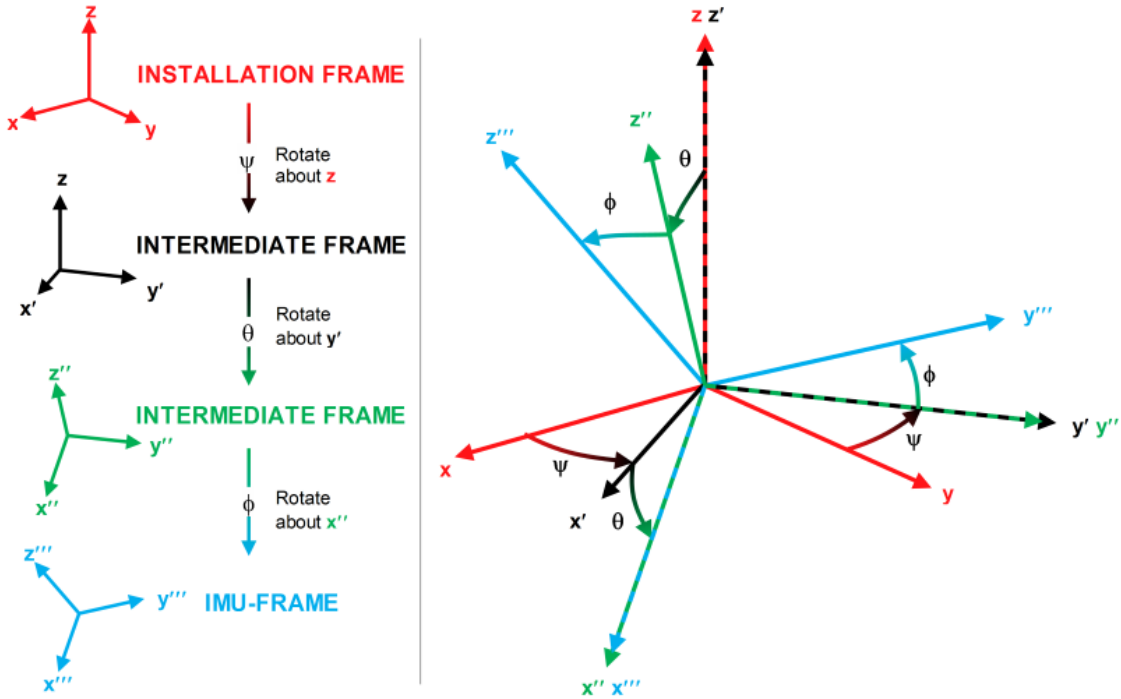
For bike platforms, the VRP is defined as the point where the wheel touches the ground. If the RTCU device is mounted on the handle bar, the VRP will be where the front wheel touches the ground, and if the RTCU device is mounted on the bike frame the VRP will be where the back wheel touches the ground.

The Device Frame is a right-handed 3D Cartesian frame connected to the RTCU device.
The x-axis points towards the back of the device (where the connectors and antenna connectors are present), the y-axis points towards the left of the device and the z-axis points up.
It is centered on the RTCU device.

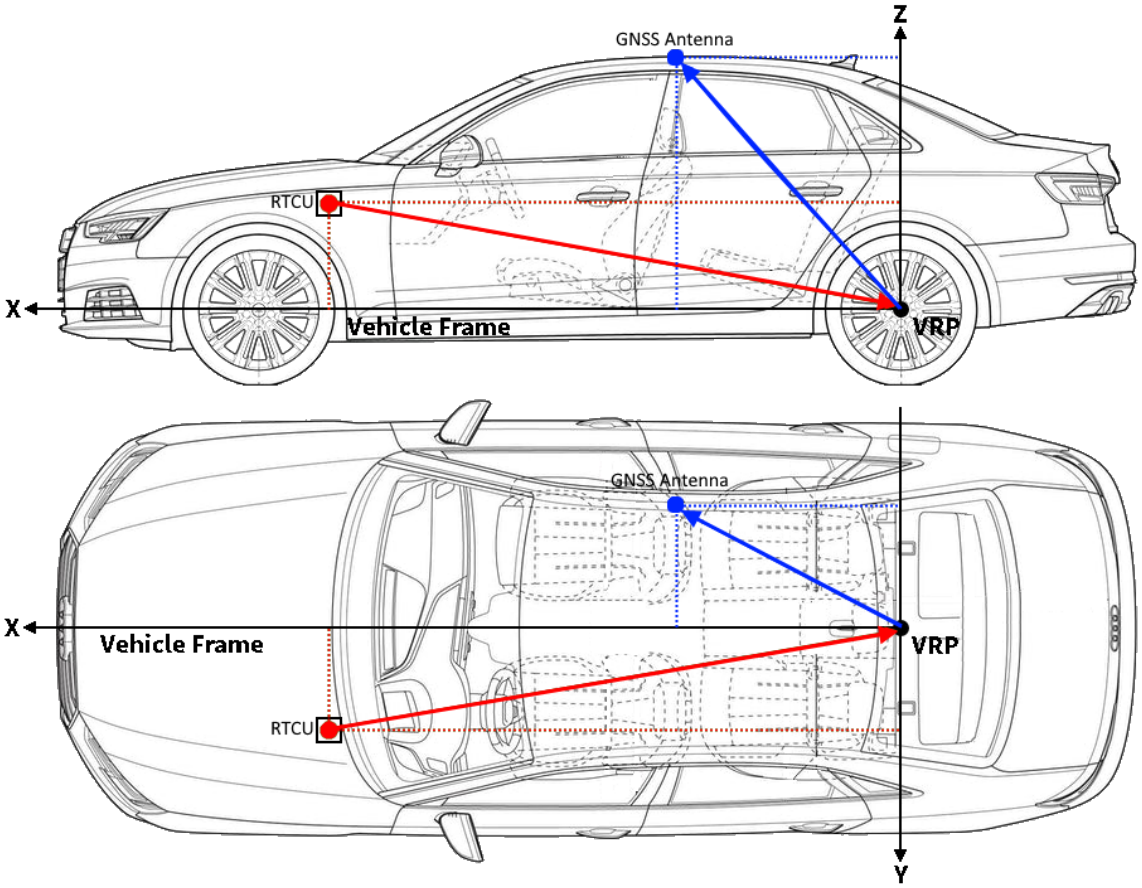
The Vehicle Frame is a right-handed 3D Cartesian frame connected to the vehicle.
The x-axis points towards the front of the vehicle, the y-axis points towards the left of the vehicle and the z-axis completes the right-handed reference system by pointing up. It is centered on the RTCU device.

The alignment between the Device Frame and the Vehicle Frame can degrade the positioning solution (if small, typically a few degrees) or even prevent Dead Reckoning (if large, typically tens of degrees).
Therefore, it is important to correctly configure the alignment.
Normally the alignment is estimated during a dedicated initialization drive through an automatic alignment procedure.
For bike platforms, the automatic alignment procedure is not possible and the alignment must be set manually..

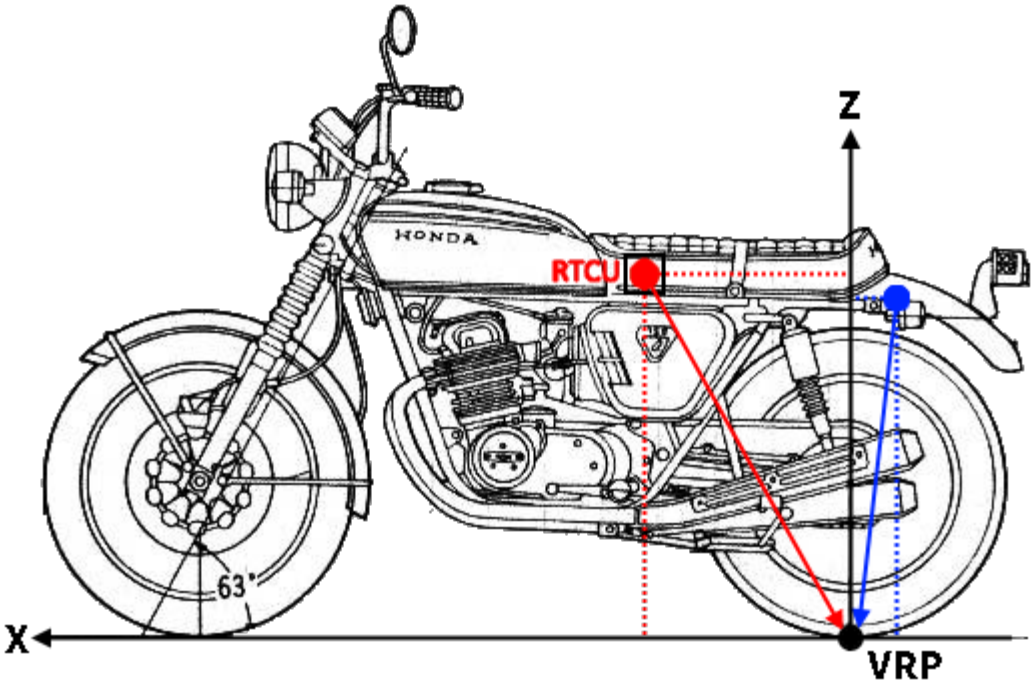
The alignment are the Euler angles required to rotate the Vehicle Frame to the Device Frame.
The yaw rotation should be performed first, then the pitch and finally the roll. At each stage, the rotation is around the appropriate axis of the transformed Vehicle Frame, meaning that the order of the rotation sequence is important (see figure below).



The Vehicle Frame is also used when configuring the RTCU device mount relative to the VRP and the GNSS Antenna, as shown in the illustration below:



and on bike platforms:



4.2.16.2 gnssFix (Functionblock)

```
Architecture:       NX32 / NX32L
Device support:     MX2 turbo/encore/warp, CX1 warp(-c) mk2, NX-200,
                    NX-400, NX-900, LX2, LX5
Firmware version:   4.00 / 1.40.00
```

The gnssFix function block is used to query the current position, time, and detailed satellite information from the internal GNSS module.

Make sure to call gpsPower or gpsPowerLP before calling gnssFix, to power on the GNSS module.

SBAS Support:
Please see gpsSetSBAS.
When the "mode" parameter that is returned from gnssFix is 4, it indicates that the fix is not only a 3D fix but has also been SBAS-corrected.

Note regarding HDOP, VDOP and PDOP:
The DOP values should be as low as possible. They describe the geometry of the satellites used. Generally a DOP value should not be higher than 6 as that will affect the accuracy negative.
Normally a DOP value will be between 2 and 3 when you have clear skies. In a city with houses screening the sky, the DOP value will be higher and accuracy will be affected.
One method to increase the accuracy of the used GNSS positions is not to use fixes with an HDOP value higher than, for example, 300.

All values returned by gnssFix() are in the WGS-84 datum.

Input:

None.

Output:

mode : SINT;
0 = No info available (GNSS module not powered on?).
1 = Fix unavailable.
2 = 2D position fix.
3 = 3D position fix.
4 = 3D position fix with SBAS-correction (this requires that SBAS has been enabled, see gpsSetSBAS).
5 = Dead Reckoning fix.

linsec : DINT
Linsec for the timestamp received from the GNSS receiver.
Also see clockLinsecToTime().

millisecond : INT
Millisecond (0..999).

latitude : DINT
Latitude expressed in a DINT.
Negative is south (ddmm.mmmm (multiplied by 10000)).

longitude : DINT
Longitude expressed in a DINT.
Negative is west (dddmm.mmmm (multiplied by 10000)).

speed : DINT
Speed over ground in meters/hour.

course : DINT
Course over ground. In xxx.xx degrees (multiplied by 100) (0=north, 90=east, 225=southwest).

height : INT
Height in meters over mean sea level.

PDOP : INT
Position dilution of precision (multiplied by 100).

HDOP : INT
Horizontal dilution of precision (multiplied by 100).

VDOP : INT
Vertical dilution of precision (multiplied by 100).

inview : SINT
Number of satellites in view. This is the number of satellites that are above the horizon.
If gpsPowerLP is used, this will be 0.

used : SINT
Number of satellites used for the solution.

Declaration:

```
FUNCTION_BLOCK gnssFix;
VAR_OUTPUT
   mode         : SINT;    | 0=No info available, 1=Fix not available, 2=2D
position fix, 3=3D position fix, 4=3D pos. with SBAS
   linsec       : DINT;    | Linsec of time-stamp
   millisecond  : INT;     | millisecond (0..999)
   latitude     : DINT;    | Negative is South (ddmm.mmmm (multiplied by
10000))
   longitude    : DINT;    | Negative is West (dddmm.mmmm (multiplied by
10000))
   speed        : DINT;    | Speed over ground in meters/hour
   course       : DINT;    | Course over ground. In xxx.xx degrees (multiplied
by 100)
   height       : INT;     | Height in meters over Mean Sea Level.
   PDOP         : INT;     | Position dilution of precision (PDOP) (multipied
by 100).
   HDOP         : INT;     | Horizontal dilution of precision (HDOP)
(multipied by 100).
   VDOP         : INT;     | Vertical dilution of precision (VDOP) (multipied
by 100).
   inview       : SINT;    | Number of satellites in view
   used         : SINT;    | Number of satellites used in solution
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   gnss  : gnssFix;
   clock : clockLinsecToTime;
   index : SINT;
```

```
   str   : STRING;
END_VAR;

gpsPower(power := ON);
BEGIN
   gnss();
   IF gnss.mode > 1 THEN
      clock(Linsec := gnss.linsec);
      DebugFmt(message:="Mode=\1", v1:=gnss.mode);
      DebugFmt(message:="   Linsec=\4", v4:=gnss.linsec);
      DebugFmt(message:="   Time=\1:\2:\3", v1:=clock.hour, v2:=clock.minute,
v3:=clock.second);
      DebugFmt(message:="   Date=\1:\2:\3", v1:=clock.year, v2:=clock.month,
v3:=clock.day);
      DebugFmt(message:="   Lat: latitude=\4", v4:=gnss.latitude);
      DebugFmt(message:="   Lon: longitude=\4", v4:=gnss.longitude);
      DebugFmt(message:="   Speed=\4", v4:=gnss.speed);
      DebugFmt(message:="   Course=\4", v4:=gnss.course);
      DebugFmt(message:="   Height=\1", v1:=gnss.height);
      DebugFmt(message:="   PDOP=\1", v1:=gnss.PDOP);
      DebugFmt(message:="   HDOP=\1", v1:=gnss.HDOP);
      DebugFmt(message:="   VDOP=\1", v1:=gnss.VDOP);
      DebugFmt(message:="   In view=\1, Used=\2", v1:=gnss.inview,
v2:=gnss.used);

   END_IF;
END;
END_PROGRAM;
```

4.2.16.3 gnssGetProfile (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 4.70 / 1.30.00

---

gnssGetProfile returns information about the the GPS/GNSS facilities available on the RTCU device.

Input:

index : SINT
This identifies which facility is being requested.
Possible values are:

_GNSS_SYS_GPS - GPS navigation system is available.
_GNSS_SYS_GLONASS - GLONASS navigation system is available.
_GNSS_SYS_GALILEO - GALILEO navigation system is available.
_GNSS_SYS_BEIDOU - BeiDou navigation system is available.
_GNSS_FEAT_ANT - Antenna detection is available.
_GNSS_FEAT_DR - Dead Reckoning is available.
_GNSS_FEAT_PPP - Precise Point Positioning is available.

Returns: **DINT**
The requested parameter
In case of TRUE/FALSE conditions, 0 (zero) will indicate FALSE and 1 (one) will indicate TRUE.

Declaration:

```
FUNCTION gnssGetProfile : DINT;
VAR_INPUT
   index : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   ant_state : INT;
   ant_old   : INT;
END_VAR;

   gpsPower(power := ON);

BEGIN
   // Only check the antenna status if supported
   IF gnssGetProfile(index := _GNSS_FEAT_ANT) > 0 THEN
      ant_state := gpsGetAntennaStatus();
      IF ant_state <> ant_old THEN
         CASE ant_state OF
            0: DebugMsg(message := "Antenna : Power OFF");
            1: DebugMsg(message := "Antenna : Present");
            2: DebugMsg(message := "Antenna : Short-circuited");
            3: DebugMsg(message := "Antenna : Missing");
         END_CASE;
      END_IF;
      ant_old := ant_state;
   END_IF;
END;
END_PROGRAM;
```

4.2.16.4 gnssSatellitesInView (Function)

Architecture: **NX32 / NX32L**
Device support: **MX2 turbo/encore/warp, CX1 warp(-c) mk2, NX-200, NX-400, NX-900, LX2, LX5**
Firmware version: **4.00 / 1.40.00**

Retrieves information about the satellites in view.
This can be used to help with determining the local signal conditions and position accuracy.

Input:

type : SINT (*Default: 0*)
0 = GPS
1 = GLONASS
2 = GALILEO.
3 = BeiDou.

tracked : BOOL *Default TRUE*
Retrieve satellites that are being tracked.

untracked : BOOL *Default FALSE*
Retrieve untracked satellites.

gnss : BOOL *Default TRUE*
Retrieve normal positioning satellites.

sbas : BOOL *Default FALSE*
Retrieve SBAS satellites.

Output:

satellites : gnssSatellites
STRUCT_BLOCK with space for information about 30 satellites:

id[n]: SINT
Satellite ID.

elevation[n]: SINT
Elevation angle for satellite n.

azimuth[n]: INT
Azimuth angle for satellite n.

SNR[n]: SINT
Signal to noise ratio for satellite n. If the satellite is untracked, the SNR will be 120.

Returns: INT

Number of found satellites matching the filter
-1 - Not powered or not supported
-2 - Invalid parameter

Declaration:

```
STRUCT_BLOCK ALIGN gnssSatellites;
  id        : ARRAY [1..30] OF SINT;
  elevation : ARRAY [1..30] OF SINT;
  azimuth   : ARRAY [1..30] OF INT;
  SNR       : ARRAY [1..30] OF SINT;
END_STRUCT_BLOCK;

FUNCTION ALIGN gnssSatellitesInView: INT;
VAR_INPUT
   type        : SINT := 0;
   tracked     : BOOL := TRUE;
   untracked   : BOOL := FALSE;
   gnss        : BOOL := TRUE;
   sbas        : BOOL := FALSE;
   satellites : ACCESS gnssSatellites;
END_VAR;
END_FUNCTION;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   sats  : gnssSatellites;
   gnss  : gnssFix;
   count : INT;
   i     : INT;
END_VAR;
```

```
gpsPower(power := ON);

BEGIN
   gnss();
   IF gnss.mode > 1 THEN
      count := gnssSatellitesInView(type:=0, satellites:=sats,
      sbas:=FALSE, untracked:=FALSE);
      DebugFmt(message:="satellites tracked: \1", v1:= count);
      FOR i := 1 TO count DO
         DebugFmt(message:="      SVID=\1, El=\2, Az=\3, SNR=\4",
                  v1:=sats.id[i], v2:=sats.elevation[i], v3:=sats.azimuth[i],
v4:=sats.SNR[i]);
      END_FOR;
   END_IF;
END;
END_PROGRAM;
```

4.2.16.5 gnssEnableType (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, CX1 warp(-c) mk2, NX-200, NX-900, LX2, LX5
Firmware version: 4.00 / 1.40.00

This function is used to enable and disable the different positioning systems.
By default all the relevant positioning systems are enabled and used to determine the position returned by gnssFix and gpsFix.

Before calling this function the GNSS receiver must be turned ON by a call to gpsPower() or gpsPowerLP.

Note: Not all GNSS receivers supports all the possible positioning systems and some GNSS receivers do not support all the combinations of positioning systems, see the technical manual for further details.

gnssGetEnabledSystems can be used to check which systems are currently enabled.

If a receiver does not support all the possible combinations, it may be necessary to use gnssEnableType to enable or disable multiple systems before the currently used systems change.

Input:

type: SINT (0..2, *Default 0*)
0 GPS
1 GLONASS
2 GALILEO
3 BeiDou

enable : BOOL (*Default: ON*)
Enables/disables the positioning system.

Returns: INT

0 - Success
-1 - GNSS module not powered or not supported.
-2 - Invalid parameter, type is not supported with this receiver.
-3 - Error performing action.

Declaration:

```
FUNCTION gnssEnableType : INT;
VAR_INPUT
   type   : SINT;
   enable : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Disable GLONASS
   gnssEnableType(type := 1, enable := OFF);
   ...
END;

END_PROGRAM;
```

4.2.16.6 gnssGetEnabledSystems (Function)

Architecture: **NX32L**
Device support: **NX-200, NX-900, LX2, LX5**
Firmware version: **2.30.00**

This function is used to get a bit mask of the currently enabled positioning systems.
Before calling this function the GNSS receiver must be turned ON by a call to gpsPower() or gpsPowerLP.

This can be used to check if the systems enabled using gnssEnableType actually have been enabled.
As some GNSS receivers do not support all possible combinations of positioning systems, it may require that multiple systems are enabled or disabled before the currently enabled systems change.

Input:
None

Returns: INT
>0 - Bit mask of enabled positioning systems:

Bit	System
0	GPS
1	GLONASS
2	GALILEO
3	BeiDou

0 - Not supported or no positioning systems are enabled.
-1 - GNSS module not powered or not supported.
-2 - Invalid parameter. system might not be supported on this device.
-3 - Error performing action.

Declaration:

```
FUNCTION gnssGetEnabledSystems : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
```

```
VAR
   rc : INT;
END_VAR
   gpsPower(power := ON);

   // Get enabled systems:
   rc := gnssGetEnabledSystems();
   IF rc < 0 THEN
      DebugFmt(message:="gnssGetEnabledSystems failed: \1", v1:=rc);
   ELSE
      IF rc AND (1 <> 0) THEN
         DebugMsg(message:="GPS");
      END_IF;
      IF rc AND (2 <> 0) THEN
         DebugMsg(message:="GLONASS");
      END_IF;
      IF rc AND (4 <> 0) THEN
         DebugMsg(message:="Galileo");
      END_IF;
      IF rc AND (8 <> 0) THEN
         DebugMsg(message:="BeiDou");
      END_IF;
   END_IF;
BEGIN
   ...
END;

END_PROGRAM;
```

4.2.16.7 gnssDynamicModelSet (Function)

Architecture: NX32, NX32L
Device support: MX2 turbo LTE UDR, NX-200 DR.
Firmware version: 5.13 / 1.74.00

This configures the dynamic model used by the GNSS module.

Input:

model: INT (*Default 4*)

2 Stationary
3 Pedestrian
4 Automotive
5 Sea
10 Bike

Returns:

1 - Success
0 - Unsupported
-1 - Invalid model.
-2 - Generic error
-3 - GNSS power is OFF

Declaration:

```
FUNCTION gnssDynamicModelSet : INT;
VAR_INPUT
   model  : INT := 4;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Switch the dynamic model to the bike model
   gnssDynamicModelSet(model:=10);
   ...
END;

END_PROGRAM;
```

4.2.16.8 gnssDRStatus (Function)

Architecture: NX32, NX32L
Device support: MX2 turbo LTE UDR, NX-200 DR.
Firmware version: 4.56 / 1.74.00

This returns the status for Dead Reckoning.

Dead Reckoning is a feature where the GNSS module uses internal sensors to calculate the position.
This improves performance in areas with poor or no signal conditions. (such as tunnels and parking garages)

The RTCU device must be securely mounted for Dead Reckoning to work.

Input:

None

Returns:

0 - Unsupported or GNSS power is OFF.
1 - Calibrating.
2 - Fusion mode. (Dead Reckoning is used to improve positions.)
3 - Suspended. (Unexpected motion. Dead Reckoning is temporary disabled)
4 - Disabled. (Continuously unexpected motion, Dead Reckoning is disabled until powered off)

Declaration:

```
FUNCTION gnssDRStatus : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Dead Reckoning status
   IF gnssDRStatus() > 2 THEN
      // The device is relocated
      // Is this a service? or tampering?
      ...
   END_IF;
   ...
```

```
END;

END_PROGRAM;
```

4.2.16.9 gnssDREnable (Function)

Architecture: NX32, NX32L
Device support: MX2 turbo LTE UDR, NX-200 DR.
Firmware version: 5.16 / 1.93.00

This function is used to enable and disable Dead Reckoning.

Dead Reckoning is enabled by default on supported devices, but there may be situations where it can be advantageous to disable it.
gnssDRStatus can be used to check if it is disabled.

Input:

enable: BOOL (*Default TRUE*)
Set to false to disable Dead Reckoning.

Returns:

1 - Success
0 - Unsupported
-2 - Generic error
-3 - GNSS power is OFF

Declaration:

```
FUNCTION gnssDREnable : INT;
VAR_INPUT
   enable : BOOL := TRUE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Disable DR
   gnssDREnable(enable := FALSE);
      ...
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.16.10 gnssDRMountSetup (Function)

Architecture: NX32, NX32L
Device support: MX2 turbo LTE UDR, NX-200 DR.
Firmware version: 5.13 / 1.74.00

This configures one of the vectors which specify the location of the RTCU Device, the GNSS Antenna and the Vehicle Reference Point (VRP) in relation to each other.

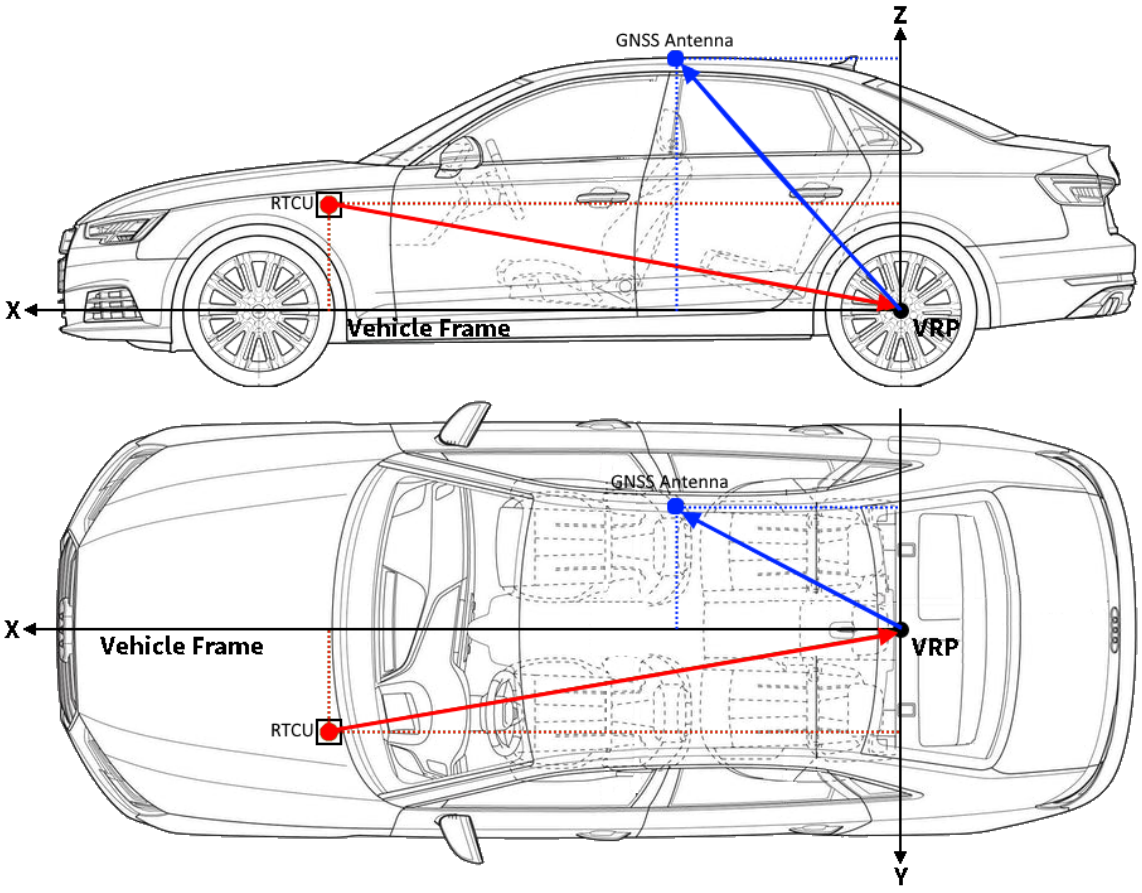
If the RTCU device is placed at a significant distance from the GNSS Antenna or the VRP, then errors can be introduced in the navigation solution (particularly when the vehicle experiences high heading rate).
Setting the mount location of the GNSS Antenna and the RTCU device, will compensate for these errors.

The vectors are located in the Vehicle Frame, originating from the VRP or the RTCU device.

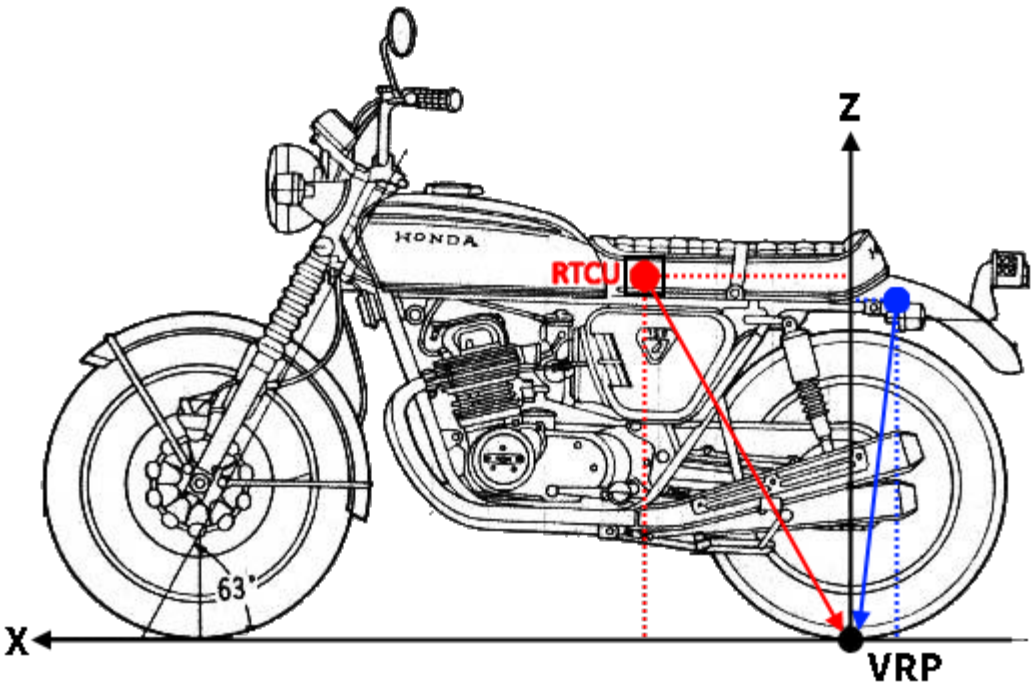
The VRP is the nominal point at which some motion constraints inherent to ground based vehicles are applied.
For automotive platforms, the VRP is defined as the center of the vehicle rear axle.
For bike platforms, the VRP is defined as the point where the wheel touches the ground. If the RTCU device is mounted on the handle bar, the VRP will be where the front wheel touches the ground, and if the RTCU device is mounted on the bike frame the VRP will be where the back wheel touches the ground.

The Vehicle Frame is a right-handed 3D Cartesian frame connected to the vehicle.
The x-axis points towards the front of the vehicle, the y-axis points towards the left of the vehicle and the z-axis completes the right-handed reference system by pointing up.

This illustrates how to determine the vectors on automotive platforms:



And for bike platforms:



Input:

type: SINT
The vector to change:
0 VRP to GNSS antenna
1 VRP to RTCU device
2 RTCU device to GNSS antenna
3 RTCU device to VRP

x: INT
The x-axis of the vector in cm.

y: INT
The y-axis of the vector in cm.

z: INT
The z-axis of the vector in cm.

Returns:

1 - Success
0 - Unsupported
-1 - Invalid input.
-2 - Generic error
-3 - GNSS power is OFF

Declaration:

```
FUNCTION gnssDRMountSetup : INT;
VAR_INPUT
    x      : INT;
    y      : INT;
    z      : INT;
```

```
   type  : SINT;
END_VAR;
```

Example:

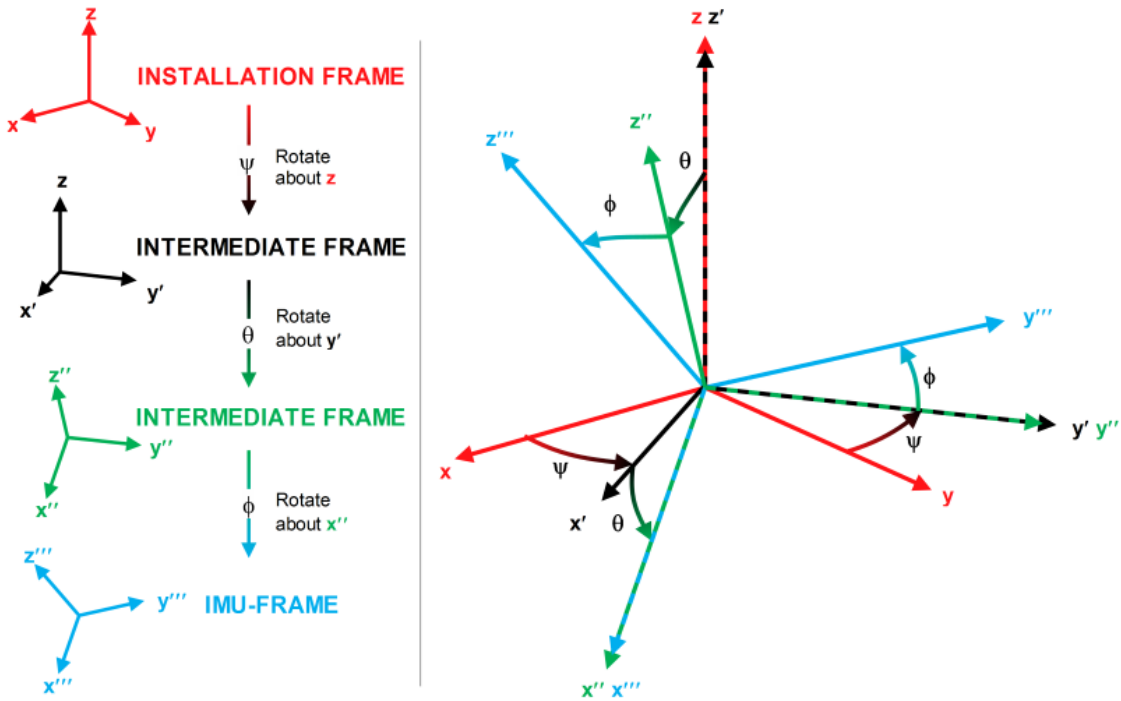
```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Specify vector from reference point to GNSS antenna
   gnssDRMountSetup(type := 0, x := 100, y := 0, z := 150);
   // Specify vector from reference point to RTCU device
   gnssDRMountSetup(type := 1, x := -20, y := -50, z := 20);
   ...
END;

END_PROGRAM;
```

4.2.16.11 gnssDRAlignSetup (Function)

Architecture: NX32, NX32L
Device support: MX2 turbo LTE UDR, NX-200 DR
Firmware version: 5.13 / 1.74.00

This configures the alignment between the Device Frame and the Vehicle Frame.

The alignment are the Euler angles required to rotate the Vehicle Frame to the Device Frame. The yaw rotation should be performed first, then the pitch and finally the roll. At each stage, the rotation is around the appropriate axis of the transformed Vehicle Frame, meaning that the order of the rotation sequence is important (see figure below).



The Device Frame is a right-handed 3D Cartesian frame connected to the RTCU device.

The x-axis points towards the back of the device (where the connectors and antenna connectors are present), the y-axis points towards the left of the device and the z-axis points up.
It is centered on the RTCU device.

The Vehicle Frame is a right-handed 3D Cartesian frame connected to the vehicle.
The x-axis points towards the front of the vehicle, the y-axis points towards the left of the vehicle and the z-axis completes the right-handed reference system by pointing up.
It is centered on the RTCU device.

Input:

automatic : BOOL (*Default TRUE*)
If true, the calibration is done automatically.
If false, the provided values are used.

yaw: DINT (0..36000)
The module mount yaw angle in 1/100°.

pitch: INT (-9000..9000)
The module mount pitch angle in 1/100°.

roll: INT (0..18000)
The module mount roll angle in 1/100°.

Returns:

1	- Success
0	- Unsupported
-1	- Invalid input.
-2	- Generic error
-3	- GNSS power is OFF

Declaration:

```
FUNCTION gnssDRAlignSetup : INT;
VAR_INPUT
   yaw       : INT;
   pitch     : INT;
   roll      : INT;
   automatic : BOOL := TRUE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Set yaw to 10.5°, pitch to 13.5° and roll to 5.4°.
   gnssDRAlignSetup(automatic:=FALSE, yaw := 1050, pitch := 1350, roll :=
540);
   ...
END;

END_PROGRAM;
```

4.2.16.12 gnssDRWheelTickSetup (Function)

Architecture: NX32, NX32L
Device support: NX-200 DR.
Firmware version: 5.13 / 1.74.00

This configures the Wheel Tick support needed for Automotive Dead Reckoning.

Wheel ticks provide the GNSS module with information about how the vehicle moves to improve the positioning when there is bad or no GNSS coverage.
The wheel ticks can be provided either by using two digital inputs as tick and direction or from software by either providing a tick counter and direction or the speed.
The software wheel ticks should be provided as often as possible for the best result.

Input:

mode : SINT (*Default 0*)
The wheel tick mode to use.
0 No wheel tick
1 Hardware Wheel Tick using two digital inputs.
2 Software using ticks sent with gnssDRWheelTickPushTick
3 Software using speed sent with gnssDRWheelTickPushSpeed

hw_tick: INT
The number of the digital input to use as the wheel tick. Only used when mode is 1.

hw_tick_inv: BOOL
Invert the polarity of the tick digital input. Only used when mode is 1.

hw_dir: INT
The number of the digital input to use as the direction input. By default, active is forward. Only used when mode is 1.

hw_dir_inv: BOOL
Invert the polarity of the direction digital input. Only used when mode is 1.

scale: DINT
Specifies the distance represented by each wheel tick in µm.
If not set (or set to zero) it will be estimated during calibration.

quant: DINT
Wheel-tick quantization in µm. If mode is 3, then this is interpreted as the speed measurement error RMS in µm/s.

countmax: DINT
Wheel-tick counter maximum value (rollover - 1). Set to zero if relative wheeltick is used. This parameter is only used in mode 2.

latency: UINT
Wheel-tick data latency due to e.g. CAN bus in ms.

freq: USINT
Nominal wheel-tick data frequency in Hz.

Returns:

1 - Success
0 - Unsupported
-1 - Invalid input.
-2 - Generic error
-3 - GNSS power is OFF
-4 - Hardware input is not available.

Declaration:

```
FUNCTION gnssDRWheelTickSetup : INT;
VAR_INPUT
   mode           : SINT := 0;
   hw_tick        : INT;
   hw_dir         : INT;
   hw_tick_inv    : BOOL;
   hw_dir_inv     : BOOL;
   scale          : DINT;
   quant          : DINT;
   countmax       : DINT;
   latency        : UINT;
   freq           : USINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Configure to use HW wheel tick on DIN1 and DIN2
   gnssDRWheelTickSetup(mode:=1, hw_tick := 1, hw_dir := 2, scale := 6646);
   ...
   // Configure to use relative SW wheel tick
   gnssDRWheelTickSetup(mode:=2, countmax := 0, scale := 6646);
   ...
END;

END_PROGRAM;
```

4.2.16.13 gnssDRWheelTickPushTick (Function)

Architecture: NX32, NX32L
Device support: NX-200 DR
Firmware version: 5.13 / 1.74.00

This pushes tick data to the GNSS module.
The GNSS module must have been configured to use SW ticks with gnssDRWheelTickSetup.

Input:

tick: DINT (*Default 0)*
The tick count. The countmax parameter in gnssDRWheelTickSetup specifies if this is absolute or relative ticks.

backwards : BOOL (*Default FALSE)*
Set to true to report backwards movement, leave at false for forward movement.

Returns:

1 - Success
0 - Unsupported
-1 - Invalid input.
-2 - Generic error
-3 - GNSS power is OFF

Declaration:

```
FUNCTION gnssDRWheelTickPushTick : INT;
VAR_INPUT
   backwards : BOOL := FALSE;
   tick      : DINT := 0;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Send ticks to GNSS module
   gnssDRWheelTickPushTick(tick := 105);
      ...
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.16.14 gnssDRWheelTickPushSpeed (Function)

Architecture: NX32, NX32L
Device support: NX-200 DR.
Firmware version: 5.13 / 1.74.00

This pushes speed data to the GNSS module.
The GNSS module must have been configured to use SW speed with gnssDRWheelTickSetup.

Input:

speed: DINT (*Default 0*)
The forwards speed in m/s. For backwards speed, use negative values.
The speed is scaled by 1000. (A speed of 1 m/s will be given as 1000)

Returns:

1 - Success
0 - Unsupported
-1 - Invalid input.
-2 - Generic error
-3 - GNSS power is OFF

Declaration:

```
FUNCTION gnssDRWheelTickPushSpeed : INT;
VAR_INPUT
   speed     : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Set speed to 14 m/s
   gnssDRWheelTickPushSpeed(speed := 14);
      ...
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.17 gprs: GPRS functions

4.2.17.1 GPRS: Mobile network functions

The mobile network functions are used for establishing a connection via a cellular network (GPRS/UMTS/LTE) to the internet.

• gprsOpen	Starts a mobile network session.
• gprsClose	Stops an active mobile network session.
• gprsConnected	Queries if a mobile network session is active.
• gprsSetModemInit	Sets modem init. string for mobile network session.
• gprsGetMonitorParm	Gets the parameters for mobile network connection monitor.
• gprsSetMonitorParm	Sets the parameters for mobile network connection monitor.

4.2.17.2 gprsOpen (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This starts up the cellular network connection to the internet.

It must be called before any communication that uses the cellular network can occur. The network connection has not actually been established when this function returns. The gprsConnected() function will indicate when the network connection has been established and is ready for communication.

An incoming data call (CSD) will terminate the mobile network session which will then automatically be restored when the data call terminates.

When the network session is opened, there is no need for further intervention from the VPL program to keep the session up. If, for some reason, the mobile network session terminates, the firmware will automatically reconnect and establish the mobile network session again.

Input:
None.

Returns: **INT**
0 - Success.
1 - GSM module is powered off.
2 - Cellular network not supported by device.

Declaration:

```
FUNCTION gprsOpen : INT;
```

Example:

Please see the "Examples - Socket Communication"

4.2.17.3 gprsClose (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This stops the cellular network connection to the internet. When this function returns, no communication that uses cellular network should occur. After this functions returns, the gprsConnected() function will indicate than the network connection is unavailable.

Input:
None.

Returns: **INT**
0 - Success.

Declaration:
```
FUNCTION gprsClose : INT;
```

Example:

Please see the "Examples - Socket Communication"

4.2.17.4 gprsConnected (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This returns information about whether a cellular network connection is established.

Note that that this does not mean that a connection to the RTCU Communication Hub has been established yet, even if it is enabled in Device: Network: RTCU Communication Hub settings. Use the function gwConnected() to check if connection has been established.

Input:
None.

Returns: **BOOL**
true if connection to the cellular network is established, false if not.

Declaration:
```
FUNCTION gprsConnected : BOOL;
```

Example:

Please see the "Examples - Socket Communication"

4.2.17.5 gprsSetModemInit (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 2.60 / 1.00.00

This function will set the modem initialization string parameter for making connection via mobile networks. It has the same effect as setting the parameters via Device: Network: Network Settings in the RTCU IDE.
The changes will take effect the next time the mobile network session is opened (gprsOpen function is called).

Input:

init : STRING (*Max 254 characters*)
The initialization parameters that need to be sent to the GSM module to set up a mobile network connection.

Returns: INT

0 - Success.
1 - String too long.

Declaration:

```
FUNCTION gprsSetModemInit;
VAR_INPUT
   init : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
...
gprsSetModemInit(init := "AT+CBST=0,0,1");
...
gprsOpen();
...

END_PROGRAM;
```

4.2.17.6 gprsSetMonitorParm (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 2.60 / 1.00.00

This function will set the parameters for the mobile network monitoring service.
The mobile network monitor will periodically check the health of the mobile network connection and automatically take the necessary recovery actions required to reestablish service.
Use of the mobile network monitoring service is only recommended in applications where the RTCU Communication Hub is not in use - such as pure FTP / SMTP applications. When using the RTCU Communication Hub, the health of the connection is automatically monitored with the periodic self-transaction.

The parameters for the mobile network monitoring service are non-volatile and therefore need only to be set once. Parameters take effect after a device resets (by using, for example, the boardReset function).

The parameter set by gprsSetMonitorParm can be read by using gprsGetMonitorParm.

Please note that this function is deprecated, please use the netSetMonitorParam function instead.

Input:

Enabled : BOOL
Determines whether the mobile network monitor service is enabled.

MaxAttempt : INT (1..60, *default 3*)
Maximum number of check attempts before mobile network recovery actions are initiated.
Recommended is 3 times.

AliveFreq : DINT (30..60000 seconds, *default 300*)
Frequency for checking the mobile network connection in seconds.
With this frequency, the connection will be checked "MaxAttempt" times before mobile network recovery actions are initiated.
Recommended alive-frequency is 300 seconds.

Returns:

None.

Declaration:

```
FUNCTION gprsSetMonitorParm;
VAR_INPUT
   Enabled    : BOOL;
   MaxAttempt : INT := 3;
   AliveFreq  : DINT := 300;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   gprscfg : gprsGetMonitorParm;
END_VAR;

// Check settings
gprscfg();
IF NOT gprscfg.Enabled OR gprscfg.MaxAttempt <> 3 OR gprscfg.AliveFreq <> 180
THEN
   // Set mobile network monitor parameters
   gprsSetMonitorParm(Enabled := TRUE, AliveFreq := 180);
   boardReset();
END_IF;

// Turn on power to the GSM module
gsmPower(power := ON);
gprsOpen();

BEGIN
```

```
   ...
END;
END_PROGRAM;
```

4.2.17.7 gprsGetMonitorParm (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 2.60 / 1.00.00

This function will read out the parameters for the mobile network monitoring service previously set by using the gprsSetMonitorParm function.

Please note that this function is deprecated, please use the netGetMonitorParam function block instead.

Input:

None.

Output:

Enabled : BOOL
Determines whether the mobile network monitor service is enabled.

MaxAttempt : INT
Maximum number of check attempts before mobile network recovery actions are initiated.

AliveFreq : DINT
Frequency for checking the mobile network connection in seconds.
With this frequency, the connection will be checked "MaxAttempt" times before mobile network recovery actions are initiated.

Declaration:

```
FUNCTION_BLOCK gprsGetMonitorParm;
VAR_OUTPUT
   Enabled    : BOOL;
   MaxAttempt : INT;
   AliveFreq  : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   gprscfg : gprsGetMonitorParm;
END_VAR;

// Check settings
gprscfg();
IF NOT gprscfg.Enabled OR gprscfg.MaxAttempt <> 3 OR gprscfg.AliveFreq <> 180
THEN
   // Set mobile network monitor parameters
   gprsSetMonitorParm(Enabled := TRUE, AliveFreq := 180);
   boardReset();
END_IF;
```

```
// Turn on power to the GSM module
gsmPower(power := ON);
gprsOpen();

BEGIN
   ...
END;
END_PROGRAM;
```

4.2.18 gps: GPS Receiver

4.2.18.1 GPS: GPS Receiver

The GPS functions allow the program to interact with a GPS/GNSS module if present on the specific RTCU device. Please consult the technical documentation for the actual RTCU device in use.

For a simple application that demonstrates the GPS functions, please have a look at the GPS Mobile application.

- gpsPower — Turns on power to the GPS receiver.
- gpsPowerLP — Turns on power to the GPS receiver in low power mode.
- gpsFix — Returns detailed information from the GPS receiver.
- gpsPointInPolygon — Checks whether a point is within a defined polygon.
- gpsEnableNMEA — Enables NMEA data output on a specific serial port.
- gpsDistance — Calculates the distance and bearing between two points.
- gpsDistanceX — Calculates the distance and bearing between two points.
- gpsGetAntennaStatus — Returns the status of the GPS antenna.
- gpsNMEA — Returns the NMEA strings.
- gpsSetSBAS — Enables SBAS-assisted positioning.
- gpsGetSBAS — Returns the SBAS enable status.
- gpsPPPStatus — Retrieves the number of satellites using Precise Point Positioning.
- gpsSetSpeedThreshold — Changes the speed threshold.
- gpsUpdateFreq — Change the position update frequency.
- gpsPositionToUtm — Converts a position to Universal Transverse Mercator (UTM) format.
- gpsUtmToPosition — Converts a position from Universal Transverse Mercator (UTM) format.
- gpsSemicircleToUtm — Converts a position to Universal Transverse Mercator (UTM) format.
- gpsUtmToSemicircle — Converts a position from Universal Transverse Mercator (UTM) format.

Note: all values returned by the GPS functions are in the WGS-84 datum.

External GPS devices
NX32L devices without built-in GPS modules can use an external GPS device, connected via the USB host port.
This means that the GPS functions can be used with the external GPS module as if it was built-in.
Some limitations to the available information should be expected.

To use an external GPS module follow these steps:
1. Connect the GPS device.
2. Ensure the RTCU is aware of the device. (With usbHostEnumerate)
3. Call gpsPower(ON)

It should now be possible to receive GPS positions.

Please note, that should the connection to the GPS device be lost, then the sequence above must be repeated again after calling gpsPower(OFF).

4.2.18.2 gpsPower (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, CX1, SX1, MX2 turbo/encore/warp, NX-200, NX-400, NX-900, LX2, LX5
Firmware version: 1.00 / 1.00.00

This function controls the power supply to the GPS receiver. By using this function, the GPS receiver can be switched off when not in use to save power. When the GPS receiver is switched on, it will take a number of seconds for it to acquire a new position. The duration of this period depends on how long it has been since the GPS receiver was last powered on.

Input:

power : BOOL
TRUE: Turns on the power.
FALSE: Turns off the power.

Returns:

None.

Declaration:

```
FUNCTION gpsPower;
VAR_INPUT
   power : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Turn on power to the GPS module
gpsPower(power := TRUE);

BEGIN
   ...
END;

END_PROGRAM;
```

4.2.18.3 gpsPowerLP (Function)

Architecture: NX32
Device support: MX2 turbo/encore/warp, CX1 series.
Firmware version: 4.00

Similar to the gpsPower() function this function controls the power to the GPS receiver.
This function operates the GPS module at a lower communication speed than gpsPower().
When using gpsPowerLP the satellites in view will not be retrieved and gpsUpdateFreq() is only supported with 1 position per. second.

Input:

power : BOOL
TRUE: Turns on the power.
FALSE: Turns off the power.

Returns: INT

0 - Success
1 - Not supported.
2 - GPS already powered on by gpsPower

Declaration:

```
FUNCTION gpsPowerLP : INT;
VAR_INPUT
   power : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Turn on power to the GPS module
gpsPowerLP(power := TRUE);

BEGIN
   ...
END;

END_PROGRAM;
```

4.2.18.4 gpsFix (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: MX2, CX1, SX1, MX2 turbo/encore/warp, NX-200, NX-400, NX-900, LX2, LX5
Firmware version: 1.00 (3.14: millisecond) / 1.00.00

The gpsFix function block is used to query the current position, time, and detailed satellite information from the internal GPS module.

Note regarding decimal minutes:
The value returned in decimal minutes, both *latdecmin* and *londecmin*, are assumed to be 4 digits large.
This means that the position 55 Deg. 51.3 Min. North is returned as *latdeg*=55 *latmin*=51 and *latdecmin*=3000, and the position 55 Deg. 52.0076 North is returned as *latdeg*=55 *latmin*=52 *latdecmin*=76.

SBAS Support:
Please see gpsSetSBAS.
When the "mode" parameter that is returned from gpsFix is 4, it indicates that the fix is not only a 3D fix but has also been SBAS-corrected.

Note regarding GPS with more than 12 channels:
While the full number of channels are used in position resolution, only the first 12 are included in the function block.

Note regarding HDOP, VDOP and PDOP:

The DOP values should be as low as possible. They describe the geometry of the satellites used. Generally a DOP value should not be higher than 6 as that will affect the accuracy negative.
Normally a DOP value will be between 2 and 3 when you have clear skies. In a city with houses screening the sky, the DOP value will be higher and accuracy will be affected.
One method to increase the accuracy of the used GPS positions is not to use fixes with an HDOP value higher than, for example, 300.

All values returned by gpsFix() are in the WGS-84 datum.

Input:
None.

Output:
mode : SINT;
0 = No info available (GPS not powered on?).
1 = Fix unavailable.
2 = 2D position fix.
3 = 3D position fix.
4 = 3D position fix with SBAS-correction (this requires that SBAS has been enabled, see gpsSetSBAS).
5 = Dead Reckoning fix.

linsec : DINT
Linsec for the timestamp received from the GPS receiver.
This is the same as the separate year, month, day, minute, and second but expressed as a linsec.
Also see clockLinsecToTime().

year : INT
Year (absolute).

month : SINT
Month (1..12).

day : SINT
Date (1..31).

hour : SINT
Hour (0..23).

minute : SINT
Minute (0..59).

second : SINT
Second (0..59).

millisecond : INT
Millisecond (0..999). Only available in firmware release 3.14 or newer.

latitude : DINT
Latitude expressed in a DINT.
Negative is south (ddmm.mmmm (multiplied by 10000)).

latsouth : BOOL
Direction of latitude (TRUE is south, FALSE is north).

latdeg : SINT
Latitude degress (0..90).

latmin : SINT
Latitude minutes (0..59).

latdecmin : INT
Latitude decimal minutes (0..9999).
This value is assumed to be 4 digits with the leading zeros removed.
Please see the note regarding decimal minutes above.

longitude : DINT
Longitude expressed in a DINT.
Negative is west (dddmm.mmmm (multiplied by 10000)).

lonwest : BOOL
Direction of longitude (TRUE is west, FALSE is east).

londeg : INT
Longitude degress (0..180).

lonmin : SINT
Longitude minutes (0..59).

londecmin : INT
Longitude decimal minutes (0..9999).
This value is assumed to be 4 digits with the leading zeros removed.
Please see note regarding decimal minutes above.

speed : DINT
Speed over ground in meters/hour.

course : DINT
Course over ground. In xxx.xx degrees (multiplied by 100) (0=north, 90=east, 225=southwest).

height : INT
Height in meters over mean sea level.

PDOP : INT
Position dilution of precision (multiplied by 100).

HDOP : INT
Horizontal dilution of precision (multiplied by 100).

VDOP : INT
Vertical dilution of precision (multiplied by 100).

inview : SINT
Number of satellites in view. This is the number of satellites that are above the horizon.
If gpsPowerLP is used, this will be 0.

used : SINT
Number of satellites used for the solution.

SVID[n] : SINT
Satellite ID.

El[n] : SINT
Elevation angle for satellite n.

Az[n] : INT
Azimuth angle for satellite n.

SNR[n] : SINT
Signal to noise ratio for satellite n. Max value is 55.

Declaration:

```
FUNCTION_BLOCK gpsFix;
VAR_OUTPUT
   mode          : SINT;     | 0=No info available, 1=Fix not available, 2=2D
position fix, 3=3D position fix, 4=3D pos. with SBAS
   linsec        : DINT;     | Linsec of time-stamp
   year          : INT;      | year (absolute, like 2004)
   month         : SINT;     | month (1..12)
   day           : SINT;     | date (1..31)
   hour          : SINT;     | hour (0..23)
   minute        : SINT;     | minute (0..59)
   second        : SINT;     | second (0..59)
   millisecond   : INT;      | millisecond (0..999)
   latitude      : DINT;     | Negative is South (ddmm.mmmm (multiplied by
10000))
   latsouth      : BOOL;     | True = South, False = North
   latdeg        : SINT;     | Degrees (0..90)
   latmin        : SINT;     | minutes (0..59)
   latdecmin     : INT;      | decimal minutes (0..9999)
   longitude     : DINT;     | Negative is West (dddmm.mmmm (multiplied by
10000))
   lonwest       : BOOL;     | True = West, False = East
   londeg        : INT;      | degrees (0..180)
   lonmin        : SINT;     | minutes (0..59)
   londecmin     : INT;      | decimal minutes (0..9999)
   speed         : DINT;     | Speed over ground in meters/hour
   course        : DINT;     | Course over ground. In xxx.xx degrees (multiplied
by 100)
   height        : INT;      | Height in meters over Mean Sea Level.
   PDOP          : INT;      | Position dilution of precision (PDOP) (multipied
by 100).
   HDOP          : INT;      | Horizontal dilution of precision (HDOP)
(multipied by 100).
   VDOP          : INT;      | Vertical dilution of precision (VDOP) (multipied
by 100).
   inview        : SINT;     | Number of satellites in view
   used          : SINT;     | Number of satellites used in solution

   // The following information is grouped, index 1 for all arrays is for the
first satellite etc.
   SVID          : ARRAY[1..12] OF SINT; // Satellite Vehicle number
   El            : ARRAY[1..12] OF SINT; // Elevation angle for satellite
   Az            : ARRAY[1..12] OF INT;  // Azimuth angle for satellite
   SNR           : ARRAY[1..12] OF SINT; // Signal to Noise ratio for satellite
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

FUNCTION ShowDecmin : STRING;
VAR_INPUT
```

```
   decmin : INT;
END_VAR;
   ShowDecmin := intToStr(v := decmin);
   WHILE strLen(str := ShowDecmin) < 4 DO
      ShowDecmin := strConcat(str1 := "0", str2 := ShowDecmin);
   END_WHILE;
END_FUNCTION;

PROGRAM test;
VAR
   gps   : gpsFix;
   index : SINT;
   str   : STRING;
END_VAR;


gpsPower(power := ON);
BEGIN
   gps();
   IF gps.mode > 1 THEN
      DebugFmt(message:="Mode=\1", v1:=gps.mode);
      DebugFmt(message:="   Linsec=\4", v4:=gps.linsec);
      DebugFmt(message:="   Time=\1:\2:\3", v1:=gps.hour, v2:=gps.minute,
v3:=gps.second);
      DebugFmt(message:="   Date=\1:\2:\3", v1:=gps.year, v2:=gps.month,
v3:=gps.day);
      DebugFmt(message:="   Lat: latitude=\4", v4:=gps.latitude);
      IF gps.latsouth THEN DebugMsg(message:="   Lat: South"); ELSE
DebugMsg(message:="   Lat: North"); END_IF;
      str := strConcat(str1 := "   Lat: Deg=\1 Min=\2 Dec=", str2 :=
ShowDecmin(decmin := gps.latdecmin));
      DebugFmt(message:=str, v1:=gps.latdeg, v2:=gps.latmin);
      DebugFmt(message:="   Lon: longitude=\4", v4:=gps.longitude);
      IF gps.lonwest THEN DebugMsg(message:="   Lon: West"); ELSE
DebugMsg(message:="   Lon: East"); END_IF;
      str := strConcat(str1 := "   Lon: Deg=\1 Min=\2 Dec=", str2 :=
ShowDecmin(decmin := gps.londecmin));
      DebugFmt(message:=str, v1:=gps.londeg, v2:=gps.lonmin);
      DebugFmt(message:="   Speed=\4", v4:=gps.speed);
      DebugFmt(message:="   Course=\4", v4:=gps.course);
      DebugFmt(message:="   Height=\1", v1:=gps.height);
      DebugFmt(message:="   PDOP=\1", v1:=gps.PDOP);
      DebugFmt(message:="   HDOP=\1", v1:=gps.HDOP);
      DebugFmt(message:="   VDOP=\1", v1:=gps.VDOP);
      DebugFmt(message:="   In view=\1, Used=\2", v1:=gps.inview,
v2:=gps.used);
      FOR index := 1 TO gps.inview DO
         DebugFmt(message:="      SVID=\1, El=\2, Az=\3, SNR=\4",
                  v1:=gps.SVID[index], v2:=gps.El[index], v3:=gps.Az[index],
v4:=gps.SNR[index]);
      END_FOR;
   END_IF;
END;
END_PROGRAM;
```

4.2.18.5 gpsPosition (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: All devices.
Firmware version: 1.00 / 1.00.00

Use this function block to read the current position from an external GPS smart-antenna connected to the serial port on the RTCU device.

On devices with an in-build GPS receiver the gpsFix() function should used.

When using this function to get the current position from an external GPS receiver, the external GPS receiver must send the RMC status with regular intervals (typically every second). The baud rate must be 4800 bps and the external GPS receiver must be connected to the RTCU through RS232 port #0.

When using the GPS SmartAntenna with the RTCU DX4i pro it is important to first power on the SmartAntenna by using boardDCOut function.

Input:

None.

Output:

valid : BOOL;
TRUE if GPS has valid position, otherwise FALSE.

year : SINT
Years since 2000.

month : SINT
Month (1..12).

day : SINT
Date (1..31).

hour : SINT
Hour (0..23).

minute : SINT
Minute (0..59).

second : SINT
Second (0..59).

latdegrees : SINT
Negative is south (-90..+90).

latminutes : SINT
Minutes (0..59).

latdecimalminutes : INT
Decimal minutes (0..9999).

londegrees : INT
Negative is west (-180..+180).

lonminutes : SINT
Minutes (0..59).

londecimalminutes : INT
Decimal minutes (0..9999).

speed : DINT
Speed over ground in meters/hour.

course : DINT
Course over ground. In xxx.xx degrees (multiplied by 100).

Declaration:

```
FUNCTION_BLOCK gpsPosition;
VAR_OUTPUT
   valid             : BOOL;    | GPS has valid position.
   year              : SINT;    | years since 2000
   month             : SINT;    | month (1..12)
   day               : SINT;    | date (1..31)
   hour              : SINT;    | hour (0..23)
   minute            : SINT;    | minute (0..59)
   second            : SINT;    | second (0..59)
   latdegrees        : SINT;    | Negative is South (-90..+90)
   latminutes        : SINT;    | minutes (0..59)
   latdecimalminutes : INT;     | decimal minutes (0..9999)
   londegrees        : INT;     | Negative is West (-180..+180)
   lonminutes        : SINT;    | minutes (0..59)
   londecimalminutes : INT;     | decimal minutes (0..9999)
   speed             : DINT;    | Speed over ground in meters/hour
   course            : DINT;    | Course over ground. In xxx.xx degrees
(multiplied by 100)
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

VAR
   gps : gpsPosition;
END_VAR;



// Turn on the GPS SmartAntenna power on the RTCU DX4i pro.
boardDCOut(enable:=on);
BEGIN
   // Call the GPS to get a new position fix
   gps();
   // Check for valid position from GPS
   IF gps.valid THEN
      DebugMsg(message:=strFormat(format:="valid=\1", v1:=INT(gps.valid)));

      DebugMsg(message:=strFormat(format:="latdegrees=\4",
v4:=gps.latdegrees));
      DebugMsg(message:=strFormat(format:="latminutes=\4",
v4:=gps.latminutes));
      DebugMsg(message:=strFormat(format:="latdecimalminutes=\4",
v4:=gps.latdecimalminutes));

      DebugMsg(message:=strFormat(format:="londegrees=\4",
v4:=gps.londegrees));
      DebugMsg(message:=strFormat(format:="lonminutes=\4",
v4:=gps.lonminutes));
      DebugMsg(message:=strFormat(format:="londecimalminutes=\4",
v4:=gps.londecimalminutes));

      DebugMsg(message:=strFormat(format:="cog=\4", v4:=gps.course));
      DebugMsg(message:=strFormat(format:="sog=\4", v4:=gps.speed));

      DebugMsg(message:=strFormat(format:="year=\4", v4:=gps.year));
      DebugMsg(message:=strFormat(format:="month=\4", v4:=gps.month));
      DebugMsg(message:=strFormat(format:="day=\4", v4:=gps.day));
```

```
        DebugMsg(message:=strFormat(format:="hour=\4", v4:=gps.hour));
        DebugMsg(message:=strFormat(format:="min=\4", v4:=gps.minute));
        DebugMsg(message:=strFormat(format:="sec=\4", v4:=gps.second));
    END_IF;
END;

END_PROGRAM;
```

4.2.18.6 gpsPointInPolygon (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: MX2, CX1, SX1, MX2 turbo/encore/warp, NX-200, NX-400, NX-900, LX2, LX5
Firmware version: 1.00 / 1.00.00

This function block is used for calculating if a given point is within a defined polygon with up to 7 sides. The function block uses points that are given by DINTs. When using this function together with GPS positions, the latitude/longitude positions are normally converted to DINT values by the user's program. Typically, this is done in the following way:
The degrees, minutes, and decimal minutes are each multiplied by a factor, so that the resulting DINT is formatted as follows:

DDD_MM_CCCC

where DDD is degrees (to be multiplied by 1,000,000), MM is minutes (to be multiplied by 10,000) and CCCC is decimal minutes. This means that all possible latitudes and longitudes can be contained within a DINT.

Please note:
The polygon corners XY positions must be presented to the function block clockwise, and in order to close the polygon, the first non-used entry in the definition of the polygon must point to the first position in order to close the polygon. Please see the example below.

Input:

Position_X : DINT
X part of the point to check.

Position_Y : DINT
Y part of the point to check.

Polygon_X : ARRAY[1..8] OF DINT
X part of each corner of the polygon.

Polygon_Y : ARRAY[1..8] OF DINT
Y part of each corner of the polygon.

Corners : SINT
Number of corners that define the polygon.

Output:

Within : BOOL
TRUE if Position_X/Position_Y is within the defined polygon, FALSE if outside of polygon.

Declaration:

```
FUNCTION_block gpsPointInPolygon;
VAR_INPUT
```

```
   // Position 1 parameters:
   Position_X    : DINT;                 | X part of the position to check
   Position_Y    : DINT;                 | Y part of the position to check

   // Polygon parameters:
   Polygon_X     : ARRAY[1..8] OF DINT; | X part of the polygon
   Polygon_Y     : ARRAY[1..8] OF DINT; | Y part of the polygon

   Corners       : SINT;                 | Number of corners defined in the
polygon
END_VAR;
VAR_OUTPUT
   Within        : BOOL;                 | True if Position_X/Position_Y is
within the polygon
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   Poly_x      : ARRAY[1..8] OF DINT;
   Poly_y      : ARRAY[1..8] OF DINT;
   P_x         : DINT;
   P_y         : DINT;
   checkPoly   : gpsPointInPolygon;
END_VAR;

   // Define a polygon with corners in (0,0) (0,1000000) (1000000,1000000)
(0,1000000) starting with lower left corner
   // going clockwise: (X is the point to search for (500000, 500000)
   //   ------------------
   //   |                 |
   //   |                 |
   //   |                 |
   //   |        X        |
   //   |                 |
   //   |                 |
   //   |                 |
   //   ------------------
   //
   poly_y[1] := 0_00_0000;
   poly_x[1] := 0_00_0000;

   poly_y[2] := 1_00_0000;
   poly_x[2] := 0_00_0000;

   poly_y[3] := 1_00_0000;
   poly_x[3] := 1_00_0000;

   poly_y[4] := 0_00_0000;
   poly_x[4] := 1_00_0000;

   poly_y[5] := poly_y[1]; // The last entry MUST point to the first in order
to "close" the polygon
   poly_x[5] := poly_x[1];

   P_y       := 0_50_0000; //   This point is within the polygon
   P_x       := 0_50_0000;

   checkPoly(Position_x := P_x, Position_y := P_y,
             Polygon_x[1] := poly_x[1], Polygon_y[1] := poly_y[1],
```

```
                Polygon_x[2] := poly_x[2], Polygon_y[2] := poly_y[2],
                Polygon_x[3] := poly_x[3], Polygon_y[3] := poly_y[3],
                Polygon_x[4] := poly_x[4], Polygon_y[4] := poly_y[4],
                Polygon_x[5] := poly_x[5], Polygon_y[5] := poly_y[5],
                Corners := 5 ); // Number of corners that defines the polygon.

   IF checkPoly.Within THEN
      DebugMsg(message:="Inside");
   ELSE
      DebugMsg(message:="Outside");
   END_IF;

END_PROGRAM;
```

4.2.18.7 gpsEnableNMEA (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, CX1, SX1, MX2 turbo/encore/warp, NX-200, NX-400, NX-900, LX2, LX5
Firmware version: 1.00 / 1.00.00

By using this function, it is possible to redirect, or copy, the raw NMEA sentences that have been received from the internal GPS receiver to a specific serial port on the RTCU device. This makes it possible to connect a PDA or similar to the serial port and be able to receive navigational data without investing in a separate GPS receiver. Please note that the serial port must be open and configured prior to calling this function. If the serial port that is to be used is configured for a baud rate lower than 4800 baud, the baud rate used will be 4800.
Please note that the NMEA sentences will only be forwarded to the specified port during the call of gpsFix(). It is therefore important that the function is called regularly.

The normal GPS function, gpsFix(), still continues to work regardless of this function.

Input:

port : SINT
This specifies which serial port to copy the NMEA data to. Please see the serOpen() for an explanation of allowed values.

enable : BOOL
TRUE: The raw NMEA sentences will be copied to the serial port specified.
FALSE: No NMEA sentences are copied.

Returns:

None.

Declaration:

```
FUNCTION gpsEnableNMEA;
VAR_INPUT
   port   : SINT;
   enable : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
VAR
   portNum : SINT := 0;             // Select which port to use for the
communication
   gps     : gpsFix;
END_VAR;
```

```
PROGRAM test;

// Turn on power to the GPS module
gpsPower(power := TRUE);
// Open the serial port, set baudrate, parity and number of bits
serOpen(port:=portNum, baud:=9600, bit:=8, parity:=0);
// Enable NMEA output on the serial port
gpsEnableNMEA(port:=PortNum, enable:=TRUE);

BEGIN
   ...
   gps();
   ...
END;

END_PROGRAM;
```

4.2.18.8 gpsDistance (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: MX2, CX1, SX1, MX2 turbo/encore/warp, NX-200, NX-400, NX-900, LX2, LX5
Firmware version: 1.00 / 1.00.00

This function block is used for calculating the distance and bearing (direction) between two latitude/longitude positions.
The input to this function block is the same format as the gpsFix() deliver positions in.

Also see gpsDistanceX().

Input:

Position 1:

lat1south : BOOL
True is south, false is north.

lat1deg : SINT
Degrees (0..90).

lat1min : SINT
Minutes (0..59).

lat1decmin : INT
Decimal minutes (0..9999).

lon1west : BOOL
True is west, false is east.

lon1deg : INT
Degrees (0..180).

lon1min : SINT
Minutes (0..59).

lon1decmin : INT
Decimal minutes (0..9999).

Position 2:

lat2south : BOOL
True is south, false is north.

lat2deg : SINT
Negative is south (-90..+90).

lat2min : SINT
Minutes (0..59).

lat2decmin : INT
Decimal minutes (0..9999).

lon2west : BOOL
True is west, false is east.

lon2deg : INT
Negative is west (-180..+180).

lon2min : SINT
Minutes (0..59).

lon2decmin : INT
Decimal minutes (0..9999).

Output:

distance : DINT
Distance between the two positions in meters.

bearing : INT
Bearing between the two points in degrees.

Declaration:

```
FUNCTION_BLOCK gpsDistance;
VAR_INPUT
   // Position 1 parameters:
   lat1south    : BOOL;     | True is South, False is North
   lat1deg      : SINT;     | degrees (0..90)
   lat1min      : SINT;     | minutes (0..59)
   lat1decmin   : INT;      | decimal minutes (0..9999)
   lon1west     : BOOL;     | True is West, False is East
   lon1deg      : INT;      | degrees (0..180)
   lon1min      : SINT;     | minutes (0..59)
   lon1decmin   : INT;      | decimal minutes (0..9999)

   // Position 2 parameters:
   lat2south    : BOOL;     | True is South, False is North
   lat2deg      : SINT;     | degrees (0..90)
   lat2min      : SINT;     | minutes (0..59)
   lat2decmin   : INT;      | decimal minutes (0..9999)
   lon2west     : BOOL;     | True is West, False is East
   lon2deg      : INT;      | degrees (0..180)
   lon2min      : SINT;     | minutes (0..59)
   lon2decmin   : INT;      | decimal minutes (0..9999)
END_VAR;
VAR_OUTPUT
   distance     : DINT;     | Distance between the two points in meters
   bearing      : INT;      | Direction in degrees between the two points
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_OUTPUT
   gsmConnect : BOOL; | Indicates that the GSM is connected to a basestation
(Green)
   gpsValid   : BOOL; | Indicates that a position fix has been obtained (Red)
END_VAR;

PROGRAM GPS_Example;
VAR
   sms      : gsmIncomingSMS;
   gps      : gpsFix;
   gc       : gpsDistance;
   str      : STRING;
   awaitFix : BOOL;
END_VAR;
   // Turn on power to the GPS receiver
   gpsPower(power:=TRUE);
   // Turn on power to the GSM module
   gsmPower(power:=TRUE);
BEGIN
   // Update GPS data
   gps();
   // If we got a valid fix from the GPS
   IF gps.mode > 1 THEN
      // Calculate distance and bearing to Logic IO in Denmark
      gc(lat1south:=FALSE, lat1deg:=55, lat1min:=51, lat1decmin:=3078,
lon1west:=FALSE, lon1deg:=9, lon1min:=51, lon1decmin:=530,
         lat2south:=gps.latsouth, lat2deg:=gps.latdeg, lat2min:=gps.latmin,
lat2decmin:=gps.latdecmin,
         lon2west:=gps.lonwest, lon2deg:=gps.londeg, lon2min:=gps.lonmin,
lon2decmin:=gps.londecmin);
      // Build string with information
      str:=strFormat(format:="Distance to Logic IO=\4.\3 KM, bearing=\2 deg",
v4:=gc.distance/1000, v3:=INT(gc.distance MOD 1000), v2:=gc.bearing);
   END_IF;

   // If we receive a SMS message, we want the next GPS position that is valid
   IF sms.status > 0 THEN
      awaitFix:=TRUE;
   END_IF;

   // If we are waiting for next valid GPS position, and GPS position is
valid,
   IF awaitFix AND gps.mode > 1 THEN
      awaitFix:=FALSE;
      // Send an SMS with the position
      gsmSendSMS(phonenumber:=sms.phonenumber, message:=str);
   END_IF;

   // Indicate on LED (green part) if we are connected to a GSM basestation
   gsmConnect:=gsmConnected();
   // Indicate on LED (red part) if valid GPS position
   gpsValid:=gps.mode > 1;
END;

END_PROGRAM;
```

4.2.18.9 gpsDistanceX (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: MX2, CX1, SX1, MX2 turbo/encore/warp, NX-200, NX-400, NX-900, LX2, LX5
Firmware version: 1.00 / 1.00.00

This function block is used for calculating the distance and bearing (direction) between two latitude/longitude positions. Compared to the gpsDistance() function block, gpsDistanceX() expects the latitude/longitude to be expressed in two DINT variables. The gpsDistanceX() function is therefore more efficient to use than gpsDistance().

Input:

Position 1:

latitude1 : DINT
Negative is south (ddmm.mmmm (multiplied by 10000)).

longitude1 : DINT
Negative is west (dddmm.mmmm (multiplied by 10000)).

Position 2:

latitude2 : DINT
Negative is south (ddmm.mmmm (multiplied by 10000)).

longitude2 : DINT
Negative is west (dddmm.mmmm (multiplied by 10000)).

Output:

distance : DINT
Distance between the two positions in meters.

bearing : INT
Bearing between the two points in degrees.

Declaration:

```
FUNCTION_BLOCK gpsDistanceX;
VAR_INPUT
   // Position 1 parameters:
   latitude1    : DINT;    | Negative is South (ddmm.mmmm (multiplied by
10000))
   longitude1   : DINT;    | negative is West (dddmm.mmmm (multiplied by
10000))

   // Position 2 parameters:
   latitude2    : DINT;    | Negative is South (ddmm.mmmm (multiplied by
10000))
   longitude2   : DINT;    | negative is West (dddmm.mmmm (multiplied by
10000))
END_VAR;
VAR_OUTPUT
   distance     : DINT;    | Distance between the two points in meters
   bearing      : INT;     | Direction in degrees between the two points
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_OUTPUT
   gsmConnect : BOOL; | Indicates that the GSM is connected to a basestation
(Green)
   gpsValid   : BOOL; | Indicates that a position fix has been obtained (Red)
END_VAR;

PROGRAM GPS_Example;
VAR
   sms      : gsmIncomingSMS;
   gps      : gpsFix;
   gc       : gpsDistanceX;
   str      : STRING;
   awaitFix : BOOL;
END_VAR;
   // Turn on power to the GPS receiver
   gpsPower(power:=TRUE);
   // Turn on power to the GSM module
   gsmPower(power:=TRUE);
BEGIN
   // Update GPS data
   gps();
   // If we got a valid fix from the GPS
   IF gps.mode > 1 THEN
      // Calculate distance and bearing to Logic IO in Denmark
      gc(latitude1:=55513078, longitude1:=9510530, latitude2:=gps.latitude,
longitude2:=gps.longitude);
      // Build string with information
      str:=strFormat(format:="Distance to Logic IO=\4.\3 KM, bearing=\2 deg",
           v4:=gc.distance/1000, v3:=INT(gc.distance MOD 1000),
v2:=gc.bearing);
   END_IF;

   // If we receive a SMS message, we want the next GPS position that is valid
   IF sms.status > 0 THEN
      awaitFix:=TRUE;
   END_IF;

   // If we are waiting for next valid GPS position, and GPS position is
valid,
   IF awaitFix AND gps.mode > 1 THEN
      awaitFix:=FALSE;
      // Send an SMS with the position
      gsmSendSMS(phonenumber:=sms.phonenumber, message:=str);
   END_IF;

   // Indicate on LED (green part) if we are connected to a GSM basestation
   gsmConnect:=gsmConnected();
   // Indicate on LED (red part) if valid GPS position
   gpsValid:=gps.mode > 1;
END;

END_PROGRAM;
```

4.2.18.10 gpsGetAntennaStatus (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2i, CX1, MX2 turbo/encore/warp, NX-200, NX-900, LX2, LX5
Firmware version: 1.10 / 1.40.00

This returns the status for the GPS antenna. By using this function, it can be determined whether a GPS antenna is connected or if a short circuit is present.

Be aware that not all devices supports this functionality. Please refer to the technical manual of the device in question.

Input:

None

Returns:

0 - Unknown/unsupported or GPS power is OFF.
1 - GPS antenna present.
2 - GPS antenna is short-circuited.
3 - GPS antenna is missing.

Declaration:

```
FUNCTION gpsGetAntennaStatus : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Test GPS antenne status
   IF gpsGetAntennaStatus() > 1 THEN
      // Something is wrong
      ...
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.18.11 gpsNMEA (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: MX2i,CX1, SX1, MX2 turbo/encore/warp, NX-200, NX-400, NX-900, LX2, LX5
Firmware version: 1.21 / 1.00.00

This returns the NMEA strings that correspond to the last call to gpsFix.
The following NMEA strings will be returned:

- RMC.

- VTG.
- GGA.
- GSA.

The information returned by gpsNMEA must be considered invalid when the mode returned by gpsFix is equal to 1. In this case, the data should be discarded by the application.
Also consult the relevant literature for more information about NMEA verbs interpretation.

Input:

None.

Output:

RMC : STRING
String containing the RMC verb.

VTG : STRING
String containing the VTG verb.

GGA : STRING
String containing the GGA verb.

GSA : STRING
String containing the GSA verb.

Declaration:

```
FUNCTION_BLOCK gpsNMEA;
VAR_OUTPUT
   RMC : STRING;
   VTG : STRING;
   GGA : STRING;
   GSA : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   gps  : gpsFix;
   nmea : gpsNMEA;
END_VAR;

   gpsPower(power := ON);

BEGIN
   ...
   // Generate GPS fix:
   gps();
   // Get the NMEA strings
   nmea();
   DebugMsg(message := "NMEA:");
   DebugMsg(message := "-rmc=" + nmea.rmc);
   DebugMsg(message := "-vtg=" + nmea.vtg);
   DebugMsg(message := "-gga=" + nmea.gga);
   DebugMsg(message := "-gsa=" + nmea.gsa);
   ...
END;

END_PROGRAM;
```

4.2.18.12 gpsSetSBAS (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2i, CX1, SX1, MX2 turbo/encore/warp, NX-200, LX2
Firmware version: 1.21 / 1.40.00

This function is used to enable/disable the SBAS positioning assistance.
The default state for the MX2 turbo/encore/warp is *enabled*, for other devices it is *disabled*.

SBAS means "Satellite Based Augmentation System" and is a system that supports wide-area or regional augmentation through the use of additional satellite broadcast messages.
The messages contain correction data which improves the accuracy of the GPS position that is returned.

The following SBAS systems are in operation globally:

- WAAS (Wide Area Augmentation System) covering North America.
- EGNOS (European Geostationary Navigation Overlay System) covering Europe and Africa.
- MSAS (Multi-Functional Satellite Augmentation System) covering Japan and part of Oceania.

Also see gpsGetSBAS and gpsFix for additional information.

Before calling this function the GPS receiver must be turned ON by a call to gpsPower().

When the device is reset the SBAS will return to the default state.

Input:

Enable : BOOL
Enables/disables SBAS.

Returns: BOOL

TRUE: SBAS assisted positioning enabled/disabled.
FALSE: Failed to enable/disable SBAS assisted positioning. The GPS receiver may be turned off.

Declaration:

```
FUNCTION gpsSetSBAS : BOOL;
VAR_INPUT
   enable : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Start using SBAS
   gpsSetSBAS(enable := ON);
   ...
END;

END_PROGRAM;
```

4.2.18.13 gpsGetSBAS (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2i, CX1, SX1, MX2 turbo/encore/warp, NX-200, LX2
Firmware version: 1.21 / 1.40.00

This returns information about whether the SBAS positioning assistance is enabled or disabled. Also see gpsSetSBAS.

Input:
None.

Returns: BOOL
TRUE: SBAS assisted positioning enabled.
FALSE: SBAS assisted positioning disabled.

Declaration:
```
FUNCTION gpsGetSBAS : BOOL;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Is SBAS enabled?
   IF gpsGetSBAS() THEN
      // SBAS enabled
      ...
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.18.14 gpsPPPStatus (Function)

Architecture: NX32, NX32L
Device support: MX2 turbo/encore/warp, NX-200 with PPP support
Firmware version: 4.10

Returns the number of satellites using Precise Point Positioning (PPP) for increased accuracy.

Input:
None.

Returns: INT
>0	Number of satellites using PPP.
0	PPP is not used or unsupported (Please consult the technical manual for the specific device in question).

Declaration:

```
FUNCTION gpsPPPStatus : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Retrieve PPP status
   DebugFmt(message:="PPP status: \1", v1:= gpsPPPStatus());
   ...
END;

END_PROGRAM;
```

4.2.18.15 gpsSetSpeedThreshold (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2i, CX1, SX1, MX2 turbo/encore/warp, NX-200, LX2
Firmware version: 2.60 / 1.40.00

This function sets the speed threshold that is used by SuperGPS-based devices to determine a new position.
If the speed is slower than the threshold, the position will stay fixed. This is to avoid undesired position drift when standing still.

Input:

threshold : SINT (*Default: 3*)
0 = None (Pedestrian. 0 m/s).
1 = Low (Sport / Bike. 0.2 m/s).
2 = Medium (Low speed vehicles. 0.8 m/s).
3 = High (Vehicles. 1.5 m/s).

Returns: BOOL

TRUE: Speed threshold is changed.
FALSE: Failed to change speed threshold. The device may not have SuperGPS.

Declaration:

```
FUNCTION gpsSetSpeedThreshold : BOOL;
VAR_INPUT
   threshold : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

gpsPower(power := ON);
gpsSetSpeedThreshold(threshold := 2);
..
```

```
BEGIN
END;
END_PROGRAM;
```

4.2.18.16 gpsPositionToUtm (Function)

```
Architecture:       X32 / NX32 / NX32L
Device support:     All
Firmware version:   2.80 / 1.00.00
```

This function is used for converting a position from RTCU format to Universal Transverse Mercator (UTM) format.

For the calculation, the WGS-84 ellipsoid is used to model the Earth in the UTM coordinate system.

Input:

lat : DINT [-80000000 .. 84000000]
Negative is south (ddmm.mmmm (multiplied by 10000)).

lon : DINT [-2147483648 .. 2147483647]
Negative is west (dddmm.mmmm (multiplied by 10000)).

Output:

zone : INT
UTM zone number.
>0 The position is on the northern hemisphere.
<0 The position is on the southern hemisphere.

east : DINT
UTM easting coordinate in meters east of the central meridian of the zone.

north : DINT
UTM northing coordinate in meters north of the Equator on the northern hemisphere. On the southern hemisphere, it is 10,000 km minus the distance to the Equator.

Return:

0 Success.
-1 Invalid input.

Declaration:

```
FUNCTION gpsPositionToUtm : INT;
VAR_INPUT
   lat    : DINT;
   lon    : DINT;
   zone   : ACCESS INT;
   east   : ACCESS DINT;
   north  : ACCESS DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM sample;
VAR
   gps         : gpsFix; // Receives information from GPS module
   report      : TON;
   zone        : INT;
   east,north  : DINT;
END_VAR;

// Activate power to the GPS module
gpsPower(power:=ON);
BEGIN
   gps();
   IF gps.mode >= 3 THEN
      report(trig := NOT report.q);
   ELSE
      report(trig := FALSE, pt := 0); // report first gps fix fast
   END_IF;
   IF report.q THEN
      report.pt := 2000; // report each 2nd second

      DebugMsg(message := "GPS position:");
      DebugFmt(message := "   latitude = \4", v4 := gps.latitude);
      DebugFmt(message := "   longitude = \4", v4 := gps.longitude);

      gpsPositionToUtm(
         lat := gps.latitude, lon := gps.longitude,
         zone := zone, east := east, north := north);

      DebugMsg(message := "UTM position:");
      DebugFmt(message := "   zone  = \1", v1 := zone);
      DebugFmt(message := "   east  = \4", v4 := east);
      DebugFmt(message := "   north = \4", v4 := north);
   ELSE
      Sleep(delay := 500);
   END_IF;
END;
END_PROGRAM;
```

4.2.18.17 gpsUtmToPosition (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 2.80 / 1.00.00

This function is used for converting a position from Universal Transverse Mercator (UTM) format to RTCU format.

For the calculation, the WGS-84 ellipsoid is used to model the Earth in the UTM coordinate system.

Input:

zone : INT [-60 .. 60]
UTM zone number.
>0 The position is on the northern hemisphere.
<0 The position is on the southern hemisphere.

east : DINT [-2147483648 .. 2147483647]
UTM easting coordinate in meters east of the central meridian of the zone.

north : DINT [-2147483648 .. 2147483647]
UTM northing coordinate in meters north of the Equator on the northern hemisphere. On the southern hemisphere, it is 10,000 km minus the distance to the Equator.

Output:

lat : DINT
Negative is south (ddmm.mmmm (multiplied by 10000)).

lon : DINT
Negative is west (dddmm.mmmm (multiplied by 10000)).

Return:

0	Success.
-1	Invalid input.

Declaration:

```
FUNCTION gpsUtmToPosition : INT;
VAR_INPUT
   zone    : INT;
   east    : DINT;
   north   : DINT;
   lat     : ACCESS DINT;
   lon     : ACCESS DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
PROGRAM sample;
VAR
   lat,lon     : DINT;
   zone        : INT;
   east,north  : DINT;
END_VAR;

BEGIN
  ...
  DebugMsg(message := "UTM position:");
  DebugFmt(message := "   zone  = \1", v1 := zone);
  DebugFmt(message := "   east  = \4", v4 := east);
  DebugFmt(message := "   north = \4", v4 := north);

  gpsUtmToPosition(
     zone := zone, east := east, north := north,
     lat := lat, lon := lon);

  DebugMsg(message := "GPS position:");
  DebugFmt(message := "   latitude = \4", v4 := lat);
  DebugFmt(message := "   longitude = \4", v4 := lon);
  ...
END;
END_PROGRAM;
```

4.2.18.18 gpsSemicircleToUtm (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 2.80 / 1.00.00

This function is used for converting a position from semi-circle format to Universal Transverse Mercator (UTM) format.

For the calculation, the WGS-84 ellipsoid is used to model the Earth in the UTM coordinate system.

Input:

lat : DINT [-954437176 .. 1002159035]
The latitude of the position in semi-circle format.
This value is limited to the range matching 80 deg. south to 84 deg. north as UTM works differently at the poles.

lon : DINT [-2147483648 .. 2147483647]
The longitude of the position in semi-circle format.

Output:

zone : INT
UTM zone number.
>0 The position is on the northern hemisphere.
<0 The position is on the southern hemisphere.

east : DINT
UTM easting coordinate in meters east of the central meridian of the zone.

north : DINT
UTM northing coordinate in meters north of the Equator on the northern hemisphere. On the southern hemisphere, it is 10,000 km minus the distance to the Equator.

Return:

0 Success.
-1 Invalid input.

Declaration:

```
FUNCTION gpsSemicircleToUtm : INT;
VAR_INPUT
   lat    : DINT;
   lon    : DINT;
   zone   : ACCESS INT;
   east   : ACCESS DINT;
   north  : ACCESS DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
PROGRAM sample;
VAR
```

```
   lat,lon     : DINT;
   zone        : INT;
   east,north  : DINT;
END_VAR;

BEGIN
  ...
  DebugMsg(message := "Semi-circle position:");
  DebugFmt(message := "   latitude = \4", v4 := lat);
  DebugFmt(message := "   longitude = \4", v4 := lon);

  gpsSemicircleToUtm(
     lat := lat, lon := lon, zone := zone,
     east := east, north := north);

  DebugMsg(message := "UTM position:");
  DebugFmt(message := "   zone  = \1", v1 := zone);
  DebugFmt(message := "   east  = \4", v4 := east);
  DebugFmt(message := "   north = \4", v4 := north);
  ...
END;
END_PROGRAM;
```

4.2.18.19 gpsUtmToSemicircle (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 2.80 / 1.00.00

This function is used for converting a position from Universal Transverse Mercator (UTM) format to semi-circle format - this can be used with e.g. the navigation API.

For the calculation, the WGS-84 ellipsoid is used to model the Earth in the UTM coordinate system.

Input:

zone : INT [-60 .. 60]
UTM zone number.
>0 The position is on the northern hemisphere.
<0 The position is on the southern hemisphere.

east : DINT
UTM easting coordinate in meters east of the central meridian of the zone.

north : DINT
UTM northing coordinate in meters north of the Equator on the northern hemisphere. On the southern hemisphere, it is 10,000 km minus the distance to the Equator.

Output:

lat : DINT
The latitude of the position in semi-circle format.

lon : DINT
The longitude of the position in semi-circle format.

Return:

0	Success.
-1	Invalid input.

Declaration:

```
FUNCTION gpsUtmToSemicircle : INT;
VAR_INPUT
   zone  : INT;
   east  : DINT;
   north : DINT;
   lat   : ACCESS DINT;
   lon   : ACCESS DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
PROGRAM sample;
VAR
   lat,lon     : DINT;
   zone        : INT;
   east,north  : DINT;
END_VAR;

BEGIN
  ...
  DebugMsg(message := "UTM position:");
  DebugFmt(message := "   zone  = \1", v1 := zone);
  DebugFmt(message := "   east  = \4", v4 := east);
  DebugFmt(message := "   north = \4", v4 := north);

  gpsUtmToSemicircle(
     zone := zone, east := east, north := north,
     lat := lat, lon := lon);

  DebugMsg(message := "Semi-circle position:");
  DebugFmt(message := "   latitude = \4", v4 := lat);
  DebugFmt(message := "   longitude = \4", v4 := lon);
  ...
END;
END_PROGRAM;
```

4.2.18.20 gpsUpdateFreq (Function)

Architecture: **X32 / NX32 / NX32L**
Device support: **MX2i (not available on the MX2), CX1, SX1, MX2 turbo/encore/warp, NX-200, LX2**
Firmware version: **3.10 / 1.40.00**

This function sets the frequency at which new positions can be read from a SuperGPS-based device.

Using an update frequency of more than 1 position per. second requires that the system speed is minimum set to *Fast* (24 Mhz).

Note:
When using gpsEnableNMEA it must be ensured that the baud-rate used on the serial port is set sufficiently high to cope with the update frequency.
Operating at mode=2 (4 positions per. second) will only forward data to a connected NMP in every 2nd call to gpsFix.

This function can not be used if the GPS in in low power mode.

Input:

mode : SINT (*Default: 0*)
0 = Standard mode (max. 1 position per second).
1 = Fast mode (max. 2 positions per second).
2 = Fastest mode (max. 4 positions per second).

Returns: INT

0 - Success.
1 - Frequency update is not supported.
2 - GPS power is OFF.
3 - Illegal parameter.
4 - System speed is not supported or gpsPowerLP is used.
5 - General error.

Declaration:

```
FUNCTION gpsUpdateFreq : INT;
VAR_INPUT
   mode : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

gpsPower(power := ON);
gpsUpdateFreq(mode := 2);
..

BEGIN
END;
END_PROGRAM;
```

4.2.19 gsm: GSM functions

4.2.19.1 GSM: GSM functions

The GSM / Cellular functions allow the program to interact with a cellular engine in the device.

• gsmAnswer	Answers an incoming voice call.
• gsmOffHook	Returns whether a voice session is active.
• gsmHangup	Terminates an active voice call.
• gsmIncomingCall	Checks for incoming voice calls.
• gsmIncomingSMS	Checks for incoming SMS messages.
• gsmConnected	Checks connection to the GSM network.
• gsmSendSMS	Sends an SMS message.
• gsmMakeCall	Makes a voice call.
• gsmMakeCallX	Makes a voice call with extended capabilities.
• gsmPower	Controls power to the GSM module.
• gsmPowerLP	Controls low power to the GSM module.
• gsmSetAntennaMode	Switches between internal and external antenna.
• gsmGetAntennaMode	Returns which antenna is selected and which antenna is used.
• gsmSignalLevel	Returns the signal level for the GSM connection.
• gsmSendPDU	Sends an SMS message as a binary PDU message.
• gsmIncomingPDU	Checks for incoming SMS binary PDU messages.
• gsmSetListOfCallers	Sets the list of phone numbers allowed to establish a CSD session.
• gsmGetListOfCallers	Gets the list of phone numbers allowed to establish a CSD session.
• gsmGetIMEI	Gets the IMEI number of the GSM module.
• gsmGetIMSI	Gets the IMSI number of the SIM card.
• gsmGetICCID	Gets the ICCID number of the SIM card.
• gsmSendDTMF	Sends a DTMF string.
• gsmGetProviderList	Gets a list of GSM providers.
• gsmSetProvider	Sets the GSM provider to use.
• gsmSetPIN	Sets the pin code to use for the SIM card.
• gsmSetSMSSCN	Sets SMS Service Center Address.
• gsmHeadset	Enables a connected headset.
• gsmHandsfree	Enables a connected speaker/microphone handsfree setup.
• gsmHandsfreeVolume	Controls the volume of the handsfree setup.
• gsmGetLAC	Gets the LAC number of the GSM module.
• gsmGetCellID	Gets the Cell ID number of the GSM module.
• gsmGetStatus	Returns the status for the GSM network connection.
• gsmGetCurrentProvider	Gets the PLMN number of the current GSM provider.
• gsmGetHomeProvider	Gets the PLMN number of the home GSM provider.
• gsmGetNetworkType	Gets the type of the network (2G/3G)
• gsmSetNetworkType	Sets the type of network to use.
• gsmSIMPresent	Determines whether the SIM card is present.
• gsmSIMLocked	Determines whether the SIM card is locked in place.
• gsmModemMode	Enters dedicated modem mode where the device can be used as modem.
• gsmSendFlashSMS	Sends a Flash SMS message
• gsmSimSetPIN	Enables/disables the PIN code in the SIM card.
• gsmSelectSIM	Selects the SIM card reader on devices with multiple SIM card readers.

• gsmNetworkTime	Retrieves the current time from the GSM network.
• gsmSetWakeup	Configure GSM module as wake-up source.
• gsmVoLTEEnable	Enable support for VoLTE.
• gsmVoLTEStatus	Check the status of VoLTE.

4.2.19.2 gsmAnswer (Function)

Architecture: X32 / NX32 / NX32L
Device support: All with voice/DTMF capability
Firmware version: 1.00 / 1.00.00

This answers an incoming call on the GSM module. After this call, the GSM module is in session with the calling party.

Input:
None.

Returns:
None.

Declaration:

```
FUNCTION gsmAnswer;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

VAR
   incoming : gsmIncomingCall;
END_VAR;

// Turn on power to the GSM module, use no pin code
gsmPower(power := TRUE);

BEGIN
   ...
   // Check for incoming calls
   IF incoming.status = 1 THEN
      // Somebody is calling us with a callerid
      // Answer incoming call
      gsmAnswer();
      // Play the "Hello" message
      voiceTalk(message := "Hello");
      // Hangup the phone
      gsmHangup();
      ...
   END_IF;
END;

END_PROGRAM;
```

4.2.19.3 gsmConnected (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This function checks whether the GSM module is connected to a GSM base station or not.

Input:
None.

Returns: BOOL
TRUE: Connected to a base station.
FALSE: Not connected to a base station.

Declaration:
```
FUNCTION gsmConnected : BOOL;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Check for successful connection to a GSM base station
   IF gsmConnected() THEN
      // RTCU is connected to a base station
      ...
   ELSE
      // RTCU is NOT connected to a base station
      ...
   END_IF;
END;

END_PROGRAM;
```

4.2.19.4 gsmHangup (Function)

Architecture: X32 / NX32 / NX32L
Device support: All with voice/DTMF capability
Firmware version: 1.00 / 1.00.00

This closes an active connection from the GSM module. After this call, the GSM module is no longer in session.
If the call has not been answered with gsmAnswer, the call will be rejected.

Input:
None.

Returns:
None.

Declaration:
```
FUNCTION gsmHangup;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

VAR
```

```
   incoming : gsmIncomingCall;
END_VAR;

// Turn on power to the GSM module, use no pin code
gsmPower(power := TRUE);

BEGIN
   ...
   // Check for incoming calls
   IF incoming.status = 1 THEN
      // Somebody is calling us with a callerid
      // Answer incoming call
      gsmAnswer();
      // Play the "Hello" message
      voiceTalk(message := "Hello");
      // Hangup the phone
      gsmHangup();
      ...
   END_IF;
END;

END_PROGRAM;
```

4.2.19.5 gsmIncomingCall (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: All with voice/DTMF capability
Firmware version: 1.00 / 1.00.00

Use this function block to check if there is an incoming voice call on the GSM module.

gsmIncomingCall() indicates that an incoming call is pending. Technically it will be 1 or 2 when RING has been received from the GSM module, and it will be cleared when the VPL application has called gsmIncomingCall(). To use it, gsmIncomingCall() should be called periodically (frequency will dictate the answer delay period), and when it returns 1 or 2, the call may be answered.

Input:

None.

Output:

status : INT (0..2)

- 0 If no call.
- 1 If call contains a caller id ("number" contains caller id).
- 2 If call does not contain a caller id.

phonenumber : STRING
Caller id of the caller (if status is 1).

Declaration:

```
FUNCTION_BLOCK gsmIncomingCall;
VAR_OUTPUT
   status      : INT;
   phonenumber : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
```

```
PROGRAM test;

VAR
   incoming : gsmIncomingCall;
END_VAR;

// Turn on power to the GSM module, use no pin code
gsmPower(power := TRUE);

BEGIN
   incoming();
   ...
   // Check for incoming calls
   IF incoming.status = 1 THEN
      // Somebody is calling us with a callerid
      // Answer incoming call
      gsmAnswer();
      // Play the "Hello" message
      voiceTalk(message := "Hello");
      // Hangup the phone
      gsmHangup();
      ...
   END_IF;
END;

END_PROGRAM;
```

4.2.19.6 gsmIncomingSMS (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

Use this function block to check whether there is an incoming SMS message from the cellular network or over the RTCU Communication Hub.

The device has a buffer where SMS and PDU messages are stored. To prevent loss of any messages, this buffer can only contain one message at a time.
Since this buffer is shared between SMS and PDU messages, it is possible if a PDU type SMS message is received and no instance of the gsmIncomingPDU is present in the application that this incoming PDU message will block the possibility of other SMS messages arriving. Because of this, it is recommended to have an instance of the gsmIncomingPDU even if the application is not normally expecting PDU messages.
The buffer will not be filled until either gsmIncomingSMS or gsmIncomingPDU is called.

The GSM network handles the concatenated messages by splitting it up into smaller pieces, which is then send as a standard SMS messages with information on how to assemble the original text again.
Since the concatenated text can be larger than the maximum size of a string, the gsmIncomingSMS returns the smaller pieces with the necessary information to assemble the original text again.

An SMS message sent from the RTCU IDE (or any other RACP-compliant peer) that uses a cable or data connection will be delivered to the application with the "phone number" set to "9999". See also gsmSendSMS.

Messages received through the RTCU Communication Hub will have the "phone number" set to the node number of the originating node prefixed with "@" (e.g. "@3917321112").

Note: the phone book on the used SIM card must be empty. If a caller that is present in the phone book sends an SMS message, this message will not be delivered to the program.

Input:
None.

Output:
status : INT (0..2)
- 0 If no message.
- 1 If message contains a caller id ("number" contains caller id).
- 2 If message does not contain a caller id.

phonenumber : STRING
Caller id of the sender (if status is 1).

If the number is prefixed with a "@", the message is received through the RTCU Communication Hub and the number after the "@" character is the originating node number (only supported in network-enabled devices).

The "phone number" will be set to "9999" for messages sent from the RTCU IDE (or any other RACP-compliant peer).

message : STRING
Message from the sender (only if status is 1 or 2).

reference : DINT
The reference ID that identifies which message the received message fragment is part of. (only if total is more than zero)

number : INT
The part number in the sequence of the received message fragment. (only if total is more than zero)

total : INT
The total number of parts in the message. (only if status is 1 or 2)

Declaration:
```
FUNCTION_BLOCK gsmIncomingSMS;
VAR_OUTPUT
   status      : INT;
   phonenumber : STRING;
   message     : STRING;
   reference   : DINT;
   number      : INT;
   total       : INT;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;

VAR
   incoming : gsmIncomingSMS;
END_VAR;

BEGIN
   incoming();
   ...
   // Check for incoming calls
```

```
   IF incoming.status = 1 THEN
      // Somebody with a callerid has sent us a SMS message
      // Set the current write position to 1,1 (leftmost, first row)
      displayXY(x:=1, y:=1);
      // Print the received callerid in the display
      displayString(message:=incoming.phonenumber);
      // Set the current write position to 1,2 (leftmost, second row)
      displayXY(x:=1, y:=2);
      // Print the received SMS message in the display
      displayString(message:=incoming.message);
      ...
   END_IF;
END;

END_PROGRAM;
```

4.2.19.7 gsmMakeCall (Function)

Architecture: X32 / NX32 / NX32L
Device support: All with voice/DTMF capability
Firmware version: 1.00 / 1.00.00

This establishes a voice connection to a telephone. The function dials the number specified and waits for either completion, timeout, or an error.
The number can contain "-" and " " (dash and space) as part of the number to call. "Timeout" specifies how many seconds the RTCU should wait for the other party to pick up the phone before the call returns FALSE.

Input:

phonenumber : STRING (max length is 20, excluding " " and "-")
Number to call.

timeout : SINT (1..65, *Default:* 45)
Number of seconds to wait for answer before returning FALSE.

Returns: BOOL

TRUE: Call was successful. Connection between GSM module and the call recipient established.
FALSE: Connection could not be established (call recipient busy etc.)

Declaration:

```
FUNCTION gsmMakeCall : BOOL;
VAR_INPUT
   phonenumber : STRING;
   timeout     : SINT := 45;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Turn on power to the GSM module
gsmPower(power := TRUE);

BEGIN
   ...
   // Make a call, wait maximum 10 seconds for an answer
   IF gsmMakeCall(phonenumber := "+44 22 33 44 55", timeout:=10) THEN
      // Successful, we have established a connection
```

```
      // Play the "Hello" message
      voiceTalk(message := "Hello");
      // Hangup the phone
      gsmHangup();
      ...
   END_IF;
END;

END_PROGRAM;
```

4.2.19.8 gsmMakeCallX (Function)

Architecture: X32 / NX32 / NX32L
Device support: All with voice/DTMF capability
Firmware version: 1.00 / 1.00.00

The gsmMakeCallX function is an extended version of the gsmMakeCall function that will return more detailed information about the result of the call. It will also offer the possibility of suppressing the caller identification of the device for the party called.

The function dials the number specified and waits for either completion, timeout, or an error.
The number can contain "-" and " " (dash and space) as part of the number to call.
"Timeout" specifies how many seconds the RTCU should wait for the other party to pick up the phone.

Input:

phonenumber : STRING (max length is 20, excluding " " and "-")
Number to call.

CLIR : BOOL (*Default:* FALSE)
CLIR = **C**alling **L**ine **I**dentification **R**estriction.
When TRUE, the device will suppress presentation of the telephone number to the called party.

timeout : SINT (1..65, *Default:* 45)
Number of seconds to wait for an answer.

Returns: INT

0 - Call was successful. Connection between GSM module and the call recipient established.
1 - No dial tone (network occupied).
2 - Busy (call recipient is busy).
3 - No answer (call recipient is not answering).
4 - No carrier (network problem or voice call already active).
5 - Not connected to GSM network.
6 - Call aborted by caller (gsmHangup() called).
7 - GSM Module occupied by data call.
8 - GSM Module not present on the device.
9 - Unspecified error.

Declaration:

```
FUNCTION gsmMakeCallX : INT;
VAR_INPUT
   phonenumber : STRING;
   CLIR        : BOOL := FALSE;
   timeout     : SINT := 45;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Turn on power to the GSM module
gsmPower(power := TRUE);

BEGIN
   ...
   // Make a call, wait maximum 10 seconds for an answer
   IF gsmMakeCallX(phonenumber := "+44 22 33 44 55", CLIR := TRUE,
timeout:=10) = 0 THEN
      // Successful, we have established a connection
      // Play the "Hello" message
      voiceTalk(message := "Hello");
      // Hangup the phone
      gsmHangup();
      ...
   END_IF;
END;

END_PROGRAM;
```

4.2.19.9 gsmSendDTMF (Function)

Architecture: X32 / NX32 / NX32L
Device support: All with voice/DTMF capability
Firmware version: 1.00 / 1.00.00

This sends a string as DTMF tones. This has the same effect as pressing the touch-tones on a "normal" phone. The maximum length of the string is 29 characters.

Input:

number : STRING (0..9, A,B,C,D,#,*)
DTMF string to send (max 29 characters).

Returns: BOOL

TRUE: Function was successful. String was sent.
FALSE: No success (not connected to base station etc.)

Declaration:

```
FUNCTION gsmSendDTMF : BOOL;
VAR_INPUT
   number : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Turn on power to the GSM module
gsmPower(power := TRUE);

BEGIN
   ...
   // Make a call
```

```
   IF gsmMakeCall(phonenumber := "+44 22 33 44 55") THEN
      // Successful, we have established a connection
      // Send a number
      gsmSendDTMF(number := "1234");
      // Hangup the phone
      gsmHangup();
      ...
   END_IF;
END;

END_PROGRAM;
```

4.2.19.10 gsmOffHook (Function)

Architecture: X32 / NX32 / NX32L
Device support: All with voice/DTMF capability
Firmware version: 1.00 / 1.00.00

This function checks whether the GSM module is in active conversation with another phone.

Input:

None.

Returns: BOOL

TRUE: In session with another telephone.
FALSE: Not in session with another telephone.

Declaration:

```
FUNCTION gsmOffHook : BOOL;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Turn on power to the GSM module, use no pin code
gsmPower(power := TRUE);

BEGIN
   ...
   // Check if GSM module is in session with another telephone
   IF gsmOffHook() = TRUE THEN
      // We are connected to another telephone
      ...
   ELSE
      // No session active
      ...
   END_IF;
END;

END_PROGRAM;
```

4.2.19.11 gsmPower (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This function controls the power to the on-board cellular engine. This function must be called before any functions depending on an active cellular connection is used - for example gprsOpen().

Note: when the devices faults (see fault log), the cellular engine will automatically be turned on, so that a remote connection can be established.
The device will automatically (if configured) connect to the RTCU Communication Hub in this situation.

Input:

power : BOOL
TRUE: Turns on the power.
FALSE: Turns off the power.

hs : BOOL
TRUE: Use the high-speed link.
FALSE: Use the standard speed link (default)
Note: *This parameter is only supported on the RTCU NX-900 series. Please refer to the technical manual for details.*

Returns: INT

0	Ok.
1	Modem is busy with: CSD session, mobile network session, or an active voice (power off not allowed in this case).
2	Modem is blocked because of network/modem problems. Recovery of the lock-up situation is already initiated.
8	System speed is not supported.
9	Modem is already operating in LP mode.
10	Modem is operating in dedicated modem mode using gsmModemMode.
11	Non-recoverable error in communication with the GSM module.
14	Cannot switch high speed mode when powered on.
15	High speed connection is not accessible.

Declaration:

```
FUNCTION gsmPower : INT;
VAR_INPUT
   power : BOOL;
   hs    : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Turn on power to the GSM module
gsmPower(power := TRUE);

BEGIN
```

```
   ...
END;

END_PROGRAM;
```

4.2.19.112 gsmPowerLP (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.02 / 1.00.00

This function controls the power supply to the on-board cellular engine. By using this function, the VPL program can switch on and off the cellular engine to save power. Unlike gsmPower, this function is operating the engine on a slower communication speed - thus enabling the use of advanced power management functions such as pmSetSpeed.
The bandwidth when using a mobile network will be lower when the engine is switched on by using this function rather than gsmPower.

Input:

power : BOOL
TRUE: Turns on the power.
FALSE: Turns off the power.

Returns: INT

Same return codes as gsmPower.

Declaration:

```
FUNCTION gsmPowerLP : INT;
VAR_INPUT
   power : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Turn on power to the GSM module
gsmPowerLP(power := TRUE);

BEGIN
   ...
END;

END_PROGRAM;
```

4.2.19.113 gsmSetAntennaMode (Function)

Architecture: X32 / NX32
Device support: SX1, AX9i, AX9 turbo/encore
Firmware version: 2.70

This function switches between the on-board internal or an optional external GSM antenna.

The selection made by calling this function will remain in effect (persistent through reset/power down) until changed explicitly by a new call to gsmSetAntennaMode.

The gsmGetAntennaMode function can be used to determine which antenna is used.

Input:

mode : SINT

1 Switches to internal antenna.
2 Switches to external antenna.
3 AX9i/AX9 turbo/encore: Selected antenna is determined by DIP-switch 4 (on=internal,off=external. Please refer to technical manual for details).

Returns: INT

0 Successful.
1 Illegal mode.

Declaration:

```
FUNCTION gsmSetAntennaMode : INT;
VAR_INPUT
   mode : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Switch to external antenna. Only required to be set once.
gsmSetAntennaMode(mode := 2);

BEGIN
   ...
END;

END_PROGRAM;
```

4.2.19.14 gsmGetAntennaMode (Function)

Architecture: X32 / NX32
Device support: SX1, AX9i, AX9 turbo/encore
Firmware version: 4.40

This function returns which antenna is used, and the current selection mode set by the gsmSetAntennaMode function.

Input:

None.

Output:

mode : SINT

1 The internal antenna is selected.
2 The external antenna is selected.
3 AX9i/AX9 turbo/encore: The active antenna is determined by DIP-switch 4.

active : SINT

1 The internal antenna is active.
2 The external antenna is active.

Returns: **INT**

0 Not supported
1 Successful.

Declaration:

```
FUNCTION gsmGetAntennaMode : INT;
VAR_INPUT
   mode   : ACCESS SINT;
   active : ACCESS SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   mode : SINT;
   used : SINT;
   str  : STRING;
END_VAR;

BEGIN
   ...
   IF gsmGetAntennaMode(mode := mode, active := used) = 0 THEN
      str := "GSM Antenna: Not supported";
   ELSE
      str := "GSM Antenna: ";
      CASE mode OF
         1: str := str + "mode=Internal";
         2: str := str + "mode=External";
         3: str := str + "mode=Dip switch";
      END_CASE;
      CASE used OF
         1: str := str + ", used=Internal";
         2: str := str + ", used=External";
      END_CASE;
   END_IF;
   DebugMsg(message := str);
   ...
END;
END_PROGRAM;
```

4.2.19.15 gsmSendSMS (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This function sends an SMS message by using the GSM module. It will return status of the send command (see below).
The number can contain "-" and " " (dash and space) as part of the number to send the message to.

Please note that you should always check the return value from this function. There are a number of reasons why this function could fail such as too much traffic on the GSM network etc., so always check the return value, and based on that it is possible to make a strategy for resending a message.

This function can also send GSM network SMS messages during an active mobile network session. The mobile network session will simply be suspended during the transmission of the SMS message and automatically be resumed after the operation has ended.

Input:

phonenumber : STRING (max length is 20, excluding " " and "-")
Number to send the SMS message to (if "9999" is used, the message will be sent to a connected PC - either through a cable connection or via an active data connection).

If the number is prefixed with an "@", the message is sent through the RTCU Communication Hub to the specified node number after the "@" character
(only supported on network-enabled devices).

In case the device is not connected to the GSM network, the return value will be 1.

message : STRING
The actual message to send (maximum of 160 characters).

Returns: INT

For GSM SMS messages:

0	If successful.
1	If general error.
2	If CMS/CME error (error on GSM net).
3	If not connected to a base station.
4	If GSM module is not powered on (see gsmPower()).
5	If GSM module is busy with a data call.

For RTCU Communication Hub VSMS messages:

0	If successful.
1	If the receiver is not connected to the GW.
2	If timeout delivering message to the receiver.
3	If not connected to a GW.
5	If the message was rejected by the receiver.

Declaration:

```
FUNCTION gsmSendSMS : INT;
VAR_INPUT
   phonenumber : STRING;
   message     : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Turn on power to the GSM module
gsmPower(power := TRUE);

BEGIN
   ...
   // Note: we are NOT checking the return code from the gsmSendSMS function
```

```
in this example
   gsmSendSMS(phonenumber := "+44 12 34 56 789", message := "This is a test
message sent using SMS");
   ...
END;

END_PROGRAM;
```

4.2.19.16 gsmSignalLevel (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This function returns the signal strength of the GSM base station it is received with. The returned value will either be the real signal strength (expressed in dBm, between -113 and -51) or as a percentage (between 0 and 100). If the input parameter "dbm" is set to false (this is the default), the function will return the signal strength as a percentage. Please note that the signal level is logarithmic in nature and that the value in percentage is just a linear representation of the real signal level from -113 to -51 dBm. 10 dB (which is 10 times the RF power) of extra signal will only give an extra percentage reading of 16.

It will return 0 if the GSM module is powered off or if the GSM module is temporarily busy

Input:

dbm : BOOL ***(default: FALSE)***

Returns: SINT (0..100 or -113..-51)

Signal level of base station (0 if the GSM module is powered off or if the GSM module is temporarily busy).

Declaration:

```
FUNCTION gsmSignalLevel : SINT;
VAR_INPUT
   dbm : BOOL := FALSE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   level : SINT;
END_VAR;

BEGIN
   ...
   // Get signallevel in % (from 0 to 100)
   level := gsmSignalLevel();
   ...
END;

END_PROGRAM;
```

4.2.19.17 gsmBER (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This function returns the Bit Error Rate in the communication with the GSM base station.
To check the bit error rate there must be a call in progress to obtain realistic values. If no call is set up, there is no BER to be determined. In this case, the indicated value may be 0 or 99 depending on the SIM card.
If the GSM is not powered on, the indicated value will be 1.

Input:
None.

Returns: SINT (0..7, 99)
Bit Error Rate (0..7).

Declaration:
```
FUNCTION gsmBER : SINT;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   ber : SINT;
END_VAR;

BEGIN
   ...
   // Get Bit Error Rate
   ber := gsmBER();
   ...
END;

END_PROGRAM;
```

4.2.19.18 gsmSendPDU (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This function sends an SMS message through the GSM network. It will return the status of the send command (see below).
The number can contain "-" and " " (dash and space) as part of the number to send the message to.
Unlike the gsmSendSMS function, which can only send simple text messages, the gsmSendPDU can send binary messages.

Please note that you should always check the return value from this function. There are a number of reasons why this function could fail such as too much traffic on the GSM network etc., so

always check the return value, and based on that it is possible to make a strategy for resending a message.

This function can also send GSM network SMS messages during an active mobile network session. The mobile network session will simply be suspended during the transmission of the SMS message and automatically be resumed after the operation has ended.

Input:

phonenumber : STRING (max length is 20, excluding " " and "-")
Number to send the SMS message to (if "9999" is used, the message will be sent to a connected PC - either through a cable connection or via an active data connection).

If the number is prefixed with an "@", the message is sent through the RTCU Communication Hub to the specified node number after the "@" character
(only supported on network-enabled devices).

In case the device is not connected to the GSM network, the return value will be 1.

message : PTR
The address of the message to send.

length : INT
Length of message to send in bytes (maximum of 140 bytes).

dcs : SINT
Data Coding Scheme to use, typically 16#F5. Please consult the appropriate documentation for this field.

Returns: INT

For GSM SMS messages:

0 If successful.
1 If general error.
2 If CMS/CME error (error on GSM net).
3 If not connected to a base station.
4 If GSM module is not powered on (see gsmPower()).
5 If GSM module is busy with data call.

For RTCU Communication Hub VSMS messages:

0 If successful.
1 If the receiver is not connected to the GW.
2 If timeout delivering message to the receiver.
3 If not connected to a GW.
5 If the message was rejected by the receiver.

Declaration:

```
FUNCTION gsmSendPDU : INT;
VAR_INPUT
   phonenumber : STRING;
   message     : PTR;
   length      : INT;
   dcs         : SINT := -11; // Equals 16#F5
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
```

```
PROGRAM test;
VAR
   PDUBuffer : ARRAY[0..31] OF SINT;
END_VAR;

// Turn on power to the GSM module
gsmPower(power := TRUE);

BEGIN
   ...
   // Note: we are NOT checking the return code from the gsmSendPDU function
in this example
   gsmSendPDU(phonenumber := "+44 12 34 56 789", message := ADDR(PDUBuffer),
length:=10, dcs:=SINT(16#F5));
   ...
END;

END_PROGRAM;
```

4.2.19.19 gsmIncomingPDU (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This function block can be used to check whether there is an incoming SMS PDU binary message from the cellular network or over the RTCU Communication Hub.

The device has a buffer where SMS and PDU messages are stored. To prevent loss of any messages, this buffer can only contain one message at a time.
Since this buffer is shared between SMS and PDU messages, it is possible if an SMS message is received and no instance of the gsmIncomingSMS is present in the application that this incoming SMS message will block the possibility of other other PDU messages arriving. Because of this, it is recommended to have an instance of the gsmIncomingSMS even if the application is not normally expecting SMS messages.

The buffer will not be filled until either gsmIncomingSMS or gsmIncomingPDU is called.

A PDU message sent from the RTCU IDE (or any other RACP-compliant peer) that uses a cable or data connection will be delivered to the application with the "phone number" set to "9999". See also gsmSendPDU.

Messages received through the RTCU Communication Hub will have the "phone number" set to the node number of the originating node prefixed with "@" (e.g. "@3917321112").

Note: the phone book on the used SIM card must be empty. If a caller that is present in the phone book sends an SMS message, this message will not be delivered to the program.

Input:

message : PTR
Address of a variable where the message from the sender should be delivered to (only if status is 1 or 2). Please make sure that the size of the variable is big enough to accommodate the biggest SMS message that can be received which in PDU mode is 140 bytes. This variable must be set before the gsmIncomingPDU instance is called the first time.

Please note that if a TEXT type SMS message is received and no instance of the gsmIncomingSMS is present in the application, this incoming TEXT SMS message will block for other PDU SMS messages to arrive. Because of this, it is a good idea to have an instance of the gsmIncomingSMS, even if your application is not expecting TEXT SMS messages.

Output:

status : INT (0..4)

0	If no message.
1	If message with caller id. ("number" contains caller id).
2	If message without caller id.
3	If the message receive buffer is not set (see message input above).
4	If unsupported data coding scheme is received.

phonenumber : STRING
Caller id of the sender (if status is 1).

If the number is prefixed with an "@", the message is received through the RTCU Communication Hub and the number after the "@" character is the originating node number (only supported on network-enabled devices).

The "phone number" will be set to "9999" for messages sent from the RTCU IDE (or any other RACP-compliant peer).

length : INT
Number of bytes received in the SMS PDU message.

Declaration:

```
FUNCTION_BLOCK gsmIncomingPDU;
VAR_INPUT
   message     : PTR;
END_VAR;
VAR_OUTPUT
   status      : INT;
   phonenumber : STRING;
   length      : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

VAR
   incoming : gsmIncomingPDU;
   buffer   : ARRAY[0..139] OF SINT;
END_VAR;

// Set address BEFORE the 'incoming' is called the first time ! (In this case,
by BEGIN)
incoming.message := ADDR(buffer);

BEGIN
   incoming();
   ...
   // Check for incoming calls with a callerid
   // NOTE: it is not enough to check if the '.status' is bigger than 0, as
the code 3 is an error, see above
   IF incoming.status = 1 THEN
      // Somebody with a callerid has sent us a SMS message
      // Set the current write position to 1,1 (leftmost, first row)
      displayXY(x:=1, y:=1);
      // Print the received callerid in the display
      displayString(message:=incoming.phonenumber);
      // Set the current write position to 1,2 (leftmost, second row)
```

```
      displayXY(x:=1, y:=2);
      // Print the value of the first byte in the received SMS message in the
display
      displayString(message:=strFormat(format:="\1", v1:=buffer[0]));
      ...
   END_IF;

END;

END_PROGRAM;
```

4.2.19.20 gsmSetListOfCallers (Function)

Architecture: X32 / NX32
Device support: All
Firmware version: 1.00

---

gsmSetListOfCallers will save a list of callers that are allowed to make data call (CSD) to the RTCU device. When an incoming data call is arriving, the caller id of the caller is checked against this list, and if the number is present in the list, the data call will be allowed and answered.

Input:

str : STRING
List of phone numbers that are allowed to make data calls to the device. Each number must be separated with a ",". If the string is empty, i.e. "", no data call is allowed at all. If the string equals "*" (a star), all callers are allowed.

Returns:

None.

Declaration:

```
FUNCTION gsmSetListOfCallers;
VAR_INPUT
   str : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   gsmSetListOfCallers(str:="+441233445,+4544778833");
   // The numbers +441233445 and +4544778833 is allowed to make datacalls
   ...
END;

END_PROGRAM;
```

4.2.19.21 gsmGetListOfCallers (Function)

Architecture: X32 / NX32
Device support: All
Firmware version: 1.00

---

gsmGetListOfCallers will return the list of callers that are allowed to make data call (CSD) to the RTCU device. When an incoming data call is arriving, the caller id of the caller is checked against this list, and if the number is present in the list, the data call will be allowed and answered. See also gsmSetListOfCallers.

Input:
None.

Returns: STRING
List of phone numbers that are allowed to make data call. Each number is separated with a ",".

Declaration:
```
FUNCTION gsmGetListOfCallers : STRING;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   DebugMsg(message := gsmGetListOfCallers());
   // The list of telephonenumbers that are allowed to make datacalls is shown
in the debug window
   ...
END;

END_PROGRAM;
```

4.2.19.22 gsmGetICCID (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 2.20 / 1.00.00

gsmGetICCID will return the ICCID (SIM card) number of the SIM card installed in the RTCU device. If no SIM card is installed, this function will return an empty string.
Note: if gsmPower is not called before this function, an empty string will be returned.

Input:
None.

Returns: STRING
The ICCID number of the GSM module.

Declaration:
```
FUNCTION gsmGetICCID : STRING;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;

gsmPower(power := ON);
```

```
BEGIN
   ...
   DebugMsg(message := gsmGetICCID());
   // The ICCID number is shown in the debug window
   ...
END;

END_PROGRAM;
```

4.2.19.23 gsmGetIMEI (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

---

gsmGetIMEI will return the IMEI number of the GSM module installed in the RTCU device. The IMEI number is a 15 digit unique number that identifies the GSM module.
Note: if gsmPower is not called before this function, an empty string will be returned.

Input:
None.

Returns: STRING
The IMEI number of the GSM module.

Declaration:
```
FUNCTION gsmGetIMEI : STRING;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

gsmPower(power := ON);

BEGIN
   ...
   DebugMsg(message := gsmGetIMEI());
   // The IMEI number is shown in the debug window
   ...
END;

END_PROGRAM;
```

4.2.19.24 gsmGetIMSI (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

---

gsmGetIMSI will return the IMSI number of the SIM card installed in the RTCU device. Only the last 15 digits of the IMSI number will be returned. The IMSI number is a unique number that identifies the SIM card.
If no SIM card is installed, this function will return an empty string.

Note:
***** The first 5 digits in the IMSI number is the Public Land Mobile Network number (PLMN) of the home network.
* If gsmPower is not called before this function, an empty string will be returned.

Input:
None.

Returns: STRING
The IMSI number of the SIM card.

Declaration:
```
FUNCTION gsmGetIMSI : STRING;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;

gsmPower(power := ON);

BEGIN
   ...
   DebugMsg(message := gsmGetIMSI());
   // The IMSI number is shown in the debug window
   ...
END;

END_PROGRAM;
```

4.2.19.25 gsmGetProviderList (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This function block can be used to get a list of available GSM providers.
This function can be called during an active mobile network session.

It is possible to use the gsmSetProvider() function to select a different operator.

Note:
This operation can take a long time to execute - up to 2 minutes.
This function may fail if the GSM module is busy or the GSM coverage is reduced. In this case, it is recommended to wait a minimum of 30 seconds and retry the operation.
It is recommended not to call this function more frequently than every 5 minutes.

Input:
None.

Output:
status : INT (0..1)
0 - If list is available.
1 - If an error occurred while getting the list or during a GSM module power-off.

CurrentLAI : DINT
The GSM Public Land Mobile Network number (PLMN) of the current provider.

LAI : ARRAY[1..16] OF DINT
The GSM PLMN number of the provider "n".

Name : ARRAY[1..16] OF STRING
The name of the provider "n".

State : ARRAY[1..16] OF SINT
The state of the provider "n": 0=unknown, 1=operator available, 2=current operator (registered), 3=forbidden operator.

Declaration:

```
FUNCTION_BLOCK gsmGetProviderList;
VAR_OUTPUT
   Status     : SINT;                  // 0 if successful, 1 if error
   CurrentLAI : DINT;                  // The current providers PLMN number

   // The following information is grouped, index 1 for all arrays is for the
first provider etc.
   LAI        : ARRAY[1..16] OF DINT;   // Providers PLMN number
   Name       : ARRAY[1..16] OF STRING; // Name of provider
   State      : ARRAY[1..16] OF SINT;   // State: 0=unknown, 1=operator
available,  2=current operator (registered), 3=forbidden operator
END_VAR;
```

Example:

```
//-------------------------------------------------------------------------
-
// test.vpl, created 2003-02-26 12:13
//
//-------------------------------------------------------------------------
-
INCLUDE rtcu.inc

VAR
   provider : gsmGetProviderList;
   i        : SINT;
END_VAR;

PROGRAM test;

// The next code will only be executed once after the program starts
gsmPower(power:=TRUE);

// Wait until GSM module registered on network
WHILE NOT gsmConnected() DO Sleep(delay:=1000); END_WHILE;

// Get list of GSM providers
provider();

DebugFmt(message:="Status=\1, Current provider =\4", v1:=provider.status,
v4:=provider.CurrentLAI);
FOR i := 1 TO 16 DO
   IF provider.LAI[i] > 0 THEN
      DebugFmt(message:="Provider=\4, State=\1", v1:=provider.State[i],
v4:=provider.LAI[i]);
      DebugMsg(message:=provider.Name[i]);
   END_IF;
```

```
END_FOR;

BEGIN

END;

END_PROGRAM;
```

4.2.19.26 gsmSetProvider (Function)

```
Architecture:       X32 / NX32 / NX32L
Device support:     All
Firmware version:   1.00 / 1.00.00
```

This function will try to register the device with a specific network provider. Use the gsmGetProviderList() function block to get a list of currently available providers. It should be noted that only providers with State=1 or State=2 (from gsmGetProviderList()), can be used in this function as the provider.

Please note that after calling this function, the gsmConnected() function will signal when (and if) successful registration is accomplished.
If the selected operator is not available, the device will be disconnected until a new operator is selected.

When using the RTCU DX4i pro LTE it is recommended to use the 'ACT' parameter for a successful connection to Cat M1/NB1.

Input:

Provider : DINT
The Public Land Mobile Network number (PLMN) of the network provider. If set to 0, the provider will be selected automatically by the device.
The first 5 digits of the IMSI number contain the PLMN number of the home network (see gsmGetIMSI).

ACT : INT *(only supported from on X32/NX32 firmware V4.70)*
The Access Technology selected:
0 = Default, 1 = GSM, 3=UMTS, 8=LTE, 9=LTE Cat M1, 10=LTE Cat NB1

Please notice that only access technologies supported by the specific product in use can be successfully selected.

Returns: INT

0 - function was successful.
1 - no success or parameter not supported.

Declaration:

```
FUNCTION gsmSetProvider : INT;
VAR_INPUT
   Provider : DINT;
   ACT : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
```

```
// Turn on power to the GSM module
gsmPower(power := TRUE);

BEGIN
   ...
   // Register at new provider:
   gsmSetProvider(Provider := 23802);
   ...
END;

END_PROGRAM;
```

4.2.19.27 gsmSetPIN (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This sets the PIN code to use for accessing the GSM module next time the GSM module is powered on by using the gsmPower() function. An empty string will disable the use of PIN code. Only a PIN code with 4 characters is supported.

The device must be reset for the new PIN code to take effect.

This function does the same as the Device-> Configuration -> GSM Parameters -> PIN code page.

Note: this function does not enable or disable the PIN code on the SIM card - to do this use the gsmSimSetPIN function.

Input:

pin : STRING
PIN code to use for accessing the SIM card.

Returns:

None.

Declaration:

```
FUNCTION gsmSetPIN;
VAR_INPUT
   pin : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Set the pin code to use
gsmSetPIN(pin :="1234"); // Needs only to be set once (The RTCU stores this
information in non-volatile memory)

// Turn on power to the GSM module (it will now use "1234" as the pincode)
gsmPower(power := TRUE);

BEGIN
   ...
END;
```

```
END_PROGRAM;
```

4.2.19.28 gsmSetSMSSCN (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This sets the SMS Service Center Address to use for all further transmission of SMS messages. On most SIM cards this is not necessary as the default SMS Service Center Address is stored on the SIM card by the GSM operator.
This function is only to be called before the GSM module is powered on, i.e. before gsmPower(power:=true) has been called.

Input:

number : STRING
The SMS Service Center Address to use when sending SMS messages from the device.

Returns:

None.

Declaration:

```
FUNCTION gsmSetSMSSCN;
VAR_INPUT
   number : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Set the SMS Service Center Address (number) to use (can be set before
gsmPower() is called)
gsmSetSMSSCN(number :="+4540506070"); // Set the SMS Service Center Address
(number)

// Turn on power to the GSM module
gsmPower(power := TRUE);

BEGIN
   ...
END;

END_PROGRAM;
```

4.2.19.29 gsmHeadset (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2 pro, MX2 turbo, NX-200, NX-400
Firmware version: 1.00 / 1.00.00

This function can enable the headset interface on the device, routing all audio in- and output from the GSM modem through the audio jack to a connected headset.
Voice and DTMF functionality will be disabled when this function is used.

If gsmHandsfree is already enabled or the GSM modem is not powered on, the headset interface can not be enabled.

Input:

enable : BOOL
TRUE: The headset interface will be enabled. RTCU voice and DTMF capability is disabled.
FALSE: The headset interface will be disabled. RTCU voice and DTMF capability is reactivated.

Returns: BOOL

TRUE: Operation was successful.
FALSE: Failed to activate headset.

Declaration:

```
FUNCTION gsmHeadset : BOOL;
VAR_INPUT
   enable : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Turn on power to the GSM module
gsmPower(power := TRUE);

BEGIN
   ...
   // Make a call
   IF gsmMakeCall(phonenumber := "+44 22 33 44 55") THEN
      // Successful, we have established a connection
      // Enable headset for 10 seconds
      gsmHeadset(enable:=TRUE);
      Sleep(delay:=10000);
      // Disable headset
      gsmHeadset(enable:=FALSE);
      // Hangup the phone
      gsmHangup();
      ...
   END_IF;
END;

END_PROGRAM;
```

4.2.19.30 gsmHandsfree (Function)

Architecture: NX32
Device support: MX2 turbo
Firmware version: 4.00

This function can enable the hands-free interface on the device, routing all audio in- and output from the GSM modem through the audio jack to a connected speaker and microphone.
Voice and DTMF functionality will be disabled when this function is used.

If gsmHeadset is already enabled or the GSM modem is not powered on, the hands-free interface can not be enabled.

Input:

enable : BOOL

TRUE: The hands-free interface will be enabled. RTCU voice and DTMF capability is disabled.
FALSE: The hands-free interface will be disabled. RTCU voice and DTMF capability is reactivated.

Returns: INT

0 - Operation was successful.
1 - Audio interface already opened by gsmHeadset.
2 - GSM module not powered on.
3 - General error.
5 - Not supported on this device.

Declaration:

```
FUNCTION gsmHandsfree : INT;
VAR_INPUT
   enable : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Turn on power to the GSM module
gsmPower(power := TRUE);

BEGIN
   ...
   // Make a call
   IF gsmMakeCall(phonenumber := "+44 22 33 44 55") THEN
      // Successful, we have established a connection
      // Set volume to 50 % for speaker and microphone
      gsmHandsfreeVolume( spk := 50, mic := 50);
      // Enable headset for 10 seconds
      gsmHandsfree(enable:=TRUE);
      Sleep(delay:=10000);
      // Disable headset
      gsmHandsfree(enable:=FALSE);
      // Hangup the phone
      gsmHangup();
      ...
   END_IF;
END;

END_PROGRAM;
```

4.2.19.31 gsmHandsfreeVolume (Function)

Architecture: NX32
Device support: MX2 turbo
Firmware version: 4.00

Controls the volume levels used for the handsfree interface. Can be set both before and after opening the hands-free interface with gsmHandsfree.
The volume levels does not necessarily correspond directly to a volume level in the device, but will be mapped to the closest supported level.

Input:

spk: SINT (-1..100) ***Default -1***
Speaker volume. Set to -1 to leave it as it is, 0 to mute.
mic: SINT (-1..100) ***Default -1***
Microphone volume. Set to -1 to leave it as it is, 0 to mute.

Returns: INT

0 - Operation was successful.
1 - Invalid volume
3 - General error
5 - Not supported on this device.

Declaration:

```
FUNCTION gsmHandsfreeVolume : INT;
VAR_INPUT
   spk : SINT := -1;
   mic : SINT := -1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Turn on power to the GSM module
gsmPower(power := TRUE);

BEGIN
   ...
   // Make a call
   IF gsmMakeCall(phonenumber := "+44 22 33 44 55") THEN
      // Successful, we have established a connection
      // Set volume to 50 % for speaker and microphone
      gsmHandsfreeVolume( spk := 50, mic := 50);
      // Enable headset for 10 seconds
      gsmHandsfree(enable:=TRUE);
      Sleep(delay:=10000);
      // Disable headset
      gsmHandsfree(enable:=FALSE);
      // Hangup the phone
      gsmHangup();
      ...
   END_IF;
END;

END_PROGRAM;
```

4.2.19.32 gsmGetLAC (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This function returns the Location Area Code (LAC) of the GSM base station the RTCU device is connected to.

Input:

None.

Returns: **DINT**
GSM LAC.

Declaration:
```
FUNCTION gsmGetLAC : DINT;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   DebugFmt(message := "Current LAC =\4", v4 := gsmGetLAC());
   // The LAC number is shown in the debug window
   ...
END;

END_PROGRAM;
```

4.2.19.33 gsmGetCellID (Function)

Architecture: **X32 / NX32 / NX32L**
Device support: **All**
Firmware version: **1.00 / 1.00.00**

This function returns the Cell ID (CID) of the GSM base station the RTCU device is connected to.

Input:
None.

Returns: **DINT**
GSM CID.

Declaration:
```
FUNCTION gsmGetCellID : DINT;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   DebugFmt(message := "Current CellID =\4", v4 := gsmGetCellID());
   // The Cell ID is shown in the debug window
   ...
END;

END_PROGRAM;
```

4.2.19.34 gsmGetStatus (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This returns the status for the GSM network connection.

Input:
None.

Returns: SINT
0 - gsmPower has not been called.
1 - Not connected to GSM network.
2 - Connected to Home Net.
3 - Searching for provider.
4 - Access to GSM network denied.
5 - Roaming.

Declaration:
```
FUNCTION gsmGetStatus : SINT;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM gsmExample;
VAR
   provider : gsmGetProviderList;
   i        : INT;
END_VAR;

gsmPower(power:=ON);
WHILE NOT gsmConnected() DO Sleep(delay:=100); END_WHILE;

IF gsmGetStatus() = 5 THEN
   provider();
   IF provider.status = 0 THEN
      FOR i := 1 TO 16 DO
         IF provider.LAI[i] > 0 THEN
            IF provider.State[i] = 1 THEN
               gsmSetProvider(Provider:=provider.lai[i]);
               EXIT;
            END_IF;
         END_IF;
      END_FOR;
   END_IF;
END_IF;

BEGIN
END;

END_PROGRAM;
```

4.2.19.35 gsmGetCurrentProvider (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This returns the GSM Public Land Mobile Network number (PLMN) of the current provider.

Input:
None.

Returns: **DINT**
The PLMN of the current provider.
This will return 0 (zero) if not connected to any provider.

Declaration:
```
FUNCTION gsmGetCurrentProvider : DINT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// The next code will only be executed once after the program starts
gsmPower(power:=TRUE);

// Wait until GSM module registered on network
WHILE NOT gsmConnected() DO Sleep(delay:=1000); END_WHILE;

DebugFmt(message:="Current provider =\4", v4:=gsmGetCurrentProvider());

BEGIN

END;

END_PROGRAM;
```

4.2.19.36 gsmGetHomeProvider (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This returns the GSM Public Land Mobile Network number (PLMN) of the home network provider.

Input:
None.

Returns: **DINT**
The PLMN of the home network provider.

Declaration:
```
FUNCTION gsmGetHomeProvider : DINT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// The next code will only be executed once after the program starts
gsmPower(power:=TRUE);

// Wait until GSM module registered on network
WHILE NOT gsmConnected() DO Sleep(delay:=1000); END_WHILE;

DebugFmt(message:="Home provider =\4", v4:=gsmGetHomeProvider());

BEGIN

END;

END_PROGRAM;
```

4.2.19.37 gsmGetNetworkType (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 4.00 / 1.00.00

This returns whether there is a 2G, 3G or 4G(LTE) connection established to the cellular network.

Input:
None.

Returns: INT
0 - Not connected / unknown.
2 - 2G
3 - 3G
4 - 4G (LTE)

Declaration:
```
FUNCTION gsmGetNetworkType : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// The next code will only be executed once after the program starts
gsmPower(power:=TRUE);

// Wait until GSM module registered on network
WHILE NOT gsmConnected() DO Sleep(delay:=1000); END_WHILE;

DebugFmt(message:="Network type = \1", v1 := gsmGetNetworkType());

BEGIN

END;
```

```
END_PROGRAM;
```

4.2.19.38 gsmSetNetworkType (Function)

```
Architecture:       X32 / NX32 / NX32L
Device support:     All
Firmware version:   4.10 / 1.00.00
```

This function selects the preferred service type from the cellular network (2G, 3G or 4G).

If a specific network service type is selected, then the device will not try to connect to other network services.
For example, if the 3G network is selected, then the connection will be lost when entering an area where an only 2G network is present.

To avoid this it is possible to specify the preferred search order (*Not available on all devices*).
The search order determines the order in which the device will try to connect to the different network types.
For example, setting the search order to '2G then 3G' will make the device try to connect to the 2G network if it is present. If the 2G network is not present or disappears, then the device will search for a 3G network.

Note:
The selection made by calling this function will remain in effect (persistent through reset/power down) until changed explicitly by a new call to gsmSetNetworkType.
Not all service types are available on all devices. Please consult the data-sheet.

Input:

type : SINT (default: 0)
The network type to use.

0	Search order.
2	2G network
3	3G network
4	4G (LTE) network

order : SINT (default: 0)
The order used to search for network types. (Is only used when type is set to Search order)

0	Automatic
1	2G then 3G
2	3G then 2G

LTE capable devices only supports the "Automatic" mode and other values will return 1;

Returns: INT

0	Success.
1	Not supported.
2	GSM module is not powered on (see gsmPower()).
3	Illegal network type.
4	Illegal search order.
5	SIM card not present.

Declaration:

```
FUNCTION gsmSetNetworkType : INT;
VAR_INPUT
   type    : SINT := 0;
```

```
   order  : SINT := 0;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Turn on power to the GSM module
gsmPower(power := TRUE);

BEGIN
   ...
   // Set search order to 2G first, then 3G
   gsmSetNetworkType(type := 0, order := 1);
   ...
END;

END_PROGRAM;
```

4.2.19.39 gsmSIMPresent (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This determines whether the SIM card is present

Notice:
On RTCU devices without a physical card detect mechanism, the actual detection of the SIM card is performed in the power-on phase when calling gsmPower. When calling gsmSIMPresent without a former power-on with gsmPower, the SIM card detection will take extended time. It is therefore recommended to turn the GSM module on before calling gsmSIMPresent.

For devices with multiple SIM card readers the status of the currently selected reader will be returned. See gsmSelectSIM().

Input:
None.

Returns: **BOOL**
TRUE: The SIM card is present.
FALSE: The SIM card is not present.

Declaration:

```
FUNCTION gsmSIMPresent : BOOL;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Check for successful connection to a GSM base station
```

```
   IF gsmConnected() THEN
      // RTCU is connected to a base station
      ...
   ELSE
      // RTCU is NOT connected to a base station
      IF gsmSIMPresent() THEN
         // SIM card is inserted
         ...
      ELSE
         // No SIM card in device
         ...
      END_IF;
   END_IF;
END;

END_PROGRAM;
```

4.2.19.40 gsmSIMLocked (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

Depending on the device this function returns whether the SIM card is locked in place or the tray is inserted.

Notice: On devices where the SIM card slot is a lid-type, this function will return TRUE when the SIM card is present (see gsmSIMPresent).

Input:
None.

Returns: BOOL
TRUE: The SIM card is locked in place / The SIM card tray is inserted.
FALSE: The SIM card is not locked in place / The SIM card tray is removed.

Declaration:
```
FUNCTION gsmSIMLocked : BOOL;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Check for successful insertion of SIM card
   IF gsmSIMPresent() AND gsmSIMLocked() THEN
      // OK
      ...
   ELSE
      // Not inserted and locked
      ...
   END_IF;
END;

END_PROGRAM;
```

4.2.19.41 gsmModemMode (Function)

Architecture: X32 / NX32
Device support: MX2 pro, DX4 pro, MX2 turbo
Firmware version: 1.11

The gsmModemMode function allows the use of the internal GSM module as a GSM modem. Using this mode will effectively "hand over" the GSM module inside the RTCU device so that it is exclusively controlled by any external device connected to serial port 1 on the device.
During the time where modem mode is enabled the GSM module will appear OFF to the VPL application and any attempt to turn the GSM module on by using the gsmPower function will return an error code. The GSM status LED will be inactive when gsmModemMode is active.

Before enabling modem mode, the GSM module must be powered on by using the gsmPower function, and when gsmModemMode returns, the GSM module will appear OFF and is therefore not accessible from the VPL application. When the modem mode is disabled again, the GSM module will resume normal operation as before the modem mode was enabled.

Controlling the GSM module from serial port 1 is with standard AT commands and detailed information about modem specific commands can be requested from Logic IO.

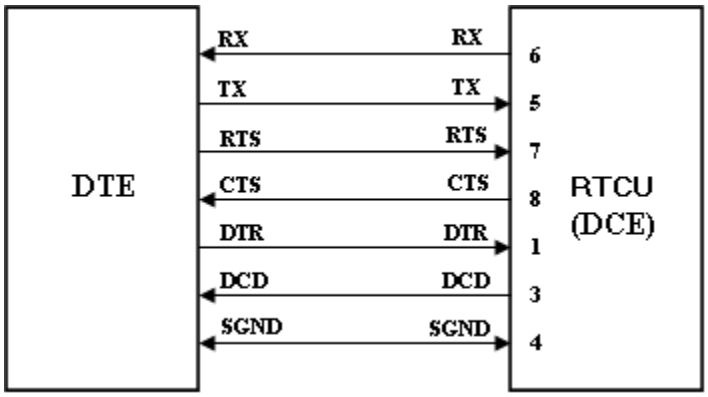
Warning:
Do not use the following AT commands: **AT+IPR, AT+ICF, or AT&W**.

DCE Hardware Interface:

Serial port 1 is has been designed and documented as a DTE interface, and a modem is a DCE interface.
When entering Modem mode, serial port 2 changes therefore from DTE to DCE mode by using the following interface:

Pin #	Signal
1	DTR.
2	Not used.
3	DCD.
4	SGND.
5	TX.
6	RX.
7	RTS.
8	CTS.

The wiring for connecting a DTE device, such as a PC, to the device in Modem mode are as follows:



Please note that it is not allowed to change the modem speed with an AT modem command.

Input:

enable : BOOL
Will enable or disable the dedicated GSM modem mode.

baud : DINT *(default: 9600)*
The baud rate (bits per second) to use for communication with the GSM module over the serial port 1.
Valid values are: 9600, 19200, or 38400.

bit : SINT *(default: 8)*
Number of data-bits to use for communication with the GSM module over the serial port 1.
Valid values are: 7 or 8.

parity : SINT *(default: 0)*
The parity to use for communication with the GSM module over the serial port 1.
Valid values are: 0 (NONE), 1 (EVEN), or 2 (ODD).

stopbit : SINT *(default: 1)*
Number of stop-bits to use for communication with the GSM module over the serial port 1
Valid values are: 1 or 2.

Returns: INT

0 Operation was successful.
1 GSM module not available.
2 Serial port 1 is not present.
3 Current processor speed is not supported.
4 Invalid baud rate.
5 Modem is active in a mobile network/CSD or VOICE session.
6 Serial port 1 is in use.
7 GSM module is not powered on.
8 Function is only supported on the MX2 pro, DX4 pro and MX2 turbo.

Declaration:

```
FUNCTION gsmModemMode : INT;
VAR_INPUT
   enable      : BOOL;
   baud        : DINT;
   bit         : SINT;
   parity      : SINT;
```

```
   stopbit     : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   DI1 : BOOL;
END_VAR;

PROGRAM example;
VAR
   rc : INT;
END_VAR;

BEGIN
   ...
   IF DI1 THEN
      // Enter modem mode, when input signal is high
      rc := gsmModemMode(enable:=TRUE,baud:=19200);
      ...
   ELSE
      // Exit modem mode, when input signal is low
      rc := gsmModemMode(enable:=FALSE);
      ...
   END_IF;
END;

END_PROGRAM;
```

4.2.19.42 gsmSendFlashSMS (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.50 / 1.00.00

This function sends a so-called Flash/Alert SMS message over the GSM network. A Flash SMS message will appear on the receiver's mobile phone as an instant pop-up message that will not require entering a menu etc. to read. Not all mobile phones support Flash SMS messages.

Please note that you should always check the return value from this function. There are a number of reasons why this function could fail such as too much traffic on the GSM network etc., so always check the return value, and based on that it is possible to make a strategy for resending a message.

Note: gsmSendFlashSMS is not designed for use with the RTCU Communication Hub.

Input:

phonenumber : STRING (max length is 20, excluding " " and "-")
Number to send the SMS message to (if "9999" is used, the message will be sent to a connected PC - either through a cable connection or via an active data connection).

In case the device is not connected to the GSM network, the return value will be 1.

message : STRING
The actual message to send (maximum of 70 characters).

Returns: INT

0 If successful.
1 If general error.
2 If CMS/CME error (error on GSM net).
3 If not connected to a base station.
4 If GSM module is not powered on (see gsmPower()).
5 If GSM module is busy with a data call.

Declaration:

```
FUNCTION gsmSendFlashSMS : INT;
VAR_INPUT
   phonenumber : STRING;
   message     : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Turn on power to the GSM module
gsmPower(power := TRUE);

BEGIN
   ...
   // Note: we are NOT checking the return code from the gsmSendFlashSMS
function in this example
   gsmSendFlashSMS(phonenumber := "+44 12 34 56 789", message := "This is a
test message sent using SMS");
   ...
END;

END_PROGRAM;
```

4.2.19.43 gsmSimSetPIN (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 2.10 / 1.00.00

This function enables or disables the PIN code on the SIM card.
Optionally the PIN code can be changed when enabled.
Only PIN codes with 4 characters are supported.

The device must be reset for the changes to take effect.

Note: when the PIN code is enabled and/or changed, it is required to call gsmSetPIN so the device knows to use the correct PIN code when accessing the GSM module.

Input:

enable : BOOL
TRUE: Enables the PIN code.
FALSE: Disables the PIN code.
pin : STRING
The current PIN code to use for accessing the SIM card.

Note: the correct PIN code is required when enabling and disabling.

pin_new : STRING
The new PIN code.
Note: if this is empty, the PIN code will not be changed.

Returns:

0	Success.
1	General error.
2	GSM module is not powered on (see gsmPower()).
3	Failed to enable/disable PIN code.
4	Failed to change PIN code.
5	Illegal PIN code.

Declaration:

```
FUNCTION gsmSimSetPIN : INT;
VAR_INPUT
   enable  : BOOL;
   pin     : STRING;
   pin_new : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Turn on power to the GSM module (it will now use "1234" as the pincode)
gsmPower(power := TRUE);

...

// Enable PIN code
gsmSimSetPIN(enable := TRUE, pin := "1234"); // Needs only to be set once
gsmSetPIN(pin :="1234"); // Needs only to be set once (The RTCU stores this
information in non-volatile memory)
boardReset();

...

END_PROGRAM;
```

4.2.19.44 gsmSelectSIM (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, NX-200, LX2
Firmware version: 4.00 / 1.40.00

This function selects which SIM card to use on devices with multiple SIM card readers.
The selected SIM card reader is persistent over resets until set again by the function or from the RTCU IDE in the configuration dialog.

If called with the currently selected SIM card reader, the function returns immediately, without affecting the GSM connection.
When changing the SIM card while the GSM modem is powered on, the entire GSM stack is restarted, causing any active calls, mobile network connections etc. to terminate. Once the GSM connection has been re-established using the new SIM card, mobile network the RTCU Communication Hub connection will be automatically reestablished.

If the GSM modem is not powered on, the selection will be stored, and used when the GSM modem is powered on next time.

Input:

mode : SINT

1 Internal SIM card reader.
2 External SIM card reader.

Returns:

0 Success.
1 Illegal mode.
2 Internal SIM card reader is not enabled.
3 SIM card reader is not present.

Declaration:

```
FUNCTION gsmSelectSIM : INT;
VAR_INPUT
   mode  : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

...

// Select internal SIM card reader
gsmSelectSIM(mode := 1); // Needs only to be set once

...

END_PROGRAM;
```

4.2.19.45 gsmNetworkTime (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 3.10 / 1.00.00 (5.04/1.40.00 for the UTC parameter)

This function requests the current time expressed as seconds since 1980-1-1 00:00:00 (LINSEC), from the cellular network.

Note:
Not all networks offer a time-service. The time information is transmitted with a variable frequency by the network.

Input:

UTC : BOOL *(default: FALSE)*
Specifies whether the local-time or UTC time should be returned.
Not all devices supports UTC time and will therefore return the local time in all cases.

Returns: **DINT**

The current time as seconds since 1980-1-1 00:00:00 (LINSEC).

```
 0   - Updated network time is currently not available.
-1   - Network time not supported by device.
-2   - Not connected to the network.
-3   - General error.
```

Declaration:

```
FUNCTION gsmNetworkTime : INT;
VAR_INPUT
   UTC  : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   update : BOOL;
END_VAR;

PROGRAM test;
VAR
   upd_old : BOOL;
   time    : DINT;
END_VAR;

   // Initialize
   gsmPower(power:=ON);
   DebugMsg(message:="Program running");

BEGIN
   ...
   // Update device clock
   IF update AND NOT upd_old THEN
      // Get time
      time := gsmNetworkTime();
      // Set current time
      IF time > 0 THEN
         clockSet(linsec := time);
      END_IF;
   END_IF;
   upd_old := update;
END;
END_PROGRAM;
```

4.2.19.46 gsmSetWakeup (Function)

Architecture: **NX32L**
Device support: **All, except the RTCU NX-400.**
Firmware version: **1.36.00**

This function is used to configure the system to wake from suspend on GSM activity.
It is similar to using the GSM parameter on pmWaitEvent.

pmSuspend return codes for this wake-up source:

Error	Type	Value	Description
True	5	-10	The GSM module is OFF.

True	5	-11	The GSM module is not in low-power mode.
True	5	-12	A voice session is active.
True	5	-13	The GSM module is busy from the last suspend.
True	5	-14	The GSM module is ON, but not selected as a wake-up source.
False	5	1	Wake-up on GSM activity.

Input:

Enable : BOOL (default: TRUE)
Set to true to enable the wake-up, set to false to disable it.

Returns: INT

1 - The system has been configured to wake.
0 - This function is not supported.
-1 - No GSM module present.
-2 - The GSM module is OFF.
-3 - The GSM module is not in low-power mode.

Declaration:

```
FUNCTION ALIGN gsmSetWakeup : INT;
VAR_INPUT
   enable: BOOL := TRUE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   gsmPowerLP(power := ON);

   // Enable wake on cellular activity
   gsmSetWakeup(enable := ON);
   ...
END;
END_PROGRAM;
```

4.2.19.47 gsmVoLTEEnable (Function)

Architecture: NX32L
Device support: All
Firmware version: 2.30.00

This function tries to enable VoLTE(Voice over LTE) on the device. Not all networks and configurations are supported.

The final success of this operation can be checked with gsmVoLTEStatus().

When VoLTE is enabled it will be stored persistently, so it should only be set once for the specific SIM card / network.

On NX devices, the operation takes some time.

On LX devices, the system will restart, and it will also take some time. It is necessary to implement a mechanism to avoid calling gsmVoLTEEnable() on every boot when using a network/sim where VoLTE is not supported, to avoid a boot loop.

Input:

enable: BOOL (default: TRUE)
Set to true to try to enable VoLTE.

Returns: INT

1	Success
0	Not supported.
-1	Not an LTE module.
-2	No SIM card present.
-3	No power to the LTE engine.
-4	Could not enable VoLTE.

Declaration:

```
FUNCTION gsmVoLTEEnable : INT;
VAR_INPUT
   enable : BOOL := TRUE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Turn on power to the GSM module
gsmPower(power := TRUE);

BEGIN
   ...
   // Enable VoLTE
   gsmVoLTEEnable(enable := TRUE);
   ...
END;

END_PROGRAM;
```

4.2.19.48 gsmVoLTEStatus (Function)

```
Architecture:       NX32L
Device support:     All
Firmware version:   2.30.00
```

This function returns the VoLTE status.

VoLTE can be enabled with gsmEnableVoLTE() and its success can be checked using gsmVoLTEStatus.

Input:

None

Returns: **INT: TODO: Update**

1	Success
0	Not supported.
-1	Not an LTE module.
-2	No SIM card present.
-3	No power to the LTE engine.
-4	Could not enable VoLTE.

Declaration:

```
FUNCTION gsmVoLTEStatus : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Turn on power to the GSM module
gsmPower(power := TRUE);

BEGIN
   ...
   // Get VoLTE status
   DebugFmt(message:="VoLTE: \1", v1:=gsmVoLTEStatus());
   ...
END;

END_PROGRAM;
```

4.2.20 gui: GUI functions (NX32L)

4.2.20.1 gui: GUI functions

The GUI functions provide a more advanced API for using the display.
It is possible to switch between this API and the more simple display API using guiOpen/guiClose and displayPower.

At its core, this API provides three different kinds of GUI elements, in addition to some common functionality:
1. Custom Forms
2. Dialogs
3. Menus

Limits
The limit to the number of different objects currently is as follows:
- 10 Forms.
- 100 controls across all forms.
- 10 Menus (each with up to 8 items).

Trying to create objects beyond these limits will result in error code -2, not enough memory / resources.

In addition to the custom GUIs, there is also some built-in GUI, to configure and monitor the status of the device.

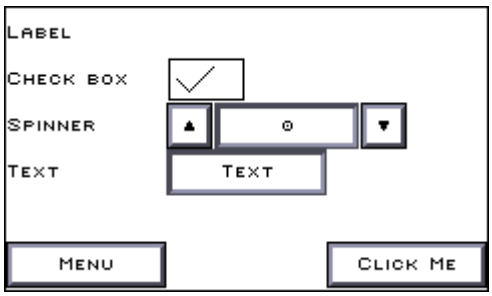
Common Functions

The following functions can be used across the library:

- guiGrabScreenShot — Takes a screenshot of the display. Can also be called when the display API is used.
- guiCalibrateTouch — Calibrates the touch panel.
- guiOpen — Opens the GUI API
- guiClose — Closes the GUI API
- guiWaitEvent — Waits for a GUI event to occur.
- guiGetTag — Retrieves the tag value for the provided handle.

Form Functions

The custom forms makes it possible to create forms with many different types of controls, to use for e.g. data entry, visualization of data, configuration and many other things.
To make it easy to layout the controls on the forms, the controls are arranged in a grid of a custom size.



- guiFormCreate — Creates a new form
- guiFormShow — Makes the form visible
- guiFormClose — Hides the visible form.
- guiFormRemove — Releases the form and the resources it uses.

Controls:

To create and use controls on the forms, please use the Control functions.

Dialogs

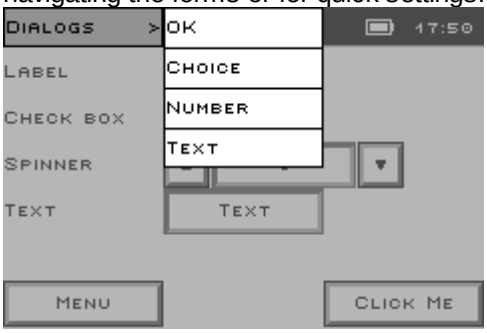
The dialogs can be used to interact with the user, without having to create a custom form.
The function blocks until the user responds or it times out.

The following dialog functions are available:

- guiCloseDialogs — Closes all open dialogs.
- guiShowMessage — Shows a dialog with a message for the user, with one or more buttons.
- guiShowChoice — Shows a dialog list of choices to the user.
- guiShowNumberInput — Shows a dialog for entering a number.
- guiShowTextInput — Shows a dialog for entering a text.
- guiShowDateInput — Shows a dialog for entering a date and/or a time.

Menus:

The menus allow for quick access to a large number of items, which can be used e.g. for navigating the forms or for quick settings.

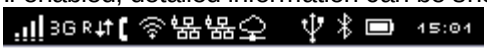


The following menu functions are available:

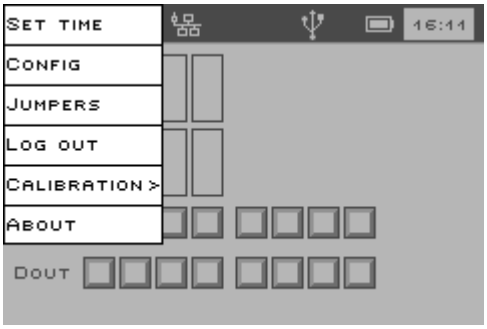
- guiMenuCreate — Creates a new menu
- guiMenuRemove — Releases the menu
- guiMenuSetItem — Sets an item in the menu
- guiMenuRemoveItem — Removes an item from the menu
- guiMenuShow — Shows the menu.
- guiMenuClose — Closes the menu
- guiMenuClickEvent — Handles a click on a menu item.

Built-in GUI:

Status panel:
The status panel contains a list of icons, showing the status for many of the interfaces on the device.
If enabled, detailed information can be shown by clicking on the icons.



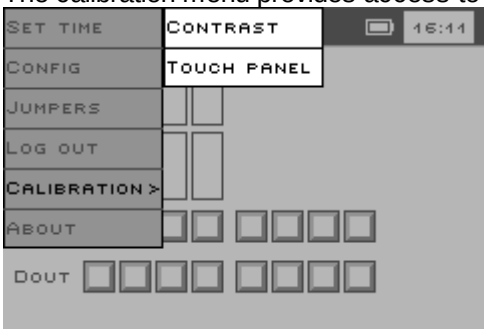
System menu:
The system menu allows for accessing viewing and editing some configuration settings on the device, such as the system switches.
The icons in the status bar will show details when clicked, if the system menu is enabled, or if it is allowed in the configuration dialog.
The system menu is accessed by clicking on the clock in the status bar.



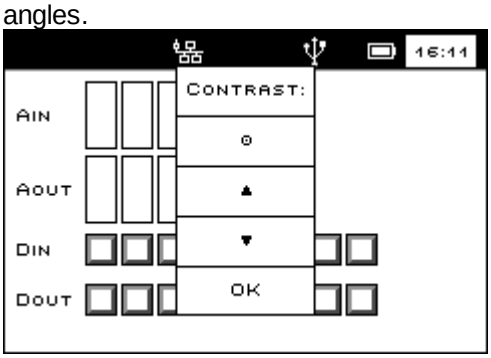
The following function controls access to the system menu:

- guiSysMenuEnable — Enables the system menu
- guiSysMenuSetPassword — Configures the password that the user can use to enable the system menu.

Calibration menu:
The calibration menu provides access to calibrating the display.

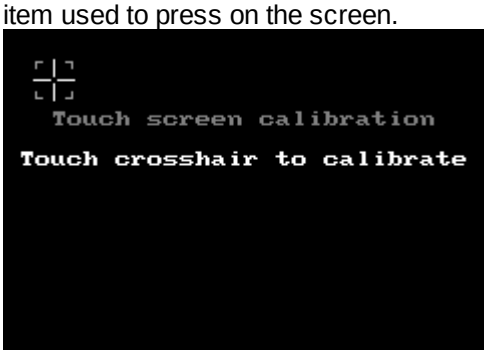


It has two entries:
Contrast
This is used for adjusting the contrast of the display, to e.g. optimize it for different viewing angles.



Clicking the arrows adjust the contrast up and down and is applied immediately.

Touch panel
This menu item is used to calibrate the touch panel, to adjust for e.g. the access angle or the item used to press on the screen.



To calibrate, click accurately on each of the five points as they are shown, and the dialog closes.
To close the dialog without changing the calibration, wait for 60 seconds and the dialog closes by itself.
The calibration dialog can also be activated with the guiCalibrateTouch function.

4.2.20.2 guiGrabScreenshot (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function grabs a screenshot of the display and saves it to a file.
The screenshot is saved as a gzipped portable graymap format file(pgm), which can then be converted to more common formats on the PC.
Some image manipulating programs such as GIMP(https://www.gimp.org/) supports gzipped pgm files directly, while others requires it to be extracted first, which can be done e.g. with 7zip(http://www.7-zip.org/). To convert from pgm to more common image formats, it is possible to use e.g. http://www.xnview.com/en/xnconvert/.

This function can also be called while using the display API.
The screenshots can be useful for e.g. documentation.

Note: Menus can not be captured with this function.

Input:

filename : STRING
The filename to save the image to.
format : SINT *default 0*
The format to use when saving the image. Currently only gzipped portable graymap (format:=0) is supported.

Returns: INT

28	- Unknown file system error.
23	- Media is write protected.
16	- Media is not present.
7	- Media is full.
6	- A file with the same name already exists.
5	- File not found.
4	- Invalid file name.
3	- Path was not found.
1	- Media not open.
0	- Success.
-1	- Display is turned off.
-3	- Invalid format.
-11	- Not supported.

Declaration:

```
FUNCTION guiGrabScreenshot : INT;
VAR_INPUT
   filename : STRING;
   format   : SINT := 0;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
```

```
PROGRAM test;
VAR
   rc : INT;
END_VAR;

BEGIN
   ...
   // Save screenshot to B:\screen.gz
   rc := guiGrabScreenshot(filename := "B:\screen.gz");
   ...
END;
END_PROGRAM;
```

4.2.20.3 guiCalibrateTouch (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.04.00

This function shows a dialog to calibrate the touch panel.
It will stay on the screen for 60 seconds an then close, unless the calibration is completed before that.
The calibration dialog can also be activated from the system menu.

Input:
None

Returns: **INT**

0	- Success.
-5	- Failed to start calibration
-9	- Busy
-11	- Not supported.
-13	- Timeout

Declaration:

```
FUNCTION guiCalibrateTouch : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc : INT;
END_VAR;

BEGIN
   ...
   // Show touch panel calibration dialog
   rc := guiCalibrateTouch();
   ...
END;
END_PROGRAM;
```

4.2.20.4 guiOpen (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function opens the GUI API.

Input:
None.

Returns: INT
0 - The GUI API is now open.
-1 - The Display API is active. The Display API and the GUI API can not be active at the same time. To disable the Display API, use displayStop or toggle DisplayPower OFF and ON again.
-11 - Not supported.

Declaration:
```
FUNCTION guiOpen : INT;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc : INT;
END_VAR;

BEGIN
   ...
   // Opens the GUI API
   rc := guiOpen();
   ...
END;
END_PROGRAM;
```

4.2.20.5 guiClose (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function closes the GUI API.

Input:
keep : BOOL *Default FALSE*
Set to TRUE to keep the handles valid after the API is closed. Leave at FALSE to free all the used handles.

Returns: INT
0 - The GUI API is now closed.
-11 - Not supported.

Declaration:
```
FUNCTION guiClose : INT;
VAR_INPUT
   keep : BOOL := FALSE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc : INT;
END_VAR;

BEGIN
   ...
   // Closes the GUI API
   rc := guiClose();
   ...
END;
END_PROGRAM;
```

4.2.20.6 guiWaitEvent (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function will wait until a GUI event is raised. It has an optional timeout.

An event is a notification that something has happened in the GUI interface.
The events are normally triggered by the user, depending on the available controls.

The events are queued until appropriate action is taken as described in this table:

Event#	Event	Action required	Action
1	A button has been clicked.	Yes	Call guiButtonClickEvent.
2	A check box has changed state.	Yes	Call guiCheckBoxChangeEvent.
3	A menu has been clicked.	Yes	Call guiMenuClickEvent.
128	A numeric spinner has changed state.	No	Call guiSpinnerGetValue to get the current value of the spinner
129	A text box has changed value.	No	Call guiTextBoxGetText to get the current text of the text box.
130	The selection has changed in a List control.	No	Call guiListGetSelected to get the current index of the selection.
131	Login to the system menu failed.	No	Optional: Disable login with guiSysMenuSetPassword in case of brute-force attack.
132	Login to the system menu has succeeded.	No	None
133	A radio button has been selected.	No	The handle contains the selected radio button.
134	The system menu has been logged out.	No	None

guiWaitEvent notifies the application of the oldest queued event.

This means that if the appropriate action is not taken, guiWaitEvent will continue to report the same event.

If no action is required, only the handle of the control is provided, and it is then recommended to manually read the wanted values of the control if needed.

Note: waiting for an event using this function will block the calling thread.

Input:

timeout : INT (-1,0..32767) ***(default -1)***
Timeout period in milliseconds to wait.
0 - Disable timeout. The function will return immediately.
-1 - Wait forever. The function will only return when data is received.

Output:

handle : SYSHANDLE
The handle of the control that caused the event.

Returns: INT

129 - A text box has changed value
128 - A numeric spinner has changed value.
3 - A menu item has been clicked.
2 - A check box has changed state.
1 - A button has been clicked.
0 - Timeout.
-1 - Interface is not open (see guiOpen).
-3 - Timeout is outside valid range.
-6 - Event is no longer valid.
-11 - GUI API is not supported.

Declaration:

```
FUNCTION guiWaitEvent : INT;
VAR_INPUT
   timeout  : INT := -1;
   handle   : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   button : SYSHANDLE;
   check  : SYSHANDLE;
   menu   : SYSHANDLE;
   label  : SYSHANDLE;
   spinner: SYSHANDLE;
   textbox: SYSHANDLE;
END_VAR;

FUNCTION ButtonHandler
VAR
   rc     : INT;
   handle : SYSHANDLE;
   tag    : DINT := 0;
   lp     : BOOL;
END_VAR;
   rc := guiButtonClickEvent(button := handle, long_press := lp);
   IF rc = 0 THEN
      guiGetTag(handle := handle, tag := tag);
      IF lp THEN
         DebugFmt(message:="Button with tag \4 was pressed", v4 := tag);
      ELSE
         DebugFmt(message:="Button with tag \4 was clicked", v4 := tag);
         guiMenuShow(menu := menu);
      END_IF;
   END_IF;
END_FUNCTION;
```

```
FUNCTION CheckBoxHandler
VAR
   rc      : INT;
   handle  : SYSHANDLE;
   tag     : DINT := 0;
   checked : BOOL;
END_VAR;
   rc := guiCheckBoxChangeEvent(check_box := handle, checked := checked);
   IF rc = 0 THEN
      guiGetTag(handle := handle, tag := tag);
      IF checked THEN
         DebugFmt(message:="Check box with tag \4 was checked", v4 := tag);
      ELSE
         DebugFmt(message:="Check box with tag \4 was unchecked", v4 := tag);
      END_IF;
   END_IF;
END_FUNCTION;

FUNCTION MenuHandler
VAR
   rc     : INT;
   handle : SYSHANDLE;
   tag    : DINT := 0;
   index  : INT;
END_VAR;
   rc := guiMenuClickEvent(menu := handle, index := index);
   IF rc = 0 THEN
      IF BOOL(handle) THEN
         guiGetTag(handle := handle, tag := tag);
         DebugFmt(message:="Menu with tag \4, index \1 was clicked", v1 :=
index, v4 := tag);
      ELSE
         DebugMsg(message:="No menu was clicked");
      END_IF;
   END_IF;
END_FUNCTION;

FUNCTION SpinnerHandler
VAR_INPUT
   handle : SYSHANDLE;
END_VAR;
VAR
   rc     : INT;
   tag    : DINT := 0;
   value  : DINT;
END_VAR;
   rc := guiSpinnerGetValue(spinner := handle, value := value);
   IF rc = 0 THEN
      guiGetTag(handle := handle, tag := tag);
      DebugFmt(message:="Spinner with tag \4 has value " + dintToStr(v :=
value), v4 := tag);
   END_IF;
END_FUNCTION;

FUNCTION TextBoxHandler
VAR_INPUT
  handle : SYSHANDLE;
END_VAR;
VAR
   rc    : INT;
   tag   : DINT := 0;
   text  : STRING;
END_VAR;
```

```
   rc := guiTextBoxGetText(text_box := handle, text := text);
   IF rc = 0 THEN
      guiGetTag(handle := handle, tag := tag);
      DebugFmt(message:="Text box with tag \4 has text '" + text + "'", v4 :=
tag);
   END_IF;
END_FUNCTION;

//-------------------------------------------------------------------------
-
// THREAD_BLOCK eventMonitor
//-------------------------------------------------------------------------
-
THREAD_BLOCK eventMonitor;
VAR
   event       : INT := 0;
   handle      : SYSHANDLE;
END_VAR;

WHILE event <> -1 DO
   event := guiWaitEvent(timeout := -1, handle := handle);
   CASE event OF
      0   : DebugFmt(message := "Timeout");
      1   : ButtonHandler();
      2   : CheckBoxHandler();
      3   : MenuHandler();
    128   : SpinnerHandler(handle := handle);
    129   : TextBoxHandler(handle := handle);
   ELSE
      DebugFmt(message := "guiWaitEvent - event=\1", v1 := event);
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;

PROGRAM example;
// These are the local variables of the program block
VAR
   eventMon: eventMonitor;
   rc : INT;
   form : SYSHANDLE;
END_VAR;
// The next code will only be executed once after the program starts

   DebugFmt(message:="guiOpen: \1", v1:=guiOpen());

   // Create a form with 5x5 tiles
   rc := guiFormCreate(form := form, grid := 6);
   DebugFmt(message:="guiFormCreate: \1", v1:=rc);

   // Row 1
   // Create a label with the text "Label"
   rc := guiLabelCreate(label := label, form:=form, x:=1, y:=1, w:=2, h:=1,
text := "Label", tag:=10);
   DebugFmt(message:="guiLabelCreate: \1", v1:=rc);

   // Row 2
   // Create a check box
   rc := guiCheckBoxCreate(check_box := check, form:=form, x:=3, y:=2, w:=1,
h:=1, tag:=20, style := 1, checked := TRUE);
   DebugFmt(message:="guiCheckBoxCreate: \1", v1:=rc);

   // Row 3
   // Create a spinner for the range -10..10
   rc := guiSpinnerCreate(spinner := spinner, form:=form, x:=3, y:=3, w:=3,
```

```
h:=1,
            tag:=30, min_value := -10, max_value := 10, value := 0);
   DebugFmt(message:="guiSpinnerCreate: \1", v1:=rc);

   // Row 4
   // Create a text box
   rc := guiTextBoxCreate(text_box := textbox, form:=form, x:=3, y:=4, w:=2,
h:=1, tag:=40, text := "Text", max_len := 10);
   DebugFmt(message:="guiTextBoxCreate: \1", v1:=rc);

   // Row 6
   // Create a button with the text "Click Me"
   rc := guiButtonCreate(button := button, form:=form, x:=5, y:=6, w:=2, h:=1,
text := "Click Me", tag:=50);
   DebugFmt(message:="guiButtonCreate: \1", v1:=rc);

   // Create the menu.
   rc := guiMenuCreate(menu := menu, tag := 60);
   DebugFmt(message:="guiMenuCreate: \1", v1:=rc);

   // Set the menu items.
   rc := guiMenuSetItem(menu := menu, index:=1, title := "Item");
   DebugFmt(message:="guiMenuSetItem: \1", v1:=rc);

   // Start event monitor
   eventMon();

   // Make form visible
   rc := guiFormShow(form := form);
   DebugFmt(message:="guiFormShow: \1", v1:=rc);

BEGIN
// Code from this point until END will be executed repeatedly
END;
END_PROGRAM;
```

4.2.20.7 guiGetTag (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function retrieves the tag value associated with the provided handle.
The tag value is a user defined value that is passed along with certain menus and controls.

Input:

handle : SYSHANDLE
Handle to get the tag for. This function is currently valid for control and menu handles.

Output:

tag : DINT
The value of the tag.

Returns: INT

0	- The tag has been retrieved.
-1	- Interface is not open (see guiOpen).
-3	- The handle is not valid.
-11	- The GUI API is not supported.

Declaration:

```
FUNCTION guiGetTag : INT;
VAR_INPUT
   handle : SYSHANDLE;
   tag    : ACCESS DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc     : INT;
   tag    : DINT;
   handle : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Get the tag value
   rc := guiGetTag(handle := handle, tag := tag);
   ...
END;
END_PROGRAM;
```

4.2.20.8 Forms

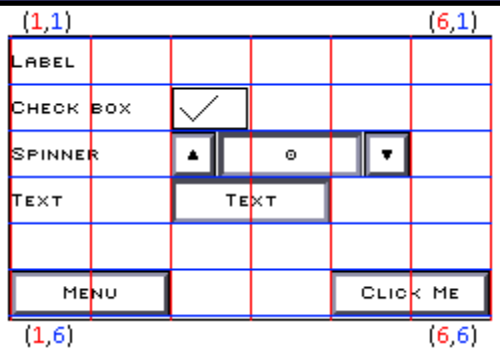
4.2.20.8.1 guiFormCreate (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function creates a new custom form and provides a handle for it.

Any controls on the form are placed on a grid with an equal number of tiles in each direction, with tile 1,1 in the upper-left corner.

This table shows an example of the layout of a form with controls:

Form(grid := 6)	Control	Text/Value	Location(x,y)	Size(w x h)
	Label	"Label"	1,1	2 x 1
	Label	"Check Box"	1,2	2 x 1
	Check box	Checked	3,2	1 x 1
	Label	"Spinner"	1,3	2 x 1
	Spinner	0	3,3	3 x 1
	Label	"Text"	1,4	2 x 1
	Text box	"Text"	3,4	2 x 1
	Button	"Menu"	1,6	2 x 1
	Button	"Click Me"	5,6	2 x 1

Input:

grid : SINT (1..10)
The size of the grid.

Output:
form : SYSHANDLE
A handle to the new form.

Returns: INT
0 - The form has been created.
-1 - Interface is not open (see guiOpen).
-2 - Not enough memory / resources.
-3 - The grid size is invalid.
-11 - The GUI API is not supported.

Declaration:
```
FUNCTION guiFormCreate : INT;
VAR_INPUT
   grid : SINT;
   form : ACCESS SYSHANDLE;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc   : INT;
   form : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Create a form with 5x5 tiles
   rc := guiFormCreate(form := form, grid := 5);
   ...
END;
END_PROGRAM;
```

4.2.20.8.2 guiFormClose (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function closes the current form.

Input:
None.

Returns: INT
0 - Success.
-1 - Interface is not open (see guiOpen).
-11 - The GUI API is not supported.

Declaration:
```
FUNCTION guiFormClose : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc   : INT;
END_VAR;

BEGIN
   ...
   // Close form
   rc := guiFormClose();
   ...
END;
END_PROGRAM;
```

4.2.20.8.3 guiFormShow (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function makes the provided form visible.

Input:

form : SYSHANDLE
A handle to the form.

Returns: INT

0	- The form is now visible.
-1	- Interface is not open (see guiOpen).
-3	- The grid size is invalid.
-11	- The GUI API is not supported.

Declaration:

```
FUNCTION guiFormShow : INT;
VAR_INPUT
   form : SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc   : INT;
   form : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Create a form with 5x5 tiles
   rc := guiFormCreate(form := form, grid := 5);
   // Make form visible
   rc := guiFormShow(form := form);
   ...
```

```
END;
END_PROGRAM;
```

4.2.20.8.4 guiFormRemove (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function releases the provided form and all the controls on it.

Input:

form : SYSHANDLE
A handle to the form. Will be made invalid once the form is removed.

Returns: **INT**

0	- Success.
-1	- Interface is not open (see guiOpen).
-3	- The form is not valid or the form is visible.
-11	- The GUI API is not supported.

Declaration:

```
FUNCTION guiFormRemove : INT;
VAR_INPUT
   form : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc   : INT;
   form : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Create a form with 5x5 tiles
   rc := guiFormCreate(form := form, grid := 5);
   // Make form visible
   rc := guiFormShow(form := form);
   // Close form
   rc := guiFormClose();
   // Remove form
   rc := guiFormRemove(form := form);
   ...
END;
END_PROGRAM;
```

4.2.20.9 Controls

4.2.20.9.1 gui: Control functions

The control functions are used to create and handle controls.

Common control functions

Functions that can be used on different types of controls.

- guiControlSetText — Sets the text of a control that supports it.
- guiControlRemove — Removes a control from a form.

Label

The label is a simple control for showing text.

- guiLabelCreate — Creates a label to show text

Button

The button can be clicked by the user to trigger events.

- guiButtonCreate — Creates a button.
- guiButtonClickEvent — Handles a click on the button.

Check Box

The check box can be used both to show the status of something, and for user input.

- guiCheckBoxCreate — Creates a new check box
- guiCheckBoxChangeEvent — Handles the event when the check box changes status.
- guiCheckBoxSetChecked — Sets if the check box is checked
- guiCheckBoxIsChecked — Determines if the check box is checked.

Numeric Spinner

The numeric spinner is used for entering numeric values.

- guiSpinnerCreate — Creates a new spinner
- guiSpinnerGetValue — Retrieves the value of the spinner
- guiSpinnerSetValue — Sets the value of the spinner

Text box

The text box is used for entering text. Clicking the text box will show a dialog to change the value of the text box.

- guiTextBoxCreate — Creates a text box.
- guiTextBoxGetText — Retrieves the text.

Radio button

The radio button is used for selecting between multiple choices.

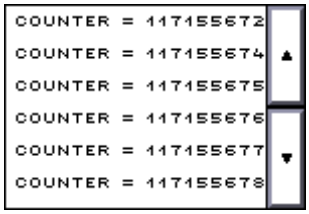
- guiRadioButtonCreate — Creates a radio button.
- guiRadioButtonIsSelected — Checks if the radio button is selected.
- guiRadioButtonSetSelected — Selects the radio button.

Progress bar

The progress bar is used for showing a process is progressing.

- guiProgressBarCreate — Creates a progress bar.
- guiProgressBarSetValue — Updates the value of the progress bar.

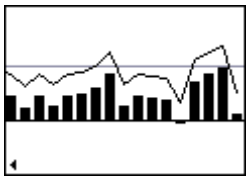
List control



The list control can be used both for showing e.g. output from an ongoing process and for allowing the user to select between multiple choices.

- guiListCreate — Creates a list control.
- guiListSetItem — Set the value of a specific item.
- guiListAppendText — Appends an item to the end of the list, dropping the first item if the list is full.
- guiListGetSelected — Retrieves the index of the selected item.
- guiListSetSelected — Selects an item.

Graph control



The graph control can be used for visualizing data.

- guiGraphCreate — Creates a new graph.
- guiGraphSetBar — Set the value of a bar in the bar graph.
- guiGraphSetLinePoint — Set the value of a point on a line graph.
- guiGraphScrollNext — Append the entries to the graph.
- guiGraphSetMarker — Adds a horizontal marker to the graph.

4.2.20.9.2 guiControlRemove (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function removes the provided control, and releases the resources associated with it.

Input:

control : SYSHANDLE
A handle to the control. Will be made invalid once the control is removed.

Returns: INT

0 - Success.
-1 - Interface is not open (see guiOpen).
-3 - The handle is not valid.
-11 - The GUI API is not supported.

Declaration:

```
FUNCTION guiControlRemove : INT;
VAR_INPUT
   control : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc     : INT;
   button : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Remove handle
   rc := guiControlRemove(control := button);
   ...
END;
END_PROGRAM;
```

4.2.20.9.3 guiControlSetText (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function sets the text for the provided control.
This is currently available for the label, button and text box controls.

Input:

control : SYSHANDLE
A handle to the control to set the text for.

text : STRING
The text to set.

Returns: INT

0	- Success.
-1	- Interface is not open (see guiOpen).
-3	- The form is not valid or the form is visible.
-8	- The text contains invalid characters.
-11	- The GUI API is not supported.

Declaration:

```
FUNCTION guiControlSetText : INT;
VAR_INPUT
   control : SYSHANDLE;
   text    : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc     : INT;
   label : SYSHANDLE;
END_VAR;

BEGIN
   ...
```

```
    // Set the text of the previous created label to "New Text"
    rc := guiControlSetText(control := label, text := "New Text");
    ...
END;
END_PROGRAM;
```

4.2.20.9.4 Button

4.2.20.9.4.1 guiButtonCreate (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function creates a new button on the provided form.
Buttons triggers button click events, when clicked by the user.
To change the text after it has been created, use guiControlSetText.

Input:

form : SYSHANDLE
A handle to the form that the button is created on.

x : SINT *default 1*
The horizontal location of the button.

y : SINT *default 1*
The vertical location of the button.

w : SINT *default 1*
The width of the button.

h : SINT *default 1*
The height of the button.

text : STRING
The text for the button to show.

tag : DINT
The tag value to store.

Output:

button : SYSHANDLE
The handle to the new button.

Returns: INT

0	- Success.
-1	- Interface is not open (see guiOpen).
-2	- Not enough memory / resources.
-3	- Invalid argument.
-8	- The text contains invalid characters.
-10	- Invalid location or dimension.
-11	- The GUI API is not supported.
-12	- The location is already used.

Declaration:

```
FUNCTION guiButtonCreate : INT;
VAR_INPUT
    form    : SYSHANDLE;
```

```
   x      : SINT := 1;
   y      : SINT := 1;
   w      : SINT := 1;
   h      : SINT := 1;
   text   : STRING;
   tag    : DINT;
   button : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc     : INT;
   button : SYSHANDLE;
   form   : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Create a button with the text "Click Me"
   rc := guiButtonCreate(button := button, form:=form, x:=2, y:=3, w:=3, h:=1,
text := "Click Me", tag:=40);
   ...
END;
END_PROGRAM;
```

4.2.20.9.4.2 guiButtonClickEvent (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function retrieves a button click event from the event queue, which is triggered when the user presses and then releases a button.
This function must be called to clear event 1 from guiWaitEvent.

Input:

None.

Output:

button : SYSHANDLE
The handle to the button.

long_press : BOOL
True if the button was kept pressed for a while, false if it was released sooner.

Returns: INT

0	- Data is ready.
-1	- Interface is not open (see guiOpen).
-6	- Data is not ready
-11	- The GUI API is not supported.

Declaration:

```
FUNCTION guiButtonClickEvent : INT;
VAR_INPUT
   button     : ACCESS SYSHANDLE;
   long_press : ACCESS BOOL;
END_VAR;
```

Example:

```
...
FUNCTION ButtonHandler
VAR
   rc     : INT;
   handle : SYSHANDLE;
   tag    : DINT := 0;
   lp     : BOOL;
END_VAR;
   rc := guiButtonClickEvent(button := handle, long_press := lp);
   IF rc = 0 THEN
      guiGetTag(handle := handle, tag := tag);
      IF lp THEN
         DebugFmt(message:="Button with tag \4 was pressed", v4 := tag);
      ELSE
         DebugFmt(message:="Button with tag \4 was clicked", v4 := tag);
      END_IF;
   END_IF;
END_FUNCTION;
...
```

4.2.20.9.5 Check box

4.2.20.9.5.1 guiCheckBoxCreate (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function creates a new check box on the provided form.
Check boxes toggles between two states when clicked, triggering the check box changed event.

Input:

form : SYSHANDLE
A handle to the form that the check box is created on.

x : SINT *default 1*
The horizontal location of the check box.

y : SINT *default 1*
The vertical location of the check box.

w : SINT *default 1*
The width of the check box.

h : SINT *default 1*
The height of the check box.

tag : DINT
The tag value to store.

locked : BOOL *default FALSE*
If this is TRUE, the state of the check box can not be changed.

checked : BOOL *default FALSE*
The default state of this check box.

style : SINT *default 0*
The type of check box to create.


0 - 3D border
1 - Simple border
2 - Toggle button

Output:

check_box : SYSHANDLE
The handle to the new check box.

Returns: INT

0	- Success.
-1	- Interface is not open (see guiOpen).
-2	- Not enough memory / resources.
-3	- Invalid argument.
-5	- Unknown error
-10	- Invalid location or dimension.
-11	- The GUI API is not supported.
-12	- The location is already used.

Declaration:

```
FUNCTION guiCheckBoxCreate : INT;
VAR_INPUT
   form      : SYSHANDLE;
   x         : SINT := 1;
   y         : SINT := 1;
   w         : SINT := 1;
   h         : SINT := 1;
   tag       : DINT;
   locked    : BOOL := FALSE;
   checked   : BOOL := FALSE;
   style     : SINT := 0;
   check_box : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc     : INT;
   check  : SYSHANDLE;
   form   : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Create a check box
   rc := guiCheckBoxCreate(check_box := check, form:=form, x:=2, y:=3, w:=1,
h:=1, tag:=42, style := 1);
   ...
END;
END_PROGRAM;
```

4.2.20.9.5.2 guiCheckBoxChangeEvent (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function retrieves a check box changed event from the event queue, which is triggered when the user clicks on the check box.
This function must be called to clear event 2 from guiWaitEvent.

Input:

None.

Output:

check_box : SYSHANDLE
The handle to the check box.

checked : BOOL
The new state of the check box

Returns: INT

0 - Data is ready.
-1 - Interface is not open (see guiOpen).
-6 - Data is not ready
-11 - The GUI API is not supported.

Declaration:

```
FUNCTION guiCheckBoxChangeEvent : INT;
VAR_INPUT
   check_box : ACCESS SYSHANDLE;
   checked   : ACCESS BOOL;
END_VAR;
```

Example:

```
...
FUNCTION CheckBoxHandler
VAR
   rc      : INT;
   handle  : SYSHANDLE;
   tag     : DINT := 0;
   checked : BOOL;
END_VAR;
   rc := guiCheckBoxChangeEvent(check_box := handle, checked := checked);
   IF rc = 0 THEN
      guiGetTag(handle := handle, tag := tag);
      IF checked THEN
         DebugFmt(message:="Check box with tag \4 was checked", v4 := tag);
      ELSE
         DebugFmt(message:="Check box with tag \4 was unchecked", v4 := tag);
      END_IF;
   END_IF;
END_FUNCTION;
...
```

4.2.20.9.5.3 guiCheckBoxIsChecked (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function returns the current state of the check box.

Input:

check_box : SYSHANDLE
A handle to the check box to get the state of.

Returns: INT

```
 1    - Check box is checked.
 0    - Check box is not checked.
-1    - Interface is not open (see guiOpen).
-3    - Invalid handle.
-11   - The GUI API is not supported.
```

Declaration:

```
FUNCTION guiCheckBoxIsChecked : INT;
VAR_INPUT
   check_box : SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   check  : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Check if the check box is checked
   IF guiCheckBoxIsChecked(check_box := check) = 1 THEN
      DebugMsg(message := "Checked");
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.20.9.5.4 guiCheckBoxSetChecked (Function)

```
Architecture:       NX32L
Device support:     NX-400
Firmware version:   1.00.00
```

This function sets the state of the check box.
Calling this function does not trigger the check box changed event.

Input:

check_box : SYSHANDLE
A handle to the check box to change the state for.

checked : BOOL
The wanted state of this check box.

Returns: INT

```
 0    - Success.
-1    - Interface is not open (see guiOpen).
-3    - Invalid handle.
-11   - The GUI API is not supported.
```

Declaration:

```
FUNCTION guiCheckBoxSetChecked : INT;
VAR_INPUT
   check_box : SYSHANDLE;
   checked   : BOOL;
END_VAR;
```

Example:

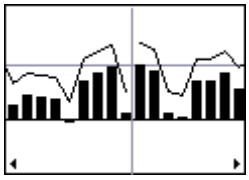
```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc      : INT;
   check   : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Set the check box as checked
   rc := guiCheckBoxSetChecked(check_box := check, checked:=TRUE);
   ...
END;
END_PROGRAM;
```

4.2.20.9.6 Graph

4.2.20.9.6.1 guiGraphCreate (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function creates a new graph on the provided form.
Graphs can be used to visualize data, supporting both a bar graph and line graphs.

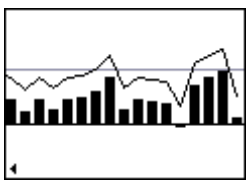

Wrap = TRUE

The total size of the graph depends on the x_scale and x_count parameters. If the product of these parameters are larger than the available area for the graph, the graph can be scrolled by clicking on the left and right thirds.

The entries can be set in two different ways.
It can be set using the index of the entry to update. This is suited to e.g. show historic data or data where each bar shows a specific reading.
Alternatively, data can be added to a buffer and then appended to the end of the graph when guiGraphScrollNext is called, dropping the oldest data if the graph is full. This is suited for showing the history of a reading, e.g. a temperature sensor.

Input:

form : SYSHANDLE
A handle to the form that the graph is created on.


Wrap = FALSE

x : SINT *default 1*
The horizontal location of the graph.

y : SINT *default 1*
The vertical location of the graph.

w : SINT ***default 1***
The width of the graph.

h : SINT ***default 1***
The height of the graph.

tag : DINT
The tag value to store.

y_min : INT ***default -100***
The minimum value to show on the graph.

y_max : INT ***default 100***
The maximum value to show on the graph.

x_scale : INT (1..25) ***default 5***
The width of each entry in pixels. This is the width of the bar and the space until the next bar for bar graphs, and the distance from one point on the line graph to the next.

x_count : INT (1..240) ***default 20***
The number of entries to store in the graph. This determines how many entries that can be added before the graph is full.

bars : INT (0,1) ***default 0***
Set to 1 to show the bar graph, set to 0 to disable the bar graph.

lines : INT (0..4) ***default 0***
Determines the number of line graphs to show.

wrap : BOOL ***default FALSE***
Determines how the graph reacts when entries are appended when it is full.
Set to TRUE, the graph will wrap around when it is full, similar to how an oscilloscope works. It shows a vertical line just after the latest data.
Set to FALSE, the entry is added at the end and the graph is scrolled, dropping the first entry.

Output:

graph : SYSHANDLE
The handle to the new graph.

Returns: INT

0	- Success.
-1	- Interface is not open (see guiOpen).
-2	- Not enough memory / resources.
-3	- Invalid argument.
-5	- Unknown error
-10	- Invalid location or dimension.
-11	- The GUI API is not supported.
-12	- The location is already used.

Declaration:

```
FUNCTION guiGraphCreate : INT;
VAR_INPUT
   form    : SYSHANDLE;
   x       : SINT := 1;
   y       : SINT := 1;
   w       : SINT := 1;
   h       : SINT := 1;
   tag     : DINT;
```

```
   y_min   : INT := -100;
   y_max   : INT :=  100;
   x_scale : INT :=  5;
   x_count : INT :=  20;
   bars    : INT :=  0;
   lines   : INT :=  0;
   wrap    : BOOL := FALSE;
   graph   : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc     : INT;
   graph  : SYSHANDLE;
   form   : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Create a graph with space for 50 items, for showing values between -10
and 100.
   rc := guiGraphCreate(graph := graph, form:=form, x:=1, y:=1, w:=5, h:=6,
      y_min := -10, x_count := 50, bars := 1, lines := 4);
   ...
END;
END_PROGRAM;
```

4.2.20.9.6.2 guiGraphScrollNext (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function appends the values set with guiGraphSetBar and guiGraphSetLinePoint with an x of -1 to the graph.
If no value is set for the bar, it will not be shown.
If no value is set for a point on one of the lines, it will be skipped, making the line continue directly to the next valid point, once it has been added.

Input:

graph : SYSHANDLE
The handle to the graph to append the items to.

Returns: INT

0	- Success.
-1	- Interface is not open (see guiOpen).
-3	- Invalid argument.
-11	- The GUI API is not supported.

Declaration:

```
FUNCTION guiGraphScrollNext : INT;
VAR_INPUT
   graph : SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc    : INT;
   graph : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Set the next bar to have the value 12
   rc := guiGraphSetBar(graph := graph, y:=12);
   // Set the next point on line 1 to 20
   rc := guiGraphSetLinePoint(graph := graph, line := 1, y := 20);
   // Append the new bar and point on the line to the graph.
   rc := guiGraphScrollNext(graph := graph);
   ...
END;
END_PROGRAM;
```

4.2.20.9.6.3 guiGraphSetBar (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function sets the value of an entry in the bar graph.

Input:

graph : SYSHANDLE
The handle to the graph to set the bar on.

x : INT (-1,1..240) *default -1*
The position to the entry to set the value for. When set to -1, it is put in the buffer, ready to be appended when calling guiGraphScrollNext, otherwise it is set directly on the graph. The maximum value is `x_count` from guiGraphCreate.

y : INT *default 0*
The value to set the bar to. If the value is outside the valid range of the graph, it will be truncated.

Returns: INT

0	- Success.
-1	- Interface is not open (see guiOpen).
-3	- Invalid argument.
-11	- The GUI API is not supported.

Declaration:

```
FUNCTION guiGraphSetBar : INT;
VAR_INPUT
   graph : SYSHANDLE;
   x     : INT := -1;
   y     : INT := 0;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
```

```
PROGRAM test;
VAR
   rc     : INT;
   graph  : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Set the value of bar 1 to -3
   rc := guiGraphSetBar(graph := graph, x:=1, y:=-3);
   // Set the value of bar 2 to 5
   rc := guiGraphSetBar(graph := graph, x:=2, y:=5);
   ...
END;
END_PROGRAM;
```

4.2.20.9.6.4 guiGraphSetLinePoint (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function sets the value of a line graph entry.

Input:

graph : SYSHANDLE
The handle to the graph to set the point on.

line : INT (1..4) *default 1*
The line to set the point on.

x : INT (-1,1..240) *default -1*
The position to the entry to set the value for. When set to -1, it is put in the buffer, ready to be appended when calling guiGraphScrollNext, otherwise it is set directly on the graph. The maximum value is `x_count` from guiGraphCreate.

y : INT *default 0*
The value to set the point on the line to. If the value is outside the valid range of the graph, it will be truncated.

Returns: INT

0	- Success.
-1	- Interface is not open (see guiOpen).
-3	- Invalid argument.
-11	- The GUI API is not supported.

Declaration:

```
FUNCTION guiGraphSetLinePoint : INT;
VAR_INPUT
   graph : SYSHANDLE;
   line  : INT := 1;
   x     : INT := -1;
   y     : INT := 0;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
```

```
VAR
   rc     : INT;
   graph  : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Set line 1 at point 1 to 32
   rc := guiGraphSetLinePoint(graph := graph, line := 1, x:=1, y := 32);
   // Set line 1 at point 2 to 20
   rc := guiGraphSetLinePoint(graph := graph, line := 1, x:=2, y := 20);
   ...
END;
END_PROGRAM;
```

4.2.20.9.6.5 guiGraphSetMarker (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function controls the horizontal markers on the graph.
The markers can be used to e.g. show maximum or minimum temperatures that the readings should not exceed.

Input:

graph : SYSHANDLE
The handle to the graph to set the marker on.

marker : INT (1..4) *default 1*
The marker to update.

value : INT *default 0*
The value of the marker, determining the horizontal position.

enable : BOOL *default TRUE*
Set to TRUE to show the marker, set to FALSE to hide it.

Returns: INT

0	- Success.
-1	- Interface is not open (see guiOpen).
-3	- Invalid argument.
-11	- The GUI API is not supported.

Declaration:

```
FUNCTION guiGraphSetMarker : INT;
VAR_INPUT
   graph  : SYSHANDLE;
   marker : INT := 1;
   value  : INT := 0;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc     : INT;
   graph  : SYSHANDLE;
```

```
END_VAR;

BEGIN
   ...
   // Show a marker at 40
   rc := guiGraphSetMarker(graph := graph, marker := 1, value := 40);
   // Show a marker at -20
   rc := guiGraphSetMarker(graph := graph, marker := 2, value := -20);
   ...
END;
END_PROGRAM;
```

4.2.20.9.7 guiLabelCreate (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function creates a new label on the provided form.
Labels can be used to display text to the user, either as a description for another control, or to show e.g. status messages.

To change the text after it has been created, use guiControlSetText.

Input:

form : SYSHANDLE
A handle to the form that the label is created on.

x : SINT *default 1*
The horizontal location of the label.

y : SINT *default 1*
The vertical location of the label.

w : SINT *default 1*
The width of the label.

h : SINT *default 1*
The height of the label.

text : STRING
The text for the label to show.

tag : DINT
The tag value to store.

valign : SINT *default 0*
The vertical alignment of the text in the label.
0 - Left aligned.
1 - Centered.
2 - Right aligned.

border : SINT *default 0*
The border type for the label.
0 - No border.
1 - Thin border.

Output:

label : SYSHANDLE
The handle to the new label.

Returns: INT

0	- Success.
-1	- Interface is not open (see guiOpen).
-2	- Not enough memory / resources.
-3	- Invalid argument.
-8	- The text contains invalid characters.
-10	- Invalid location or dimension.
-11	- The GUI API is not supported.
-12	- The location is already used.

Declaration:

```
FUNCTION guiLabelCreate : INT;
VAR_INPUT
   form   : SYSHANDLE;
   x      : SINT := 1;
   y      : SINT := 1;
   w      : SINT := 1;
   h      : SINT := 1;
   text   : STRING;
   tag    : DINT;
   valign : SINT := 0;
   border : SINT := 0;
   label  : ACCESS SYSHANDLE;
END_VAR;
```

Example:

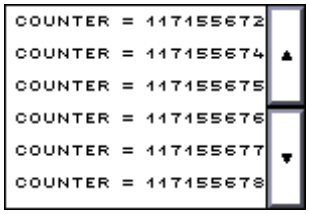
```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc    : INT;
   label : SYSHANDLE;
   form  : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Create a label with the text "Label"
   rc := guiLabelCreate(label := label, form:=form, x:=2, y:=3, w:=3, h:=1,
text := "Label", tag:=45);
   ...
END;
END_PROGRAM;
```

4.2.20.9.8 List

4.2.20.9.8.1 guiListCreate (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

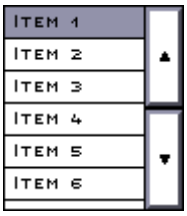
This function creates a new list control on the provided form.
List controls can be used to allow the user to select between a number or things, or to provide line based data to the user.



Input:

form : SYSHANDLE
A handle to the form that the list control is created on.

x : SINT *default 1*
The horizontal location of the list control.



y : SINT *default 1*
The vertical location of the list control.

w : SINT *default 1*
The width of the list control.

h : SINT *default 1*
The height of the list control.

tag : DINT
The tag value to store.

size : INT (1..100) *default 10*
The number of items the list will contain.

type : INT *default 1*
The type of list to create:

- 1 - A read-only list that automatically scrolls as new items are appended. Suited for e.g. showing output from an ongoing process.
- 2 - A list with selectable items. Suited for providing a list of choices to the user.

Output:

list : SYSHANDLE
The handle to the new list control.

Returns: INT

- 0 - Success.
- -1 - Interface is not open (see guiOpen).
- -2 - Not enough memory / resources.
- -3 - Invalid argument. Invalid size, type or form handle.
- -5 - Unknown error
- -10 - Invalid location or dimension.
- -11 - The GUI API is not supported.
- -12 - The location is already used.

Declaration:

```
FUNCTION guiListCreate : INT;
VAR_INPUT
   form     : SYSHANDLE;
   x        : SINT := 1;
   y        : SINT := 1;
   w        : SINT := 1;
   h        : SINT := 1;
   tag      : DINT;
   size     : INT := 10;
   type     : INT := 1;
   list     : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc     : INT;
   list   : SYSHANDLE;
   form   : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Create a scrolling list with space for 10 items
   rc := guiListCreate(list := list, form:=form, x:=1, y:=1, w:=5, h:=6, size
:= 10, type:=1);
   ...
END;
END_PROGRAM;
```

4.2.20.9.8.2 guiListSetItem (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function sets the value of an item in the list.

Input:

list : SYSHANDLE
The handle to the list to set the item in.

index : INT
The index of the item to set.

text : STRING
The text to show in the item.

Returns: INT

0	- Success.
-1	- Interface is not open (see guiOpen).
-3	- Invalid handle or index.
-8	- The text contains invalid characters.
-11	- The GUI API is not supported.

Declaration:

```
FUNCTION guiListSetItem : INT;
VAR_INPUT
   list  : SYSHANDLE;
   index : INT;
   text  : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
```

```
VAR
   rc     : INT;
   list   : SYSHANDLE;
   form   : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Create a scrolling list with space for 10 items
   rc := guiListCreate(list := list, form:=form, x:=1, y:=1, w:=5, h:=6, size
:= 10, type:=1);
   // Set item 1 to "Item 1".
   rc := guiListSetItem(list := list, index:=1, text := "Item 1");
   ...
END;
END_PROGRAM;
```

4.2.20.9.8.3 guiListAppendText (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function appends an item to the list. If the list is full, the first item will be removed and all the other items will change index, to make room for the new item.
This function is therefore best suited for use with a list of type 1, as the selected item will change index when used on a list of type 2.

Input:

list : SYSHANDLE
The handle to the list to set the item in.

text : STRING
The text to show in the item.

Returns: INT

0	- Success.
-1	- Interface is not open (see guiOpen).
-3	- Invalid handle.
-8	- The text contains invalid characters.
-11	- The GUI API is not supported.

Declaration:

```
FUNCTION guiListAppendText : INT;
VAR_INPUT
   list : SYSHANDLE;
   text : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc     : INT;
   list   : SYSHANDLE;
   form   : SYSHANDLE;
END_VAR;
```

```
BEGIN
   ...
   // Create a scrolling list with space for 10 items
   rc := guiListCreate(list := list, form:=form, x:=1, y:=1, w:=5, h:=6, size
:= 10, type:=1);
   // Set item 1 to "Item 1".
   rc := guiListSetItem(list := list, index:=1, text := "Item 1");
   // Append "Item 2" to the list.
   rc := guiListAppendText(list := list, text := "Item 2");
   ...
END;
END_PROGRAM;
```

4.2.20.9.8.4 guiListGetSelected (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function returns the index of the selected item in the list. When the selection is changed, the list selection changed event is sent.

Input:

list : SYSHANDLE
The handle to the list to find the selection in.

Returns: INT

>0	- The index of the selected item.
0	- No item selected.
-1	- Interface is not open (see guiOpen).
-3	- Invalid handle.
-11	- The GUI API is not supported.

Declaration:

```
FUNCTION guiListGetSelected : INT;
VAR_INPUT
   list  : SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc     : INT;
   list   : SYSHANDLE;
   form   : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Get selected index".
   rc := guiListGetSelected(list := list);
   IF rc > 0 THEN
      DebugFmt(message:="Selected item: \1", v1:= rc);
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.20.9.8.5 guiListSetSelected (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function sets the index of the selected item in the list.

Input:

list : SYSHANDLE
The handle to the list to set the selection on.

index : INT
The index of the item to select.

Returns: INT

0 - Success
-1 - Interface is not open (see guiOpen).
-3 - Invalid handle or index.
-11 - The GUI API is not supported.

Declaration:

```
FUNCTION guiListSetSelected : INT;
VAR_INPUT
   list  : SYSHANDLE;
   index   : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc     : INT;
   list   : SYSHANDLE;
   form   : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Select item 2
   rc := guiListSetSelected(list := list, index := 2);
   ...
END;
END_PROGRAM;
```

4.2.20.9.9 Numeric spinner

4.2.20.9.9.1 guiSpinnerCreate (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function creates a new numeric spinner on the provided form.
Numeric spinners are used to allow the user to enter a numeric value, which triggers the spinner changed event.

If the width of the spinner is three or more times larger than the height, it will show two buttons for incrementing and decrementing the value.
Clicking the button increments/decrements the value by 1, and long pressing increments/decrements the value by 100.

Input:

form : SYSHANDLE
A handle to the form that the spinner is created on.

x : SINT *default 1*
The horizontal location of the spinner.

y : SINT *default 1*
The vertical location of the spinner.

w : SINT *default 1*
The width of the spinner.

h : SINT *default 1*
The height of the spinner.

tag : DINT
The tag value to store.

value : DINT
The initial value of the spinner.

min_value : DINT *default -2147483648*
The minimum allowed value of the spinner.

max_value : DINT *default 2147483647*
The maximum allowed value of the spinner.

Output:

spinner : SYSHANDLE
The handle to the new spinner.

Returns: INT

0	- Success.
-1	- Interface is not open (see guiOpen).
-2	- Not enough memory / resources.
-3	- Invalid argument. min/max or value out of range.
-5	- Unknown error
-10	- Invalid location or dimension.
-11	- The GUI API is not supported.
-12	- The location is already used.

Declaration:

```
FUNCTION guiSpinnerCreate : INT;
VAR_INPUT
   form      : SYSHANDLE;
   x         : SINT := 1;
   y         : SINT := 1;
   w         : SINT := 1;
   h         : SINT := 1;
   tag       : DINT;
   value     : DINT;
   min_value : DINT := -2147483648;
   max_value : DINT :=  2147483647;
```

```
   spinner    : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc       : INT;
   spinner  : SYSHANDLE;
   form     : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Create a spinner for the range -10..10
   rc := guiSpinnerCreate(spinner := spinner, form:=form, x:=2, y:=3, w:=3,
h:=1,
             tag:=43, min_value := -10, max_value := 10, value := 0);
   ...
END;
END_PROGRAM;
```

4.2.20.9.9.2 guiSpinnerGetValue (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function retrieves the value of a numeric spinner.

Input:

spinner : SYSHANDLE
A handle to the numeric spinner to get the value from.

Output:

value : DINT
The current value of the spinner.

Returns: INT

```
   0    - Success.
  -1    - Interface is not open (see guiOpen).
  -3    - Invalid handle.
 -11    - The GUI API is not supported.
```

Declaration:

```
FUNCTION guiSpinnerGetValue : INT;
VAR_INPUT
   spinner : SYSHANDLE;
   value   : ACCESS DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc       : INT;
```

```
   val     : DINT;
   spinner : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Read the value
   rc := guiSpinnerGetValue(spinner := spinner, value:=val);
   ...
END;
END_PROGRAM;
```

4.2.20.9.9.3 guiSpinnerSetValue (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function sets the value of a numeric spinner.
Calling this function does not trigger the spinner changed event.

Input:

spinner : SYSHANDLE
A handle to the numeric spinner to change the value for.

value : DINT
The new value to set the spinner to.

Returns: INT

0	- Success.
-1	- Interface is not open (see guiOpen).
-3	- Invalid handle or the value is outside the range.
-11	- The GUI API is not supported.

Declaration:

```
FUNCTION guiSpinnerSetValue : INT;
VAR_INPUT
   spinner : SYSHANDLE;
   value   : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc      : INT;
   spinner : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Set the value to 5
   rc := guiSpinnerSetValue(spinner := spinner, value:=5);
   ...
END;
END_PROGRAM;
```

4.2.20.9.10 ProgressBar

4.2.20.9.10.1 guiProgressBarCreate (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function creates a new progress bar on the provided form.
Progress bars are used to show how far a process is from being completed.

Input:

form : SYSHANDLE
A handle to the form that the progress bar is created on.

x : SINT *default 1*
The horizontal location of the progress bar.

y : SINT *default 1*
The vertical location of the progress bar.

w : SINT *default 1*
The width of the progress bar.

h : SINT *default 1*
The height of the progress bar.

tag : DINT
The tag value to store.

value : DINT *default 0*
The value of the progress bar.

min_value : DINT *default 0*
The minimum value of the progress bar.

max_value : DINT *default 100*
The maximum value of the progress bar.

Output:

progress_bar : SYSHANDLE
The handle to the new progress bar.

Returns: INT

0	- Success.
-1	- Interface is not open (see guiOpen).
-2	- Not enough memory / resources.
-3	- Invalid argument. Make sure that the values are all valid: `min_value` <= `value` <= `max_value`, `min_value` <> `max_value`.
-5	- Unknown error
-10	- Invalid location or dimension.
-11	- The GUI API is not supported.
-12	- The location is already used.

Declaration:

```
FUNCTION guiProgressBarCreate : INT;
VAR_INPUT
   form         : SYSHANDLE;
```

```
   x            : SINT := 1;
   y            : SINT := 1;
   w            : SINT := 1;
   h            : SINT := 1;
   tag          : DINT;
   value        : DINT := 0;
   min_value    : DINT := 0;
   max_value    : DINT := 100;
   progress_bar : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc     : INT;
   prgbar : SYSHANDLE;
   form   : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Create a progress bar with a position of 50%
   rc := guiProgressBarCreate(progress_bar := prgbar, form:=form, x:=2, y:=3,
w:=3, h:=1, value := 50, tag:=43);
   ...
END;
END_PROGRAM;
```

4.2.20.9.10.2 guiProgressBarSetValue (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function updates the value of a progress bar.

Input:

progress_bar : SYSHANDLE
A handle to the progress bar to change the value for.

value : DINT
The new value to set the progress bar to.

Returns: INT

0	- Success.
-1	- Interface is not open (see guiOpen).
-3	- Invalid handle or the value is outside the range.
-11	- The GUI API is not supported.

Declaration:

```
FUNCTION guiProgressBarSetValue : INT;
VAR_INPUT
   progress_bar : SYSHANDLE;
   value        : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc     : INT;
   prgbar : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Set the value to 75
   rc := guiProgressBarSetValue(progress_bar := prgbar, value:=75);
   ...
END;
END_PROGRAM;
```

4.2.20.9.11 RadioButton

4.2.20.9.11.1 guiRadioButtonCreate (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function creates a new radio button on the provided form.
Radio buttons are used to select between multiple choices, where only one choice can be active at a time. Selecting a radio button triggers the radio button selected event.
Multiple groups of radio buttons can be created, where only one radio button can be enabled at a time in each group.

Input:

form : SYSHANDLE
A handle to the form that the radio button is created on.

x : SINT *default 1*
The horizontal location of the radio button.

y : SINT *default 1*
The vertical location of the radio button.

w : SINT *default 1*
The width of the radio button.

h : SINT *default 1*
The height of the radio button.

tag : DINT
The tag value to store.

group : SINT
A value that must be the same for all radio buttons in the same group.

selected : BOOL *default false*
If the radio button should start selected. If this is used on multiple radio buttons in the same group, the radio button that is initialized last will be selected.

Output:

radio_button : SYSHANDLE
The handle to the new radio button.

Returns: INT

0	- Success.
-1	- Interface is not open (see guiOpen).
-2	- Not enough memory / resources.
-3	- Invalid argument.
-5	- Unknown error
-10	- Invalid location or dimension.
-11	- The GUI API is not supported.
-12	- The location is already used.

Declaration:

```
FUNCTION guiRadioButtonCreate : INT;
VAR_INPUT
   form         : SYSHANDLE;
   x            : SINT := 1;
   y            : SINT := 1;
   w            : SINT := 1;
   h            : SINT := 1;
   tag          : DINT;
   group        : SINT := 0;
   selected     : BOOL := FALSE;
   radio_button : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc          : INT;
   radiobutton : SYSHANDLE;
   form        : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Create a radio button
   rc := guiRadioButtonCreate(radio_button := radiobutton, form:=form, x:=2,
y:=3, w:=1, h:=1, group := 0, tag:=43);
   ...
END;
END_PROGRAM;
```

4.2.20.9.11.2 guiRadioButtonIsSelected (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function returns the current selected state of the radio button.

Input:

radio_button : SYSHANDLE
A handle to the radio button to get the state of.

Returns: INT

1	- Radio button is selected.

0 - Radio button is not selected.
-1 - Interface is not open (see guiOpen).
-3 - Invalid handle.
-11 - The GUI API is not supported.

Declaration:

```
FUNCTION guiRadioButtonIsSelected : INT;
VAR_INPUT
   radio_button : SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   radio1  : SYSHANDLE;
   radio2  : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Check if radio1 is selected
   IF guiRadioButtonIsSelected(radio_button := radio1) = 1 THEN
      DebugMsg(message := "Radio 1 selected");
   END_IF;
   // Check if radio2 is selected
   IF guiRadioButtonIsSelected(radio_button := radio2) = 1 THEN
      DebugMsg(message := "Radio 2 selected");
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.20.9.11.3 guiRadioButtonSetSelected (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function makes a radio button selected, and deselects any currently selected radio button in the group.
Calling this function does not trigger the radio button selected event.

Input:

radio_button : SYSHANDLE
A handle to the radio button to select.

Returns: INT

0 - Success.
-1 - Interface is not open (see guiOpen).
-3 - Invalid handle.
-11 - The GUI API is not supported.

Declaration:

```
FUNCTION guiRadioButtonSetSelected : INT;
VAR_INPUT
   radio_button  : SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc     : INT;
   form   : SYSHANDLE;
   radio1 : SYSHANDLE;
   radio2 : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Create two radio buttons, radio2 is selected
   rc := guiRadioButtonCreate(radio_button := radio1, form:=form, x:=2, y:=3,
w:=1, h:=1,
         group := 0, tag:=43);
   rc := guiRadioButtonCreate(radio_button := radio2, form:=form, x:=2, y:=4,
w:=1, h:=1,
         group := 0, selected := TRUE, tag:=44);
   ...
   // Set radio1 as selected, deselects radio 2
   rc := guiRadioButtonSetSelected(radio_button := radio1);
   ...
END;
END_PROGRAM;
```

4.2.20.9.12 Text box

4.2.20.9.12.1 guiTextBoxCreate (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function creates a new text box on the provided form.
Text boxes are used to allow the user to enter a text string, which triggers the text box changed event.
Clicking the text box shows a dialog to change the text string, using the same keyboard as guiShowTextInput.

Input:

form : SYSHANDLE
A handle to the form that the text box is created on.

x : SINT *default 1*
The horizontal location of the text box.

y : SINT *default 1*
The vertical location of the text box.

w : SINT *default 1*
The width of the text box.

h : SINT *default 1*
The height of the text box.

tag : DINT
The tag value to store.

text : STRING
The initial value of the text box.

max_len : INT (1..254) ***default 20***
The maximum length of the string.

Output:

text_box : SYSHANDLE
The handle to the new text box.

Returns: INT

0	- Success.
-1	- Interface is not open (see guiOpen).
-2	- Not enough memory / resources.
-3	- Invalid argument.
-5	- Unknown error
-8	- The text contains invalid characters.
-10	- Invalid location or dimension.
-11	- The GUI API is not supported.
-12	- The location is already used.

Declaration:

```
FUNCTION guiTextBoxCreate : INT;
VAR_INPUT
   form     : SYSHANDLE;
   x        : SINT := 1;
   y        : SINT := 1;
   w        : SINT := 1;
   h        : SINT := 1;
   tag      : DINT;
   text     : STRING;
   max_len  : INT :=  20;
   text_box : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc      : INT;
   textbox : SYSHANDLE;
   form    : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Create a text box
   rc := guiTextBoxCreate(text_box := textbox, form:=form, x:=2, y:=3, w:=3,
h:=1, tag:=43, text := "Text", max_len := 10);
   ...
END;
END_PROGRAM;
```

4.2.20.9.12.2 guiTextBoxGetText (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function retrieves the text from a text box.

Input:

text_box : SYSHANDLE
A handle to the text box to get the text from.

Output:

text : STRING
The text in the text box.

Returns: INT

0 - Success.
-1 - Interface is not open (see guiOpen).
-3 - Invalid handle.
-8 - The text contains invalid characters.
-11 - The GUI API is not supported.

Declaration:

```
FUNCTION guiTextBoxGetText : INT;
VAR_INPUT
   text_box : SYSHANDLE;
   text     : ACCESS STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc       : INT;
   text     : STRING;
   text_box : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Read the text
   rc := guiTextBoxGetText(text_box := text_box, text:=text);
   ...
END;
END_PROGRAM;
```

4.2.20.10 Dialogs

4.2.20.10.1 guiCloseDialogs (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function closes all open dialogs, including system dialogs.

Input:
None.

Returns: INT
0 - Success.
-11 - The GUI API is not supported.

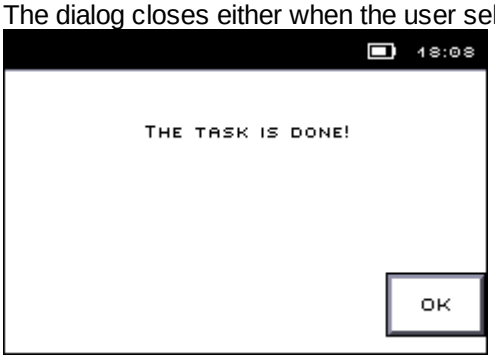
Declaration:
```
FUNCTION guiCloseDialogs : INT;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc       : INT;
END_VAR;

BEGIN
   ...
   // Closes any open dialogs
   rc := guiCloseDialogs();
   ...
END;
END_PROGRAM;
```

4.2.20.10.2 guiShowMessage (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function shows a dialog, that contains a message and a number of buttons.
The dialog closes either when the user selects one of the buttons, or when it times out.



Input:
message : STRING
The message to show to the user

timeout : INT (-1, 0..3600) ***default -1***
The number of seconds without activity before the dialog closes by itself. Use -1 for no timeout.

type : INT
The type of buttons to show:
0 - OK button
1 - OK and Cancel buttons
2 - Yes and No buttons

3 - Yes, No and Cancel buttons.

Returns: **INT**

4 - No
3 - Yes
2 - Cancel
1 - OK
0 - Timeout.
-1 - Interface is not open (see guiOpen).
-3 - Invalid parameter.
-8 - The text contains invalid characters.
-9 - A dialog is already being shown.
-11 - The GUI API is not supported.

Declaration:

```
FUNCTION guiShowMessage : INT;
VAR_INPUT
   message : STRING;
   type    : INT;
   timeout : INT := -1;
END_VAR;
```

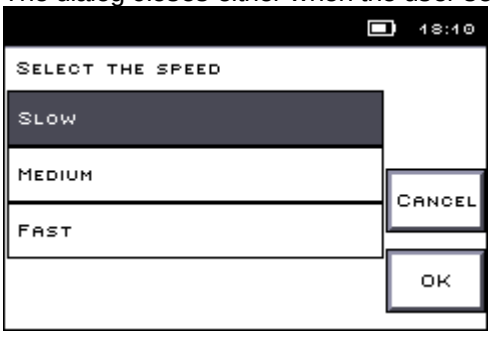
Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc      : INT;
END_VAR;

BEGIN
   ...
   // Show an OK dialog that disappears after 10 seconds
   rc := guiShowMessage(message := "The task is done!", type:=0, timeout:=10);
   ...
END;
END_PROGRAM;
```

4.2.20.10.3 guiShowChoice (Function)

Architecture: **NX32L**
Device support: **NX-400**
Firmware version: **1.00.00**

This function shows a dialog that contains a message and a number of choices.
The dialog closes either when the user selects OK or Cancel, or when it times out.



Input:

message : STRING
The message to show to the user

timeout : INT (-1, 0..3600) ***default -1***
The number of seconds without activity before the dialog closes by itself. Use -1 for no timeout.

count : INT (1..8)
The number of choices to show.

choice1 : STRING
The first choice to show.

choice2 : STRING
The second choice to show.

choice3 : STRING
The third choice to show.

choice4 : STRING
The fourth choice to show.

choice5 : STRING
The fifth choice to show.

choice6 : STRING
The sixth choice to show.

choice7 : STRING
The seventh choice to show.

choice8 : STRING
The eighth choice to show.

Output:

choice : INT
The index of the chosen choice.

Returns: INT

2	- Cancel
1	- OK
0	- Timeout.
-1	- Interface is not open (see guiOpen).
-3	- Invalid count or timeout.
-8	- The text contains invalid characters.
-9	- A dialog is already being shown.
-11	- The GUI API is not supported.

Declaration:

```
FUNCTION guiShowChoice : INT;
VAR_INPUT
   message : STRING;
   timeout : INT := -1;
   count   : INT;
   choice1 : STRING;
   choice2 : STRING;
   choice3 : STRING;
   choice4 : STRING;
   choice5 : STRING;
   choice6 : STRING;
```

```
   choice7 : STRING;
   choice8 : STRING;
   choice  : ACCESS INT;
END_VAR;
```

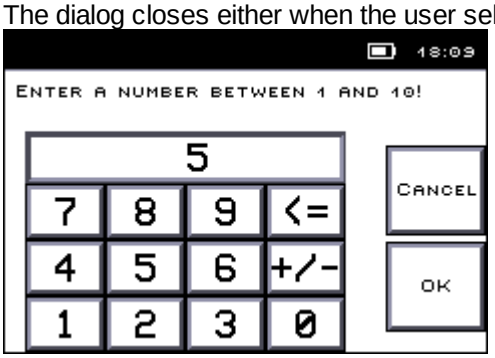
Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc      : INT;
   choice  : INT;
END_VAR;

BEGIN
   ...
   // Show a dialog to select a speed
   rc := guiShowChoice(message := "Select the speed", timeout:=10, count := 3,

      choice1 := "Slow", choice2 := "Medium", choice3 := "Fast", choice :=
choice);
   ...
END;
END_PROGRAM;
```

4.2.20.10.4 guiShowNumberInput (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function shows a dialog that shows a keypad for the user to enter a number.
The dialog closes either when the user selects OK or Cancel, or when it times out.



Input:

message : STRING
The message to show to the user

timeout : INT (-1, 0..3600) ***default -1***
The number of seconds without activity before the dialog closes by itself. Use -1 for no timeout.

min_val : DINT ***default -2147483648***
The minimum allowed value.

max_val : DINT ***default 2147483647***
The maximum allowed value.

value : DINT
The initial value of the number.

Output:

value : DINT
The new value of the number.

Returns: INT

```
 2   - Cancel
 1   - OK
 0   - Timeout
-1   - Interface is not open (see guiOpen).
-3   - Invalid timeout or min_val/max_val or value out of range..
-8   - The text contains invalid characters.
-9   - A dialog is already being shown.
-11  - The GUI API is not supported.
```

Declaration:

```
FUNCTION guiShowNumberInput : INT;
VAR_INPUT
   message : STRING;
   timeout : INT := -1;
   min_val : DINT := -2147483648;
   max_val : DINT :=  2147483647;
   value   : ACCESS DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc      : INT;
   value   : DINT;
END_VAR;

BEGIN
   ...
   // Ask the user for a number between 1 and 10
   value := 1;
   rc := guiShowNumberInput(message := "Enter a number between 1 and 10!",
timeout:=30,
      value := value, min_val := 1, max_val := 10);
   IF rc = 1 THEN
      DebugFmt(message:="Number \4 was selected", v4:=value);
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.20.10.5 guiShowTextInput (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function shows a dialog that shows a keyboard for the user to enter a text string.
The dialog closes either when the user selects OK or Cancel, or when it times out.

The buttons on the bottom of the keyboard besides the OK and space(␣) buttons are used to switch between different layouts:

- ? Click to switch between lower and upper case letters. For the symbol layout, it switches between two different sets of symbols.
- !?* Click to switch between the normal layout and the symbol layout. The symbol layout contains the numbers as well as common punctuation. The second page (accessed by pressing ?) contains currency symbols and additional symbols.
- ÅÜ Click to switch between the normal layout and the language specific letters and diacritics. To select the language to use, keep the button pressed for about 3 seconds. This shows a list of the available language groups. The selection made in this list is remembered across resets. ? switches between upper and lower case letters.

Input:

message : STRING
The message to show to the user

timeout : INT (-1, 0..3600) *default -1*
The number of seconds without activity before the dialog closes by itself. Use -1 for no timeout.

max_len : INT (-1..254)
The maximum length of the string. Use -1 to get the maximum length.

text : STRING
The initial text to show.

Output:

text : STRING
The new text.

Returns: INT

- 2 - The Back button was clicked.
- 1 - The OK button was clicked.
- 0 - Timeout.
- -1 - Interface is not open (see guiOpen).
- -3 - Invalid timeout or max_len is invalid.
- -8 - The text contains invalid characters.
- -9 - A dialog is already being shown.
- -11 - The GUI API is not supported.

Declaration:

```
FUNCTION guiShowTextInput : INT;
VAR_INPUT
   message : STRING;
   timeout : INT := -1;
   max_len : INT := -1;
   text    : ACCESS STRING;
END_VAR;
```

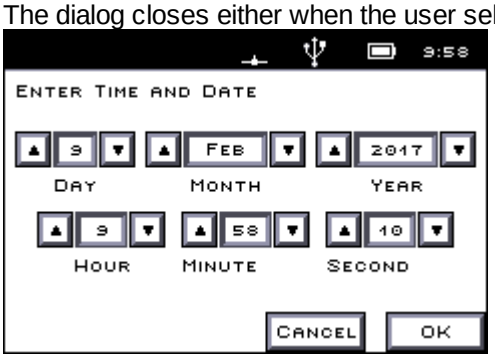
Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc   : INT;
   text : STRING;
END_VAR;

BEGIN
   ...
   // Ask the user for the password
   text := "";
   rc := guiShowTextInput(message := "Enter the password!", timeout:=60,
      text := text, max_len := 10);
   IF rc = 1 THEN
      IF text = "password" THEN
         DebugMsg(message:="Password was correct");
      ELSE
         DebugMsg(message:="Password was not correct");
      END_IF;
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.20.10.6 guiShowDateInput (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function shows a dialog that allows the user to enter a date and/or a time.
The dialog closes either when the user selects OK or Cancel, or when it times out.



Input:

message : STRING
The message to show to the user

timeout : INT (-1, 0..3600) ***default -1***
The number of seconds without activity before the dialog closes by itself. Use -1 for no timeout.

style : DINT ***default 3***
The style is a mask that determines the type of dialog to show.

Value	Style
1	- Show date
2	- Show time
4	- Make the dialog read-only

A style of 3 will show a dialog with both date and time, while a style of 6 will show a read-only time.

timestamp : DINT
Linsec for the initial time/date to show.

Output:
timestamp : DINT
The entered time and date as a linsec.

Returns: INT
2 - Cancel
1 - OK
0 - Timeout
-1 - Interface is not open (see guiOpen).
-3 - Invalid timeout, style or timestamp.
-8 - The text contains invalid characters.
-9 - A dialog is already being shown.
-11 - The GUI API is not supported.

Declaration:
```
FUNCTION guiShowDateInput : INT;
VAR_INPUT
   message   : STRING;
   timeout   : INT := -1;
   style     : DINT := 3;
   timestamp : ACCESS DINT;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc   : INT;
   ts   : DINT;
END_VAR;

BEGIN
   ...
   // Ask the user for an arrival time and date, with the current time as
starting value
   ts := clockNow();
   rc := guiShowDateInput(message := "Enter the arrival time!", timeout:=30,
style := 3,
      timestamp := ts);
   IF rc = 1 THEN
      DebugFmt(message:="Arrival time \4 was entered.", v4:=ts);
   END_IF;
   ...
END;
END_PROGRAM;
```

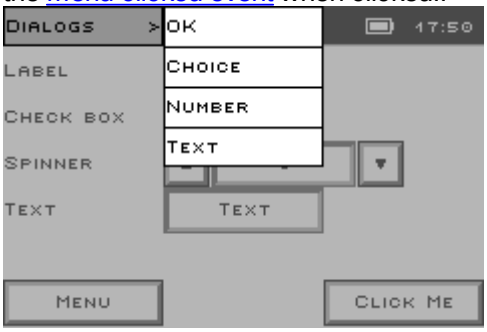
4.2.20.11 Menus

4.2.20.11.1 guiMenuCreate (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function creates a new menu.

Each menu can contain up to 8 items, which are either sub menus or menu items that triggers the menu clicked event when clicked..



Input:

tag : DINT
The tag value to store.

Output:

menu : SYSHANDLE
The handle to the new menu.

Returns: **INT**

0	- Success.
-1	- Interface is not open (see guiOpen).
-2	- Not enough memory / resources.
-11	- The GUI API is not supported.

Declaration:

```
FUNCTION guiMenuCreate : INT;
VAR_INPUT
   tag     : DINT;
   menu    : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc     : INT;
   menu   : SYSHANDLE;
END_VAR;

BEGIN
```

```
    ...
    // Create a new menu.
    rc := guiMenuCreate(menu := menu, tag:=41);
    ...
END;
END_PROGRAM;
```

4.2.20.11.2 guiMenuRemove (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function removes the menu and releases the resources.

Input:

menu : SYSHANDLE
The handle to the menu to remove. Will be made invalid once the menu is removed.

Returns: INT

0	- Success.
-1	- Interface is not open (see guiOpen).
-3	- Invalid handle.
-11	- The GUI API is not supported.

Declaration:

```
FUNCTION guiMenuRemove : INT;
VAR_INPUT
    menu   : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
    rc     : INT;
    menu   : SYSHANDLE;
END_VAR;

BEGIN
    ...
    // Remove the menu.
    rc := guiMenuRemove(menu := menu);
    ...
END;
END_PROGRAM;
```

4.2.20.11.3 guiMenuSetItem (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function sets the value of an item in the menu.
If the sub menu is supplied, then clicking the item will not trigger a menu clicked event, but instead show the sub menu, unless it would be shown outside the screen.

Input:

menu : SYSHANDLE
The handle to the menu to set the item on.

sub_menu : SYSHANDLE
A handle to a menu to use as a sub menu. If not set, the item does not have a sub menu and will cause a menu click event when clicked.

index : INT (1..8)
The index of the item to set.

title : STRING
The text to show in the item.

Returns: INT

0	- Success.
-1	- Interface is not open (see guiOpen).
-3	- Invalid handle or index.
-8	- The title contains invalid characters.
-9	- The menu is currently shown and can not be altered.
-11	- The GUI API is not supported.

Declaration:

```
FUNCTION guiMenuSetItem : INT;
VAR_INPUT
   menu     : SYSHANDLE;
   sub_menu : SYSHANDLE;
   index    : INT;
   title    : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc        : INT;
   menu      : SYSHANDLE;
   test_menu : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Create the main menu.
   rc := guiMenuCreate(menu := menu, tag := 1);
   // Create the testing menu.
   rc := guiMenuCreate(menu := test_menu, tag := 2);
   // Set the menu items.
   rc := guiMenuSetItem(menu := menu, sub_menu := test_menu, index:=1, title
:= "Test");

   rc := guiMenuSetItem(menu := test_menu, index:=1, title := "Start");

   rc := guiMenuSetItem(menu := test_menu, index:=2, title := "Stop");
   ...
END;
END_PROGRAM;
```

4.2.20.11.4 guiMenuRemoveItem (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function removes an item from the menu.

Input:

menu : SYSHANDLE
The handle to the menu to remove the item from.

index : INT (1..8)
The index of the item to remove.

Returns: INT

0	- Success.
-1	- Interface is not open (see guiOpen).
-3	- Invalid handle or index.
-9	- The menu is currently shown and can not be altered.
-11	- The GUI API is not supported.

Declaration:

```
FUNCTION guiMenuRemoveItem : INT;
VAR_INPUT
   menu    : SYSHANDLE;
   index   : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc        : INT;
   menu      : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Remove item
   rc := guiMenuRemoveItem(menu := menu, index:=1);
   ...
END;
END_PROGRAM;
```

4.2.20.11.5 guiMenuShow (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function shows the provided menu.
The menu is closed when the user selects an item, clicks outside the menu, or guiMenuClose is called.

Input:
menu : SYSHANDLE
The handle to the menu.

Returns: INT
0 - Success.
-1 - Interface is not open (see guiOpen).
-3 - Invalid handle
-9 - A menu is already visible.
-10 - No items in the menu.
-11 - The GUI API is not supported.

Declaration:
```
FUNCTION guiMenuShow : INT;
VAR_INPUT
   menu  : SYSHANDLE;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc    : INT;
   menu  : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Show the menu.
   rc := guiMenuShow(menu := menu);
   ...
END;
END_PROGRAM;
```

4.2.20.11.6 guiMenuClose (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function closes the active menu.

Input:
None.

Returns: INT
0 - Success.
-1 - Interface is not open (see guiOpen).
-3 - No active menu.
-11 - The GUI API is not supported.

Declaration:
```
FUNCTION guiMenuClose : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc    : INT;
END_VAR;

BEGIN
   ...
   // Close the current menu.
   rc := guiMenuClose();
   ...
END;
END_PROGRAM;
```

4.2.20.11.7 guiMenuClickEvent (Function)

Architecture: NX32L
Device support: NX-400
Firmware version: 1.00.00

This function retrieves a menu click event from the event queue, which is triggered when the user clicks a menu item.
This function must be called to clear event 3 from guiWaitEvent.

Input:
None.

Output:
menu : SYSHANDLE
The handle to the menu.

index : INT
The index of the clicked item.

Returns: INT
0 - Data is ready.
-1 - Interface is not open (see guiOpen).
-6 - Data is not ready
-11 - The GUI API is not supported.

Declaration:
```
FUNCTION guiMenuClickEvent : INT;
VAR_INPUT
   menu   : ACCESS SYSHANDLE;
   index  : ACCESS INT;
END_VAR;
```

Example:

```
...
FUNCTION MenuHandler
VAR
   rc     : INT;
   handle : SYSHANDLE;
   tag    : DINT := 0;
   index  : INT;
END_VAR;
```

```
   rc := guiMenuClickEvent(menu := handle, index := index);
   IF rc = 0 THEN
      IF BOOL(handle) THEN
         guiGetTag(handle := handle, tag := tag);
         DebugFmt(message:="Menu with tag \4, index \1 was clicked", v1 :=
index, v4 := tag);
      ELSE
         DebugMsg(message:="No menu was clicked");
      END_IF;
   END_IF;
END_FUNCTION;
...
```

4.2.20.12 guiSysMenuEnable (Function)

Architecture: NX32L
Device support: NX-400, LX4
Firmware version: 1.00.00

This function enables or disables the system menu.
This function is the same as displaySysMenuEnable.
This works similarly to logging in using a password, enabled with guiSysMenuSetPassword.
The system menu can be used to show the status for the device, including network status and RCH status.
On the NX-400, the system menu can also be used to change some settings including calibration of the touch screen and setting the state of the virtual jumpers.

On the NX-400, guiWaitEvent can be used to detect when the menu is enabled:
- If the system menu is enabled, the login succeeded event is sent.
- When logging out, either by clicking "Log out" in the system menu, or when logging out due to the timeout, the logged out event is sent.

On the NX-400, the enabled menu can be access by pressing on the clock shown in the top right corner of the display.
On the LX4, the menu will be shown on the display immediately and can be navigated using the arrow keys and the OK and ESC keys.
When the menu is disabled or exited on the LX4, it will close the menu and switch back to the normal contents of the display.

Input:

enable : BOOL
If true, the system menu is enabled. If false, the system menu is disabled.

timeout: INT (-1..900) ***default -1***
Number of seconds without activity before logging out automatically. 0 disables the timeout, -1 leaves it at its current value.

Returns: INT

0	- Success.
-3	- Invalid timeout
-5	- Internal error
-11	- The GUI API is not supported.

Declaration:

```
FUNCTION guiSysMenuEnable : INT;
VAR_INPUT
   enable    : BOOL;
```

```
   timeout   : INT := -1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Enable system menu with 20 second timeout.
   rc := guiSysMenuEnable(enable := TRUE, timeout := 20);
   ...
END;
END_PROGRAM;
```

4.2.20.13 guiSysMenuSetPassword (Function)

Architecture: NX32L
Device support: NX-400, LX4
Firmware version: 1.00.00

This function configures the password for entering the system menu. If it is disabled, only guiSysMenuEnable and displaySysMenuEnable can be used to enable the system menu.
This function is the same as guiSysMenuSetPassword.
To open the dialog to enter the password on the NX-400, press on the clock in the status bar for about 3 seconds and release.
To open the dialog to enter the password on the LX4, keep the OK key pressed for about 10 seconds.

On the NX-400, guiWaitEvent can be used to detect login attempts:
- If an invalid password is entered, the failed login event is sent, otherwise the login succeeded event is sent.
- When logging out, either by clicking "Log out" in the system menu, or when logging out due to the timeout set with displaySysMenuEnable, the logged out event is sent.

Input:

enable : BOOL
If password based access to the system menu should be allowed.

password : STRING
The password that must be entered on the on screen keyboard, to enable the system menu. Minimum length is 3 characters.
For the LX4, the password must only contain the numbers 0-9 and has a maximum length of 16 characters.
On the LX4, the password can also be empty, which disables the password requirement while still allowing access to the system menu by pressing OK for 10 seconds.

Returns: INT

0	- Success.
-3	- Invalid password
-5	- Internal error
-11	- The GUI API is not supported.

Declaration:

```
FUNCTION guiSysMenuSetPassword : INT;
VAR_INPUT
   enable    : BOOL;
   password  : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Enable login with password "password"
   rc := guiSysMenuSetPassword(enable := TRUE, password := "password");
   ...
END;
END_PROGRAM;
```

4.2.21 gw: RTCU Communication Hub / Gateway

4.2.21.1 gw: RTCU Communication Hub / Gateway

The RTCU Communication Hub functions are used for establishing a connection to the RTCU Communication Hub.

The RTCU Communication Hub constitutes a back-bone in the communication between the RTCU devices and the back-end applications. The product ensures reliable and economical communication in any environment and application. The RTCU Communication Hub also supports the RTCU Deployment Server. Please refer to the Users Manual for the RTCU Communication Hub for more information.

The RTCU Communication Hub has support for two types of connections:

The **standard connection** is equivalent to the connection type of the former RTCU Gateway 2 and works in the exact same way. This type of connection uses a standard TCP/IP connection with optional payload encryption.

The **Secure connection** uses the Transport Layer Security (TLS) protocol to open a connection to the RTCU Communication Hub.
The connection requires that the X509 root certificate needed to verify the server is present.
Secure connections are only supported on NX32L architecture devices.

NX32L architecture devices, running firmware 1.54.00 or later, can contain two configurations. The primary server configuration can handle both secure and standard connections while the fallback configuration only supports standard connections.

If the device fails to connect to the primary server, and a fallback server configuration is present, it will switch to using the fallback server if one of the following conditions are met:
- The connection failed because of a TLS error.
- Login to the server has failed 5 times.
- The connection has been failing for the last 12 hours.

When it switches, fault code 154 is logged in the fault log (see Fault codes for further details).

When the main system fails to boot (e.g. due to an interrupted update), and the device boots into monitor mode, it will only use the fallback configuration even if a different primary server configuration is configured.

The RTCU Communication Hub is fully compatible with the former RTCU Gateway using the 'gw' API.

• gwOpen	Opens the connection to the RTCU Gateway.
• gwClose	Closes the connection to the RTCU Gateway.
• gwIsOpen	Queries if connection to the RTCU Gateway is opened.
• gwConnected	Queries if connected to the RTCU Gateway.
• sockSetGWParm	Sets RTCU Gateway parameters.
• sockGetGWParm	Gets RTCU Gateway parameters.
• gwSuspend	Suspend gateway connection.
• gwResume	Resume gateway connection.
• gwPacketMode	Sets the mode for handling data packets.
• gwPacketSize	Query the maximum packet size for a receiver.
• gwReceivePacket	Receives a block of data over the RTCU Gateway.

- gwReceivePacketDone — Releases data buffer and sends reply to sender.
- gwSendPacket — Sends a block of data over the RTCU Gateway.
- gwSetMedia — Changes the media used to connect to the RTCU Gateway.
- gwGetMedia — Get the media used to connect to the RTCU Gateway.
- gwTimeGet — Retrieves the current time from the RTCU Gateway.
- gwEnableLPS — Controls Large Packet Support (LPS).

The 'rch' functions are only supported under the NX32L execution architecture and requires also the RTCU Communication Hub for full support.

Functions to configure the RTCU Communication Hub connection:

- rchAutoConnectGet — Queries if auto connection is enabled.
- rchAutoConnectSet — Configure auto connection.
- rchConfigGet — Queries the connection parameters.
- rchConfigSet — Set the connection parameters.
- rchFallbackGet — Queries the parameters of the fallback connection.
- rchFallbackSet — Sets the parameters of the fallback connection.
- rchHeartBeat — Manages the RTCU Communication Hub Heartbeat functionality.
- rchInterfaceSet — Changes the network interface used to connect to the RTCU Communication Hub.
- rchInterfaceGet — Get the network interface used to connect to the RTCU Communication Hub.

Autostart
Autostart is a feature where the device will open a connection to the RTCU Communication Hub automatically.
If autostart is not enabled, the connection to the RTCU Communication Hub must be opened manually with the gwOpen function.
The autostart feature can be enable/disabled for the fallback configuration with the GWenable parameter in the RTCU Communication Hub configuration (sockSetGWParm / sockGetGWParm).
The autostart feature can be enable/disabled for the primary configuration with rchAutoConnectSet.

It is still possible to use gwClose and gwOpen to change the RTCU Communication Hub configuration when autostart is used.

The device will always automatically open a connection to the RTCU Communication Hub when entering recovery mode or fault mode. On NX32L architecture devices it will try all the available interfaces, starting with the currently selected interface.

Large Packet Support
Large Packet Support (**LPS**) supports the transmission of extended size data-packets over the RTCU Communication Hub / RTCU Gateway 2.
LPS is supported by all NX32 / NX32L architecture devices.

The standard packet-size is 480 bytes and by using LPS this is increased to 4064 bytes.
Using the gwSendPacket / gwReceivePacket functions with the larger packet-size increases the actual bandwidth over especially 3G/4G networks substantially.

RTCU IDE / RACP operations such as transfer of large files or applications to/from the device automatically take advantage of the LPS functionality when available.

LPS is enabled by default, but can be disabled/enabled using gwEnableLPS.

4.2.21.2 gwOpen (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 4.60 / 1.00.00

This function will open a connection to the RTCU Communication Hub, using the configuration set with sockSetGWParm, rchConfigSet or rchFallbackSet.
Also see gwClose for closing the connection to the RTCU Communication Hub.

Input:
None.

Returns: INT
1 - Success.
0 - Not available.
-1 - The RTCU Communication Hub connection is already open.
-2 - Invalid RTCU Communication Hub configuration.

Declaration:
```
FUNCTION gwOpen : INT;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   timer    : clockDayTimer;
   rc       : INT;
END_VAR;

   // Initialize
   timer(enable:=ON, start_hour:=12, start_minute:=0, stop_hour:=13,
stop_minute:=30);

BEGIN
   // Manage RTCU Communication Hub connection
   timer();
   IF timer.q THEN
      IF NOT gwIsOpen() THEN
         DebugMsg(message := "Starting connection");
         rc := gwOpen();
         IF rc < 1 THEN
            DebugFmt(message := "  gwOpen=\1", v1 := rc);
         END_IF;
      END_IF;
   ELSE
      IF gwIsOpen() THEN
         DebugMsg(message := "Stopping connection");
         rc := gwClose();
         IF rc < 1 THEN
            DebugFmt(message := "  gwClose=\1", v1 := rc);
         END_IF;
```

```
      END_IF;
   END_IF;

   IF gwConnected() THEN
      // Connection is ready for work
   END_IF;
END;
END_PROGRAM;
```

4.2.21.3 gwClose (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 4.60 / 1.00.00

This function closes the connection to the RTCU Communication Hub.

Please also see gwOpen.

Input:

None.

Returns: INT

1 - Success.
0 - Not available.
-1 - The RTCU Communication Hub connection is not open.

Declaration:

```
FUNCTION gwClose : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   timer    : clockDayTimer;
   rc       : INT;
END_VAR;

   // Initialize
   timer(enable:=ON, start_hour:=12, start_minute:=0, stop_hour:=13,
stop_minute:=30);

BEGIN
    // Manage RTCU Communication Hub connection
   timer();
   IF timer.q THEN
      IF NOT gwIsOpen() THEN
         DebugMsg(message := "Starting connection");
         rc := gwOpen();
         IF rc < 1 THEN
            DebugFmt(message := "  gwOpen=\1", v1 := rc);
         END_IF;
      END_IF;
   ELSE
      IF gwIsOpen() THEN
         DebugMsg(message := "Stopping connection");
         rc := gwClose();
```

```
            IF rc < 1 THEN
               DebugFmt(message := "  gwClose=\1", v1 := rc);
            END_IF;
         END_IF;
      END_IF;

      IF gwConnected() THEN
         // Connection is ready for work
      END_IF;
   END;
END_PROGRAM;
```

4.2.21.4 gwIsOpen (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 4.60 / 1.00.00

This returns information about whether an RTCU Communication Hub connection is opened in the device. The function will not tell if an actual connection is finally established, to determine this the gwConnected() function must be used.

Input:
None.

Returns: BOOL
True if connection is opened, false if not.

Declaration:
```
FUNCTION gwIsOpen : BOOL;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   timer    : clockDayTimer;
   rc       : INT;
END_VAR;

   // Initialize
   timer(enable:=ON, start_hour:=12, start_minute:=0, stop_hour:=13,
stop_minute:=30);

BEGIN
   // Manage RTCU Communication Hub connection
   timer();
   IF timer.q THEN
      IF NOT gwIsOpen() THEN
         DebugMsg(message := "Starting connection");
         rc := gwOpen();
         IF rc < 1 THEN
            DebugFmt(message := "  gwOpen=\1", v1 := rc);
         END_IF;
      END_IF;
   ELSE
```

```
      IF gwIsOpen() THEN
         DebugMsg(message := "Stopping connection");
         rc := gwClose();
         IF rc < 1 THEN
            DebugFmt(message := "  gwClose=\1", v1 := rc);
         END_IF;
      END_IF;
   END_IF;

   IF gwConnected() THEN
      // Connection is ready for work
   END_IF;
END;
END_PROGRAM;
```

4.2.21.5 gwConnected (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This returns information about whether the device is connected to the RTCU Communication Hub.

Input:
None.

Returns: **BOOL**
True if connection is established, false if not.

Declaration:

```
FUNCTION gwConnected : BOOL;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   incoming : gwReceivePacket;
   Buf_in   : ARRAY [1..4064] OF SINT;
   Buf_out  : ARRAY [1..4064] OF SINT;
   size     : INT;
END_VAR;

// Turn on power to the GSM module
gsmPower(power := TRUE);
gprsOpen();

// Wait for rch connection
WHILE NOT gwConnected() DO
   Sleep(delay := 2000);
END_WHILE;

// Determine the maximum size to send
size := gwPacketSize(nodeid := 2000);
IF size = 0 THEN
```

```
   size := 480;
END_IF;

// Set address BEFORE the 'incoming' is called the first time!
incoming.buffer    := ADDR(Buf_in);
incoming.maxlength := size;

BEGIN
   ...
   // Check for incoming packets
   incoming();
   IF incoming.sender > 0 THEN
      // A packet is received
      ...
   END_IF;
   ...
   gwSendPacket(receiver := 2000, buffer := ADDR(Buf_out), length := size);
   ...
END;
END_PROGRAM;
```

4.2.21.6 gwSuspend (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 4.10 / 1.00.00

Suspends the RTCU Communication Hub communication.
This function, in combination with gwResume, can be used to control when the RTCU Communication Hub connection should be active.

Input:

None.

Returns: INT

0 - Success.
3 - RTCU Communication Hub communication is not open.

Declaration:

```
FUNCTION gwSuspend : INT;
```

Example:

```
INCLUDE rtcu.inc
PROGRAM test;
VAR
  last_hour : SINT := -1;
  clock     : clockGet;
END_VAR;
// Turn on power to the GSM module
gsmPower(power := TRUE);
gprsOpen();
// Suspend connection
gwSuspend();
BEGIN
   clock();
   // Once every hour
   IF clock.Hour <> last_hour THEN
```

```
      last_hour := clock.hour;
      // Connect for 5 minutes
      gwResume(time:=5*60);
   END_IF;

   Sleep(delay:=5000);
END;
END_PROGRAM;
```

4.2.21.7 gwResume (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 4.10 / 1.00.00

Resumes the RTCU Communication Hub communication and optionally suspends it again after a time interval.
This function, in combination with gwSuspend, can be used to control when the RTCU Communication Hub connection should be active.

Please note that changes to the RTCU Communication Hub configuration will NOT be applied when resuming the connection.
To apply a new configuration, use gwClose and gwOpen instead.

Input:

time : DINT (0..2147483647) (*Default 0*)
The number of seconds to remain enabled. Use 0 to keep the RTCU Communication Hub enabled forever.

Returns: INT

0 - Success.
3 - RTCU Communication Hub communication is not open.
4 - RTCU Communication Hub communication is not suspended.
6 - Invalid time.

Declaration:

```
FUNCTION gwResume : INT;
VAR_INPUT
   time : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
PROGRAM test;
VAR
  last_hour : SINT := -1;
  clock     : clockGet;
END_VAR;
gsmPower(power := TRUE);
gprsOpen();
// Suspend connection
gwSuspend();
BEGIN
   clock();
   // Once every hour
   IF clock.Hour <> last_hour THEN
      last_hour := clock.hour;
      // Connect for 5 minutes
      gwResume(time:=5*60);
```

```
   END_IF;

   Sleep(delay:=5000);
END;
END_PROGRAM;
```

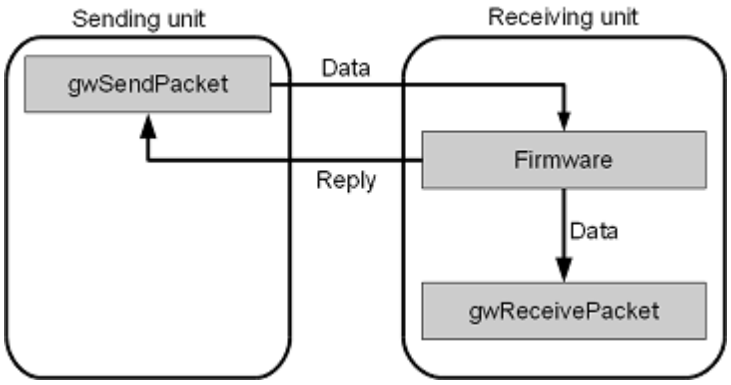
4.2.21.8 gwPacketMode (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 2.10 / 1.00.00

This function changes the mode for handling data packets received with gwReceivePacket. There are two modes:

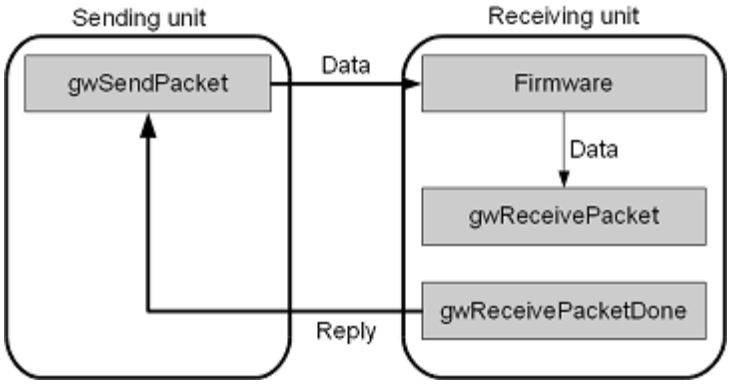
Buffered mode (default)

- gwSendPacket sends the data and waits for a reply from the receiver.
- The data packet received is placed into an internal buffer in control of the firmware. A reply is automatically returned to the sender.
- gwReceivePacket will fetch the data from the internal buffer and release it for new packets.
- gwReceivePacketDone will do nothing and is therefore not necessary to call.



Unbuffered mode

- gwSendPacket sends the data and waits for a reply from the receiver.
- The data packet is placed directly into the applications buffer, and a reply is not automatically sent back to the sender.
- gwReceivePacket will deliver the data packet to the application.
- gwReceivePacketDone will send a user-defined reply back to the sender.



Input:

mode : SINT (0/1) (*Default 0*)
The data packet receiver mode.
0 - Buffered mode.
1 - Unbuffered mode.

Returns: INT

0 - Success.
1 - Invalid parameter.

Declaration:

```
FUNCTION gwPacketMode : INT;
VAR_INPUT
   mode : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   incoming : gwReceivePacket;
   buffer   : ARRAY[1..480] OF SINT;
END_VAR;

// Set address BEFORE the 'incoming' is called the first time!
incoming.buffer    := ADDR(buffer);
incoming.maxlength := SIZEOF(buffer);

// Turn on power to the GSM module
gsmPower(power := TRUE);
gprsOpen();
gwPacketMode(mode := 1);

BEGIN
   ...
   // Check for incoming packets
   incoming();
   IF incoming.sender > 0 THEN
      // A packet is received
      ...
      gwReceivePacketDone(reply := 128);
   END_IF;
   ...
```

```
END;
END_PROGRAM;
```

4.2.21.9 gwReceivePacket (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This function block can be used to check if there is an incoming data packet from the RTCU Communication Hub.
Please also see gwPacketMode for the operating mode used when receiving a packet.

Note: if more data is received than the receive buffer can hold, the remaining data is lost.

When Large Packet Support (LPS) is enabled, packets with up to 4064 bytes can be received. (See function overview for more more information about LPS)
Use the gwPacketSize function to determine the maximum size of a packet that can be received from a sender.

Input:

buffer : PTR
The address of the receive buffer.

maxlength : INT (1..4064)
Maximum size of the data to be received (max 4064 bytes).

Output:

sender : DINT
NODE ID of sending node (0=no data available or invalid parameters).

length : INT
Length of data received (max 4064 bytes) (0=no data available or invalid parameters).

Declaration:

```
FUNCTION_BLOCK gwReceivePacket;
VAR_INPUT
   buffer    : PTR;
   maxlength : INT;
END_VAR;
VAR_OUTPUT
   sender    : DINT;
   length    : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   incoming : gwReceivePacket;
   Buf_in   : ARRAY [1..4064] OF SINT;
   Buf_out  : ARRAY [1..4064] OF SINT;
   size     : INT;
END_VAR;

// Turn on power to the GSM module
```

```
gsmPower(power := TRUE);
gprsOpen();

// Wait for hub connection
WHILE NOT gwConnected() DO
   Sleep(delay := 2000);
END_WHILE;

// Determine the maximum size to send
size := gwPacketSize(nodeid := 2000);
IF size = 0 THEN
   size := 480;
END_IF;

// Set address BEFORE the 'incoming' is called the first time!
incoming.buffer    := ADDR(Buf_in);
incoming.maxlength := size;

BEGIN
   ...
   // Check for incoming packets
   incoming();
   IF incoming.sender > 0 THEN
      // A packet is received
      ...
   END_IF;
   ...
   gwSendPacket(receiver := 2000, buffer := ADDR(Buf_out), length := size);
   ...
END;
END_PROGRAM;
```

4.2.21.10 gwReceivePacketDone (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 2.10 / 1.00.00

This function works differently depending on the mode (see gwPacketMode).

When in unbuffered mode, this function releases the data packet and sends a reply to the sender. In buffered mode, this function has no effect as the reply was already sent by the firmware.

It is very important to use this function after calling the gwReceivePacket function block. Otherwise no further data packets will be received.

Input:

reply : INT (0,128..255)
Reply code returned to the sender.
Returning 0 (zero) indicates success and values 128..255 are reserved for user-defined return codes.
Other values will result in packet rejection (return code 4).

Returns: INT

0 - Success.
1 - Not ready to receive data packets.
2 - No data packet received.

Declaration:

```
FUNCTION gwReceivePacketDone : INT;
VAR_INPUT
   reply : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   incoming : gwReceivePacket;
   buffer   : ARRAY [1..480] OF SINT;
END_VAR;

// Set address BEFORE the 'incoming' is called the first time!
incoming.buffer    := ADDR(buffer);
incoming.maxlength := SIZEOF(buffer);

gsmPower(power := TRUE);
gprsOpen();
gwPacketMode(mode := 1);

BEGIN
   ...
   // Check for incoming packets
   incoming();
   IF incoming.sender > 0 THEN
      // A packet is received
      ...
      gwReceivePacketDone(reply := 128);
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.21.11 gwSendPacket (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This function sends a packet of binary data over the RTCU Communication Hub. It will return the status of the send command. See below.

Please note that the return value from this function should always be checked. There are a number of reasons why this function could fail - too much traffic on the network for example. Therefore, always check the return value, and, based on that, have a strategy for resending a message etc.

When Large Packet Support (LPS) is enabled, packets with up to 4064 bytes can be sent.
(See function overview for more more information about LPS)
Use the gwPacketSize function to determine the maximum size of a packet that can be sent to a receiver.

Input:

receiver : DINT
NODE ID of receiver node.

buffer : PTR
The address of the receive buffer.

length : INT (1..4064)
Length of data to send (max 4064 bytes).

Returns: INT

0 - Success.
1 - Destination unreachable.
2 - Timeout delivering packet.
3 - Communication channel not open/connected.
4 - Packet rejected by receiver.
5 - Unspecified error.
6 - Invalid parameter.
7 - The length of the data is too large for the receiver.
8 - Destination is busy.

Note: 128 - 255 is reserved for user-defined return codes from gwReceivePacketDone.

Declaration:

```
FUNCTION gwSendPacket : INT;
VAR_INPUT
   receiver : DINT;
   buffer   : PTR;
   length   : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   incoming : gwReceivePacket;
   Buf_in   : ARRAY [1..4064] OF SINT;
   Buf_out  : ARRAY [1..4064] OF SINT;
   size     : INT;
END_VAR;

// Turn on power to the GSM module
gsmPower(power := TRUE);
gprsOpen();

// Wait for rch connection
WHILE NOT gwConnected() DO
   Sleep(delay := 2000);
END_WHILE;

// Determine the maximum size to send
size := gwPacketSize(nodeid := 2000);
IF size = 0 THEN
   size := 480;
END_IF;

// Set address BEFORE the 'incoming' is called the first time!
incoming.buffer    := ADDR(Buf_in);
incoming.maxlength := size;

BEGIN
```

```
   ...
   // Check for incoming packets
   incoming();
   IF incoming.sender > 0 THEN
      // A packet is received
      ...
   END_IF;
   ...
   gwSendPacket(receiver := 2000, buffer := ADDR(Buf_out), length := size);
   ...
END;
END_PROGRAM;
```

4.2.21.112 gwPacketSize (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 4.40 / 1.00.00

This function is a part of the Large Packet Support (LPS), and will return the maximum size of a packet that can be sent/received from a peer node connected to the RTCU Communication Hub.

Using this function it is possible to determine the maximum size of data that can be sent/received using gwSendPacket and gwReceivePacket.

Calling this function may submit a request to the RTCU Communication Hub to get the information on the peer node data size.
It is therefore recommended that this function is only called once for each peer node.

The function requires that the device is connected to the RTCU Communication Hub.

Input:

nodeid : DINT
The NODE ID of the node.

Returns: INT

The maximum number of bytes that can be send to, or received from, the specified node.

Declaration:

```
FUNCTION gwPacketSize : INT;
VAR_INPUT
   nodeid : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   incoming : gwReceivePacket;
   Buf_in   : ARRAY [1..4064] OF SINT;
   Buf_out  : ARRAY [1..4064] OF SINT;
   size     : INT;
END_VAR;

gsmPower(power := TRUE);
```

```
gprsOpen();

// Wait for hub connection
WHILE NOT gwConnected() DO
   Sleep(delay := 2000);
END_WHILE;

// Determine the maximum size to send
size := gwPacketSize(nodeid := 2000);
IF size = 0 THEN
   size := 480;
END_IF;

// Set address BEFORE the 'incoming' is called the first time!
incoming.buffer    := ADDR(Buf_in);
incoming.maxlength := size;

BEGIN
   ...
   // Check for incoming packets
   incoming();
   IF incoming.sender > 0 THEN
      // A packet is received
      ...
   END_IF;
   ...
   gwSendPacket(receiver := 2000, buffer := ADDR(Buf_out), length := size);
   ...
END;
END_PROGRAM;
```

4.2.21.113 gwEnableLPS (Function)

Architecture: NX32 / NX32L
Device support: All
Firmware version: 4.40 / 1.00.00

This function controls whether Large Packet Support (LPS) is used when communicating over the RTCU Communication Hub.

The mode selected by this function takes effect when opening a RTCU Communication Hub session with gwOpen, netOpen or gprsOpen.
To make the change take effect immediately then any active RTCU Communication Hub session must be terminated/suspended, using gwSuspend or gwClose.

Input:

enable : BOOL
Enable or disable Large Packet Support (LPS).

Returns: BOOL

TRUE - Success.
FALSE - Not supported.

Declaration:

```
FUNCTION gwEnableLPS : BOOL;
VAR_INPUT
```

```
   enable : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

gsmPower(power := TRUE);

// Disable Large Packet Support
gwEnableLPS(enable := OFF);

netOpen(iface:=1);
   ...

BEGIN
   ...
END;
END_PROGRAM;
```

4.2.21.14 gwSetMedia (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.10 / 1.00.00

This function temporarily changes the media used to connect to the RTCU Communication Hub. This function works similarly to rchInterfaceSet.

The default media is Mobile.
In case the device faults, it will try to connect to the RTCU Communication Hub using the media selected by this function.

If a non-fallback server configuration is enabled, this function must be called after gwOpen to work.

Input:

media : SINT (0/1) (*Default 0*)
Media to use for connecting to the RTCU Communication Hub.
- 0 - Mobile network
- 1 - Local network media (Ethernet).
- 2 - Local network media (USB Ethernet)
- 3 - Wireless network media (built-in WLAN or USB WLAN)

Returns: INT

- 0 - Success.
- 1 - Media not found.

Declaration:

```
FUNCTION gwSetMedia : INT;
VAR_INPUT
   media : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   eth : BOOL;
END_VAR;

VAR_OUTPUT
   LED_GPRS : BOOL;
   LED_GW   : BOOL;
END_VAR;

PROGRAM EthernetTest;
VAR
   gw_media : SINT;
END_VAR;

   // Init GPRS
   gsmPower(power:=ON);
   gprsOpen();

   // Init Ethernet
   ethOpen();

   DebugMsg(message:="Program running");

BEGIN
   LED_GPRS := gprsConnected();
   LED_GW   := gwConnected();

   // Change gateway media?
   IF gw_media = 0 AND eth THEN
      // Select media
      gwSetMedia(media:=1);  // Use ethernet
      // Set curent media
      gw_media := 1;
   ELSIF gw_media = 1 AND NOT eth THEN
      // Select media
      gwSetMedia(media:=0);  // Use GPRS
      // Set curent media
      gw_media := 0;
  END_IF;

END;
END_PROGRAM;
```

4.2.21.15 gwGetMedia (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 5.12 / 1.62.00

This function reads the media used to connect to the RTCU Communication Hub.
This function works similarly to rchInterfaceGet.

Output:

media : SINT
Media to use for connecting to the RTCU Communication Hub.
0 - Mobile network

1 - Local network media (Ethernet).
2 - Local network media (USB Ethernet)
3 - Wireless network media (built-in WLAN or USB WLAN)

Returns: INT

1 - Success
0 - Not supported
-1 - Error reading media

Declaration:

```
FUNCTION gwGetMedia : INT;
VAR_INPUT
   media : ACCESS SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
  eth : BOOL;
END_VAR;

VAR_OUTPUT
  LED_GPRS : BOOL;
  LED_GW   : BOOL;
END_VAR;

PROGRAM EthernetTest;
VAR
  gw_media : SINT;
END_VAR;

  // Init GPRS
  gsmPower(power := ON);
  gprsOpen();

  // Init Ethernet
  ethOpen();

  DebugMsg(message := "Program running");

BEGIN
  LED_GPRS := gprsConnected();
  LED_GW   := gwConnected();

  // Change gateway media?
  gwGetMedia(media := gw_media);
  IF gw_media = 0 AND eth THEN
    // Select media
    gwSetMedia(media := 1); // Use ethernet
  ELSIF gw_media = 1 AND NOT eth THEN
    // Select media
    gwSetMedia(media := 0); // Use GPRS
  END_IF;

END;
END_PROGRAM;
```

4.2.21.16 gwTimeGet (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.50 / 1.00.00

This function requests the current time, expressed as seconds since 1980-1-1 00:00:00 (LINSEC), from the RTCU Communication Hub.
The time requested can be either UTC or the Local time of the server.

Note: the time service functionality must be available and enabled in the connected RTCU Communication Hub.

Input:

UTC : BOOL (*Default FALSE*)
The format of the time fetched from the RTCU Communication Hub.
TRUE: Requests the UTC time.
FALSE: Requests the Local time.

Returns: DINT

The current time as seconds since 1980-1-1 00:00:00 (LINSEC).
0 - Error retrieving the time.

Declaration:

```
FUNCTION gwTimeGet : DINT;
VAR_INPUT
   UTC : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   update : BOOL;
END_VAR;

PROGRAM EthernetTest;
VAR
   upd_old : BOOL;
   time    : DINT;
END_VAR;

   // Init mobile network
   gsmPower(power:=ON);
   gprsOpen();

   DebugMsg(message:="Program running");

BEGIN
   ...
   // Update device clock
   IF update AND NOT upd_old THEN
      // Get time
      time := gwTimeGet(UTC := TRUE);
      // Set curent time
      IF time > 0 THEN
```

```
            clockSet(linsec := time);
         END_IF;
      END_IF;
      upd_old := update;
END;
END_PROGRAM;
```

4.2.21.17 sockGetGWParm (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.21 / 1.00.00

---

sockGetGWParm will read out the RTCU Communication Hub Parameters previously set from the RTCU IDE or by using the sockSetGWParm function.
On NX32L architecture devices this will read the fallback configuration, see also rchFallbackGet.

Input:

None

Output:

GWEnabled : BOOL
Determines whether the RTCU Communication Hub service will start automatically.
(See Autostart feature)

GWIP : STRING
The IP address or symbolic name of the RTCU Communication Hub.

GWPort : DINT
The IP port the device should use to connect to the RTCU Communication Hub.

GWKey : STRING
The key (password) the device should use to connect to the RTCU Communication Hub.

MaxConnectionAttempt : INT
Max number of connection attempts before the network media reconnects.

MaxSendReqAttempt : INT
Max number of send-request attempts before send fails.

ResponseTimeout : INT
Time waiting for response in seconds.

AliveFreq : DINT
Frequency for sending self-transactions in seconds.

CryptKey[n] : SINT
The key used to encrypt communication with the RTCU Communication Hub.

Declaration:

```
FUNCTION_BLOCK sockGetGWParm;
VAR_OUTPUT
   //RTCU Communication Hub parameters:
   GWEnabled            : BOOL;
   GWIP                 : STRING;
   GWPort               : DINT;
   GWKey                : STRING;
```

```
   //advanced settings:
   MaxConnectionAttempt : INT;
   MaxSendReqAttempt    : INT;
   ResponseTimeout      : INT;
   AliveFreq            : DINT;
   CryptKey             : ARRAY[1..16] OF SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   gprsParm  : sockGetTCPIPParm;
   gwParm    : sockGetGWParm;
   parmReset : BOOL := FALSE;
   CryptKey  : ARRAY[1..16] OF SINT;
END_VAR;

// Check mobile network and Hub settings
gprsParm();
gwParm();
IF gprsParm.IP <> 0 OR gprsParm.SubnetMask <> 0 OR gprsParm.Gateway <> 0 OR
   gprsParm.DNS1 <> 0 OR gprsParm.DNS2 <> 0 OR gprsParm.Authenticate <> 0 OR
   strCompare(str1 := gprsParm.Username, str2 := "") <> 0 OR
   strCompare(str1 := gprsParm.Password, str2 := "") <> 0 OR
   strCompare(str1 := gprsParm.APN, str2 := "internet") <> 0 THEN
   // Set APN and keep the default values for the others
   sockSetTCPIPParm(APN := "internet");
   parmReset := TRUE;
END_IF;
IF NOT gwParm.GWEnabled OR gwParm.GWPort <> 5001 OR gwParm.CryptKey[1] <> 0 OR
   gwParm.CryptKey[2] <> 0 OR gwParm.CryptKey[3] <> 0 OR gwParm.CryptKey[4] <>
 0 OR
   gwParm.CryptKey[5] <> 0 OR gwParm.CryptKey[6] <> 0 OR gwParm.CryptKey[7] <>
 0 OR
   gwParm.CryptKey[8] <> 0 OR gwParm.CryptKey[9] <> 0 OR gwParm.CryptKey[10]
<> 0 OR
   gwParm.CryptKey[11] <> 0 OR gwParm.CryptKey[12] <> 0 OR gwParm.CryptKey[13]
 <> 0 OR
   gwParm.CryptKey[14] <> 0 OR gwParm.CryptKey[15] <> 0 OR gwParm.CryptKey[16]
 <> 0 OR
   strCompare(str1 := gwParm.GWIP, str2 := "gw.rtcu.dk") <> 0 OR
   strCompare(str1 := gwParm.GWKey, str2 := "AABBCCDD") <> 0 THEN
   // Clear encryption key
   CryptKey[1]  := 0;
   CryptKey[2]  := 0;
   CryptKey[3]  := 0;
   CryptKey[4]  := 0;
   CryptKey[5]  := 0;
   CryptKey[6]  := 0;
   CryptKey[7]  := 0;
   CryptKey[8]  := 0;
   CryptKey[9]  := 0;
   CryptKey[10] := 0;
   CryptKey[11] := 0;
   CryptKey[12] := 0;
   CryptKey[13] := 0;
   CryptKey[14] := 0;
   CryptKey[15] := 0;
   CryptKey[16] := 0;
```

```
    // Set Gateway parameters
    sockSetGWParm(GWEnabled := TRUE, GWIP := "gw.rtcu.dk", GWPort := 5001,
GWKey := "AABBCCDD", CryptKey := ADDR(CryptKey));
    parmReset := TRUE;
END_IF;
IF parmReset THEN
    // Reset device before the changes are used
    boardReset();
END_IF;

gsmPower(power := TRUE);
gprsOpen();

BEGIN
    ...
END;
END_PROGRAM;
```

4.2.21.18 sockSetGWParm (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

Sets the RTCU Communication Hub specific parameters.
This has the same effect as setting the parameters via Device: Network: RTCU Communication Hub settings.
The changes will take effect at the next call to gwOpen or at device reset. (e.g. by using boardReset).
On NX32L architecture devices this will set the fallback configuration, see also rchFallbackSet.

Input:

GWEnabled : BOOL
Determines whether the RTCU Communication Hub will start automatically.
(See Autostart feature)

GWIP : STRING (*Max 41 characters*)
The IP address or symbolic name of the RTCU Communication Hub.

GWPort : DINT *(default 5001)*
The IP port the device should use to connect to the RTCU Communication Hub.

GWKey : STRING (*Max 8 characters*)
The key (password) the device should use to connect to the RTCU Communication Hub.

MaxConnectionAttempt : INT *(default 3)*
Max number of connection attempts before the network media reconnects.

MaxSendReqAttempt : INT *(default 5)*
Max number of send-request attempt before send fails.

ResponseTimeout : INT *(default 45)*
Time waiting for response in seconds.

AliveFreq : DINT *(default 300)*
Frequency for sending self-transactions in seconds.
The purpose of the self-transaction is to ensure a healthy two-way communication channel over the RTCU Communication Hub.

For applications that are only sending data from the device to the server, this frequency can safely be increased.
Setting the value to zero will disable the self-transactions completely.

CryptKey : PTR
The key used to encrypt communication with the RTCU Communication Hub.
If this parameter is not set (CryptKey set to 0), then the encryption key is not changed.

Returns: INT

0 - Success.
1 - String too long.

Declaration:

```
FUNCTION sockSetGWParm;
VAR_INPUT
   //RTCU Hub parameters:
   GWEnabled            : BOOL;         // false=disabled, true=enable RCH
   GWIP                 : STRING;       // RCHaddress (dotted / symbolic) (max
41 chars)
   GWPort               : DINT := 5001; // RCH port (def: 5001)
   GWKey                : STRING;       // RCH key, 8 characters

   //advanced settings (modification not recommended):
   MaxConnectionAttempt : INT := 3;     // Max number of connection attempts
before the network media re-connects.
                                        // Interval: 1..60. (default: 3)
   MaxSendReqAttempt    : INT := 5;     // Max number of send-request attempt
before send fails.
                                        // Interval: 1..60 (default: 5)
   ResponseTimeout      : INT := 45;    // Time waiting for response in
seconds
                                        // Interval: 5..60 (def: 45 seconds).
   AliveFreq            : DINT := 300;  // Frequency for sending self-
transactions in seconds
                                        // Interval: 0..60000 (def: 300
seconds).
                                        // 0 (zero) will disable self-trans.
   CryptKey             : PTR;          // Pointer to array containing 16
SINT.
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   gprsParm  : sockGetTCPIPParm;
   gwParm    : sockGetGWParm;
   parmReset : BOOL := FALSE;
   CryptKey  : ARRAY[1..16] OF SINT;
END_VAR;

// Check mobile network and hub settings
gprsParm();
gwParm();
IF gprsParm.IP <> 0 OR gprsParm.SubnetMask <> 0 OR gprsParm.Gateway <> 0 OR
   gprsParm.DNS1 <> 0 OR gprsParm.DNS2 <> 0 OR gprsParm.Authenticate <> 0 OR
   strCompare(str1 := gprsParm.Username, str2 := "") <> 0 OR
```

```
   strCompare(str1 := gprsParm.Password, str2 := "") <> 0 OR
   strCompare(str1 := gprsParm.APN, str2 := "internet") <> 0 THEN
   // Set APN and keep the default values for the others
   sockSetTCPIPParm(APN := "internet");
   parmReset := TRUE;
END_IF;
IF NOT gwParm.GWEnabled OR gwParm.GWPort <> 5001 OR gwParm.CryptKey[1] <> 0 OR
   gwParm.CryptKey[2] <> 0 OR gwParm.CryptKey[3] <> 0 OR gwParm.CryptKey[4] <>
 0 OR
   gwParm.CryptKey[5] <> 0 OR gwParm.CryptKey[6] <> 0 OR gwParm.CryptKey[7] <>
 0 OR
   gwParm.CryptKey[8] <> 0 OR gwParm.CryptKey[9] <> 0 OR gwParm.CryptKey[10]
<> 0 OR
   gwParm.CryptKey[11] <> 0 OR gwParm.CryptKey[12] <> 0 OR gwParm.CryptKey[13]
 <> 0 OR
   gwParm.CryptKey[14] <> 0 OR gwParm.CryptKey[15] <> 0 OR gwParm.CryptKey[16]
 <> 0 OR
   strCompare(str1 := gwParm.GWIP, str2 := "gw.rtcu.dk") <> 0 OR
   strCompare(str1 := gwParm.GWKey, str2 := "AABBCCDD") <> 0 THEN
   // Clear encryption key
   CryptKey[1]  := 0;
   CryptKey[2]  := 0;
   CryptKey[3]  := 0;
   CryptKey[4]  := 0;
   CryptKey[5]  := 0;
   CryptKey[6]  := 0;
   CryptKey[7]  := 0;
   CryptKey[8]  := 0;
   CryptKey[9]  := 0;
   CryptKey[10] := 0;
   CryptKey[11] := 0;
   CryptKey[12] := 0;
   CryptKey[13] := 0;
   CryptKey[14] := 0;
   CryptKey[15] := 0;
   CryptKey[16] := 0;
   // Set hub parameters
   sockSetGWParm(GWEnabled := TRUE, GWIP := "gw.rtcu.dk", GWPort := 5001,
GWKey := "AABBCCDD", CryptKey := ADDR(CryptKey));
   parmReset := TRUE;
END_IF;
IF parmReset THEN
   // Reset device before the changes are used
   boardReset();
END_IF;

gsmPower(power := TRUE);
gprsOpen();

BEGIN
   ...
END;
END_PROGRAM;
```

4.2.21.19 rchAutoConnectGet (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.54.00

This function will query the primary server configuration to determine if the connection to the RTCU Communication Hub will be opened automatically when the network interface is opened.

To query the fallback configuration use rchFallbackGet or SockGetGWParm.

Input:
None

Returns: BOOL
TRUE if the connection will be opened, FALSE if not.

Declaration:
```
FUNCTION rchAutoConnectGet : BOOL;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   parm  : rchConfigGet;
   reset : BOOL := FALSE;
END_VAR;

// Check the RTCU Communication Hub settings
parm();
IF NOT rchAutoConnectGet() THEN
   rchAutoConnectSet(enable := ON);
   reset := TRUE;
END_IF;
IF NOT parm.enabled OR parm.secure OR parm.port <> 5001 OR
   parm.CryptKey[1] <> 0 OR parm.CryptKey[2] <> 0 OR
   parm.CryptKey[3] <> 0 OR parm.CryptKey[4] <> 0 OR
   parm.CryptKey[5] <> 0 OR parm.CryptKey[6] <> 0 OR
   parm.CryptKey[7] <> 0 OR parm.CryptKey[8] <> 0 OR
   parm.CryptKey[9] <> 0 OR parm.CryptKey[10] <> 0 OR
   parm.CryptKey[11] <> 0 OR parm.CryptKey[12] <> 0 OR
   parm.CryptKey[13] <> 0 OR parm.CryptKey[14] <> 0 OR
   parm.CryptKey[15] <> 0 OR parm.CryptKey[16] <> 0 OR
   parm.iface <> 2 OR strCompare(str1 := parm.address, str2 := "gw.rtcu.dk")
<> 0 OR
   strCompare(str1 := parm.loginkey, str2 := "AABBCCDD") <> 0 THEN
   // Update settings
   rchConfigSet(enable := TRUE,
                address := "gw.rtcu.dk",
                port := 5001,
                iface := 2,
                loginkey := "AABBCCDD",
                secure := FALSE);
   reset := TRUE;
END_IF;
IF reset THEN
   // Reset device before the changes are used
   boardReset();
END_IF;

// Start network
netOpen(iface := 2);

BEGIN
END;
END_PROGRAM;
```

4.2.21.20 rchAutoConnectSet (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.54.00

This function controls whether a connection to the RTCU Communication Hub will be opened automatically using the primary server configuration.

The changes will take effect at device reset. (e.g. by using boardReset).
To change the fallback configuration, use rchFallbackSet or SockSetGWParm.

Input:

enable : BOOL *(default TRUE)*
Enable or disable auto connect.

Returns: INT

1 - Success.
0 - Not supported.

Declaration:

```
FUNCTION rchAutoConnectSet : INT;
VAR_INPUT
   enable : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   parm  : rchConfigGet;
   reset : BOOL := FALSE;
END_VAR;

// Check the RTCU Communication Hub settings
parm();
IF NOT rchAutoConnectGet() THEN
   rchAutoConnectSet(enable := ON);
   reset := TRUE;
END_IF;
IF NOT parm.enabled OR parm.secure OR parm.port <> 5001 OR
   parm.CryptKey[1] <> 0 OR parm.CryptKey[2] <> 0 OR
   parm.CryptKey[3] <> 0 OR parm.CryptKey[4] <> 0 OR
   parm.CryptKey[5] <> 0 OR parm.CryptKey[6] <> 0 OR
   parm.CryptKey[7] <> 0 OR parm.CryptKey[8] <> 0 OR
   parm.CryptKey[9] <> 0 OR parm.CryptKey[10] <> 0 OR
   parm.CryptKey[11] <> 0 OR parm.CryptKey[12] <> 0 OR
   parm.CryptKey[13] <> 0 OR parm.CryptKey[14] <> 0 OR
   parm.CryptKey[15] <> 0 OR parm.CryptKey[16] <> 0 OR
   parm.iface <> 2 OR strCompare(str1 := parm.address, str2 := "gw.rtcu.dk")
<> 0 OR
   strCompare(str1 := parm.loginkey, str2 := "AABBCCDD") <> 0 THEN
   // Update settings
   rchConfigSet(enable := TRUE,
                address := "gw.rtcu.dk",
                port := 5001,
```

```
                iface := 2,
                loginkey := "AABBCCDD",
                secure := FALSE);
   reset := TRUE;
END_IF;
IF reset THEN
   // Reset device before the changes are used
   boardReset();
END_IF;

// Start network
netOpen(iface := 2);

BEGIN
END;
END_PROGRAM;
```

4.2.21.21 rchConfigGet (Functionblock)

Architecture: NX32L
Device support: All
Firmware version: 1.54.00

---

rchConfigGet will read out the RTCU Communication Hub parameters previously set from the RTCU IDE, or the rchConfigSet function.

Input:

None

Output:

enabled : BOOL
Whether the parameters are valid..

address : STRING
The IP address or symbolic name of the server.

port : DINT
The IP port the device should use to connect to the server.

iface : SINT
The network interface the device should use to connect to the server.

secure : BOOL
Whether a standard or secure connection is to be used.

loginkey : STRING
The key (password) the device should use to connect to the server.

cryptkey[n] : SINT
The key used to encrypt communication on a standard connection to the server.

MaxConnectionAttempt : INT
Max number of connection attempts before the network media reconnects.

MaxSendReqAttempt : INT
Max number of send-request attempts before send fails.

ResponseTimeout : INT
Time waiting for response in seconds.

AliveFreq : DINT
Frequency for sending self-transactions in seconds.

Declaration:

```
FUNCTION_BLOCK rchConfigGet;
VAR_OUTPUT
   enabled              : BOOL;
   address              : STRING;
   port                 : DINT;
   iface                : SINT;
   secure               : BOOL;
   loginkey             : STRING;
   cryptkey             : ARRAY [1..16] OF SINT;
   MaxConnectionAttempt : INT;
   MaxSendReqAttempt    : INT;
   ResponseTimeout      : INT;
   AliveFreq            : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   parm  : rchConfigGet;
   reset : BOOL := FALSE;
END_VAR;

// Check the RTCU Communication Hub settings
parm();
IF NOT rchAutoConnectGet() THEN
   rchAutoConnectSet(enable := ON);
   reset := TRUE;
END_IF;
IF NOT parm.enabled OR parm.secure OR parm.port <> 5001 OR
   parm.CryptKey[1] <> 0 OR parm.CryptKey[2] <> 0 OR
   parm.CryptKey[3] <> 0 OR parm.CryptKey[4] <> 0 OR
   parm.CryptKey[5] <> 0 OR parm.CryptKey[6] <> 0 OR
   parm.CryptKey[7] <> 0 OR parm.CryptKey[8] <> 0 OR
   parm.CryptKey[9] <> 0 OR parm.CryptKey[10] <> 0 OR
   parm.CryptKey[11] <> 0 OR parm.CryptKey[12] <> 0 OR
   parm.CryptKey[13] <> 0 OR parm.CryptKey[14] <> 0 OR
   parm.CryptKey[15] <> 0 OR parm.CryptKey[16] <> 0 OR
   parm.iface <> 2 OR strCompare(str1 := parm.address, str2 := "gw.rtcu.dk")
<> 0 OR
   strCompare(str1 := parm.loginkey, str2 := "AABBCCDD") <> 0 THEN
   // Update settings
   rchConfigSet(enable := TRUE,
                address := "gw.rtcu.dk",
                port := 5001,
                iface := 2,
                loginkey := "AABBCCDD",
                secure := FALSE);
   reset := TRUE;
END_IF;
IF reset THEN
   // Reset device before the changes are used
```

```
   boardReset();
END_IF;

// Start network
netOpen(iface := 2);

BEGIN
END;
END_PROGRAM;
```

4.2.21.22 rchConfigSet (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.54.00

Sets the parameters for making connections to the RTCU Communication Hub. This has the same effect as setting the parameters via Device: Network: RTCU Communication Hub settings. The changes will take effect at the next call to gwOpen or at device reset. (e.g. by using boardReset).

Input:

enable : BOOL ***(Default TRUE)***
TRUE to set the parameters, FALSE to clear the parameters.

address : STRING (*Max 50 characters*)
The IP address or symbolic name of the server.

port : DINT ***(default 5001)***
The IP port the device should use to connect to the server.

iface : SINT ***(default 1)***
The network interface the device should use to connect to the server.

secure : BOOL ***(Default TRUE)***
Control if a standard connection or secure connection should be used to connect to the server.

port : DINT ***(default 5001)***
The IP port the device should use to connect to the server.

loginkey : STRING (*Max 8 characters*)
The key (password) the device should use to connect to the server.

MaxConnectionAttempt : INT ***(1..60, default 3)***
Max number of connection attempts before the network media reconnects.

MaxSendReqAttempt : INT ***(1..60, default 5)***
Max number of send-request attempts before send fails.

ResponseTimeout : INT ***(5..60, default 45)***
Time waiting for response in seconds.

AliveFreq : DINT ***(0..60000, default 300)***
Frequency for sending self-transactions in seconds.
The purpose of the self-transaction is to ensure a healthy two-way communication channel over the server.

For applications that are only sending data from the device to the server, this frequency can safely be increased.
Setting the value to zero will disable the self-transactions completely.

cryptkey : PTR
The key used to encrypt communication with the server using a standard connection.
If this parameter is not set (CryptKey set to 0), then the encryption key is not changed.

Returns: INT

1 - Success.
0 - Not supported.
-2 - Invalid network interface.
-3 - Invalid parameter.

Declaration:

```
FUNCTION rchConfigSet : INT;
VAR_INPUT
   enable               : BOOL;
   address              : STRING;
   port                 : DINT := 5001;
   iface                : SINT := 1;
   secure               : BOOL := TRUE;
   loginkey             : STRING;
   cryptkey             : PTR;
   MaxConnectionAttempt : INT := 3;
   MaxSendReqAttempt    : INT := 5;
   ResponseTimeout      : INT := 45;
   AliveFreq            : DINT := 300;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   parm  : rchConfigGet;
   reset : BOOL := FALSE;
END_VAR;

// Check the RTCU Communication Hub settings
parm();
IF NOT rchAutoConnectGet() THEN
   rchAutoConnectSet(enable := ON);
   reset := TRUE;
END_IF;
IF NOT parm.enabled OR parm.secure OR parm.port <> 5001 OR
   parm.CryptKey[1] <> 0 OR parm.CryptKey[2] <> 0 OR
   parm.CryptKey[3] <> 0 OR parm.CryptKey[4] <> 0 OR
   parm.CryptKey[5] <> 0 OR parm.CryptKey[6] <> 0 OR
   parm.CryptKey[7] <> 0 OR parm.CryptKey[8] <> 0 OR
   parm.CryptKey[9] <> 0 OR parm.CryptKey[10] <> 0 OR
   parm.CryptKey[11] <> 0 OR parm.CryptKey[12] <> 0 OR
   parm.CryptKey[13] <> 0 OR parm.CryptKey[14] <> 0 OR
   parm.CryptKey[15] <> 0 OR parm.CryptKey[16] <> 0 OR
   parm.iface <> 2 OR strCompare(str1 := parm.address, str2 := "gw.rtcu.dk")
<> 0 OR
   strCompare(str1 := parm.loginkey, str2 := "AABBCCDD") <> 0 THEN
   // Update settings
```

```
   rchConfigSet(enable := TRUE,
                address := "gw.rtcu.dk",
                port := 5001,
                iface := 2,
                loginkey := "AABBCCDD",
                secure := FALSE);
   reset := TRUE;
END_IF;
IF reset THEN
   // Reset device before the changes are used
   boardReset();
END_IF;

// Start network
netOpen(iface := 2);

BEGIN
END;
END_PROGRAM;
```

4.2.21.23 rchFallbackGet (Functionblock)

Architecture: NX32L
Device support: All
Firmware version: 1.54.00

---

rchFallbackGet will read out the fallback parameters for the RTCU Communication Hub, previously set from the RTCU IDE or by using the rchFallbackSet function.

Input:

None

Output:

enabled : BOOL
True if fallback connection is enabled, false if not.

address : STRING
The IP address or symbolic name of the server.

port : DINT
The IP port the device should use to connect to the server.

loginkey : STRING
The key (password) the device should use to connect to the server.

MaxConnectionAttempt : INT
Max number of connection attempts before the network media reconnects.

MaxSendReqAttempt : INT
Max number of send-request attempts before send fails.

ResponseTimeout : INT
Time waiting for response in seconds.

AliveFreq : DINT
Frequency for sending self-transactions in seconds.

CryptKey[n] : SINT
The key used to encrypt communication with the server.

Declaration:

```
FUNCTION_BLOCK rchFallbackGet;
VAR_OUTPUT
   enabled              : BOOL;
   address              : STRING;
   port                 : DINT;
   loginkey             : STRING;
   MaxConnectionAttempt : INT;
   MaxSendReqAttempt    : INT;
   ResponseTimeout      : INT;
   AliveFreq            : DINT;
   CryptKey             : ARRAY[1..16] OF SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   gprsParm  : sockGetTCPIPParm;
   rchParm   : rchFallbackGet;
   parmReset : BOOL := FALSE;
   CryptKey  : ARRAY[1..16] OF SINT;
END_VAR;

// Check mobile network and RTCU Communication Hub settings
gprsParm();
rchParm();
IF gprsParm.IP <> 0 OR gprsParm.SubnetMask <> 0 OR gprsParm.Gateway <> 0 OR
   gprsParm.DNS1 <> 0 OR gprsParm.DNS2 <> 0 OR gprsParm.Authenticate <> 0 OR
   strCompare(str1 := gprsParm.Username, str2 := "") <> 0 OR
   strCompare(str1 := gprsParm.Password, str2 := "") <> 0 OR
   strCompare(str1 := gprsParm.APN, str2 := "internet") <> 0 THEN
   // Set APN and keep the default values for the others
   sockSetTCPIPParm(APN := "internet");
   parmReset := TRUE;
END_IF;
IF NOT rchParm.GWEnabled OR rchParm.GWPort <> 5001 OR rchParm.CryptKey[1] <> 0
 OR
   rchParm.CryptKey[2] <> 0 OR rchParm.CryptKey[3] <> 0 OR rchParm.CryptKey[4]
 <> 0 OR
   rchParm.CryptKey[5] <> 0 OR rchParm.CryptKey[6] <> 0 OR rchParm.CryptKey[7]
 <> 0 OR
   rchParm.CryptKey[8] <> 0 OR rchParm.CryptKey[9] <> 0 OR
rchParm.CryptKey[10] <> 0 OR
   rchParm.CryptKey[11] <> 0 OR rchParm.CryptKey[12] <> 0 OR
rchParm.CryptKey[13] <> 0 OR
   rchParm.CryptKey[14] <> 0 OR rchParm.CryptKey[15] <> 0 OR
rchParm.CryptKey[16] <> 0 OR
   strCompare(str1 := rchParm.GWIP, str2 := "gw.rtcu.dk") <> 0 OR
   strCompare(str1 := rchParm.GWKey, str2 := "AABBCCDD") <> 0 THEN
   // Clear encryption key
   CryptKey[1]  := 0;
   CryptKey[2]  := 0;
   CryptKey[3]  := 0;
   CryptKey[4]  := 0;
   CryptKey[5]  := 0;
```

```
   CryptKey[6]  := 0;
   CryptKey[7]  := 0;
   CryptKey[8]  := 0;
   CryptKey[9]  := 0;
   CryptKey[10] := 0;
   CryptKey[11] := 0;
   CryptKey[12] := 0;
   CryptKey[13] := 0;
   CryptKey[14] := 0;
   CryptKey[15] := 0;
   CryptKey[16] := 0;
   // Set Gateway parameters
   rchFallbackSet(enabled := TRUE, address := "gw.rtcu.dk", port := 5001,
loginkey := "AABBCCDD", CryptKey := ADDR(CryptKey));
   parmReset := TRUE;
END_IF;
IF parmReset THEN
   // Reset device before the changes are used
   boardReset();
END_IF;

// Turn on power to the GSM module
gsmPower(power := TRUE);
gprsOpen();

BEGIN
   ...
END;
END_PROGRAM;
```

4.2.21.24 rchFallbackSet (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.54.00

Sets the parameters for making connections to the fallback RTCU Cummunication Hub. This has the same effect as setting the parameters via Device: Network: RTCU Communication Hub settings or by using the sockSetGWParm function.
The changes will take effect at the next call to gwOpen or at device reset. (e.g. by using boardReset).

Input:

enable : BOOL
True to enable fallback connection, false to disable.

address : STRING (*Max 41 characters*)
The IP address or symbolic name of the server.

port : DINT *(default 5001)*
The IP port the device should use to connect to the server.

loginkey : STRING (*Max 8 characters*)
The key (password) the device should use to connect to the server.

MaxConnectionAttempt : INT *(default 3)*
Max number of connection attempts before the network media reconnects.

MaxSendReqAttempt : INT ***(default 5)***
Max number of send-request attempts before send fails.

ResponseTimeout : INT ***(default 45)***
Time waiting for response in seconds.

AliveFreq : DINT ***(default 300)***
Frequency for sending self-transactions in seconds.
The purpose of the self-transaction is to ensure a healthy two-way communication channel over the server.
For applications that are only sending data from the device to the server, this frequency can safely be increased.
Setting the value to zero will disable the self-transactions completely.

CryptKey : PTR
The key used to encrypt communication with the server.
If this parameter is not set (CryptKey set to 0), then the encryption key is not changed.

Returns: INT

0 - Success.
1 - String too long.

Declaration:

```
FUNCTION rchFallbackSet;
VAR_INPUT
   enable               : BOOL;
   address              : STRING;
   port                 : DINT := 5001;
   loginkey             : STRING;
   MaxConnectionAttempt : INT := 3;
   MaxSendReqAttempt    : INT := 5;
   ResponseTimeout      : INT := 45;
   AliveFreq            : DINT := 300;
   CryptKey             : PTR;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   gprsParm  : sockGetTCPIPParm;
   rchParm   : rchFallbackGet;
   parmReset : BOOL := FALSE;
   CryptKey  : ARRAY[1..16] OF SINT;
END_VAR;

// Check mobile network and RTCU Communication Hub settings
gprsParm();
rchParm();
IF gprsParm.IP <> 0 OR gprsParm.SubnetMask <> 0 OR gprsParm.Gateway <> 0 OR
   gprsParm.DNS1 <> 0 OR gprsParm.DNS2 <> 0 OR gprsParm.Authenticate <> 0 OR
   strCompare(str1 := gprsParm.Username, str2 := "") <> 0 OR
   strCompare(str1 := gprsParm.Password, str2 := "") <> 0 OR
   strCompare(str1 := gprsParm.APN, str2 := "internet") <> 0 THEN
   // Set APN and keep the default values for the others
   sockSetTCPIPParm(APN := "internet");
```

```
   parmReset := TRUE;
END_IF;
IF NOT rchParm.GWEnabled OR rchParm.GWPort <> 5001 OR rchParm.CryptKey[1] <> 0
 OR
   rchParm.CryptKey[2] <> 0 OR rchParm.CryptKey[3] <> 0 OR rchParm.CryptKey[4]
 <> 0 OR
   rchParm.CryptKey[5] <> 0 OR rchParm.CryptKey[6] <> 0 OR rchParm.CryptKey[7]
 <> 0 OR
   rchParm.CryptKey[8] <> 0 OR rchParm.CryptKey[9] <> 0 OR
rchParm.CryptKey[10] <> 0 OR
   rchParm.CryptKey[11] <> 0 OR rchParm.CryptKey[12] <> 0 OR
rchParm.CryptKey[13] <> 0 OR
   rchParm.CryptKey[14] <> 0 OR rchParm.CryptKey[15] <> 0 OR
rchParm.CryptKey[16] <> 0 OR
   strCompare(str1 := rchParm.GWIP, str2 := "gw.rtcu.dk") <> 0 OR
   strCompare(str1 := rchParm.GWKey, str2 := "AABBCCDD") <> 0 THEN
   // Clear encryption key
   CryptKey[1]  := 0;
   CryptKey[2]  := 0;
   CryptKey[3]  := 0;
   CryptKey[4]  := 0;
   CryptKey[5]  := 0;
   CryptKey[6]  := 0;
   CryptKey[7]  := 0;
   CryptKey[8]  := 0;
   CryptKey[9]  := 0;
   CryptKey[10] := 0;
   CryptKey[11] := 0;
   CryptKey[12] := 0;
   CryptKey[13] := 0;
   CryptKey[14] := 0;
   CryptKey[15] := 0;
   CryptKey[16] := 0;
   // Set Gateway parameters
   rchFallbackSet(enable := TRUE, address := "gw.rtcu.dk", port := 5001,
loginkey := "AABBCCDD", CryptKey := ADDR(CryptKey));
   parmReset := TRUE;
END_IF;
IF parmReset THEN
   // Reset device before the changes are used
   boardReset();
END_IF;

// Turn on power to the GSM module
gsmPower(power := TRUE);
gprsOpen();

BEGIN
   ...
END;
END_PROGRAM;
```

4.2.21.25 rchHeartBeat (Function)

Architecture: NX32 / NX32L
Device support: All
Firmware version: 5.12 / 1.70.00

Calling this function will enable the heartbeat functionality supported by the RTCU Communication Hub.
UDP/IP heartbeat packages will be sent to the RTCU Communication Hub in accordance with the parameters specified when calling this function.

This functionality will notify an RTCU Communication Hub plug-in with corresponding "heartbeat offline" and "heartbeat online" events.
Please refer to the RTCU Communication Hub for more information.

Using this Heartbeat functionality allows very fast detection of a device that goes offline/online for example when the quality of the cellular connection is suboptimal.

The 'freq' is how often the Heartbeat should be sent. The 'timeout' is the maximum time without a heartbeat received by the RTCU Communication Hub until the device is considered offline.

Input:

timeout : UINT *(default 0)*
Timeout in seconds. Specifying a value of 0 will use a value that is two-times the self-transaction frequency.

freq : USINT *(default 0)*
The frequency of the heartbeat in seconds. Specifying 0 will disable the heartbeat.

Returns: INT

1 - Success.
0 - Not supported.
-1 - The timeout must be larger than the frequency.

Declaration:

```
FUNCTION rchHeartBeat;
VAR_INPUT
   timeout              : UINT := 0;
   freq                 : USINT := 0;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

rchHeartBeat(timeout:=30,freq:=10);
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.21.26 rchInterfaceSet (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.54.00

This function temporarily changes the network interface used to connect to the RTCU Communication Hub.
This function works similarly to gwSetMedia.
The default network interface is cellular.

If a non-fallback server configuration is enabled, this function must be called after gwOpen to work.

Input:

iface : SINT (*Default 1*)
The network interface to use to connect to the server. (see Network for available interfaces)

Returns: INT

0 - Success.
1 - Iface not found.

Declaration:

```
FUNCTION rchInterfaceSet : INT;
VAR_INPUT
   iface : SINT := 1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   eth : BOOL;
END_VAR;

VAR_OUTPUT
   LED_GPRS : BOOL;
   LED_RCH  : BOOL;
END_VAR;

PROGRAM example;
VAR
   rch_iface : SINT;
END_VAR;

   // Init cellular
   gsmPower(power := ON);
   netOpen(iface := 1);

   // Init Ethernet
   netOpen(iface := 2);

   DebugMsg(message:="Program running");

BEGIN
   LED_GPRS := netConnected(iface := 1);
   LED_RCH  := gwConnected();

   // Change RCH interface?
   rchInterfaceGet(iface := rch_iface);
   IF rch_iface <> 2 AND eth THEN
      // Select interface
      rchInterfaceSet(iface := 2);  // Use Ethernet
   ELSIF rch_iface <> 1 AND NOT eth THEN
      // Select interface
      rchInterfaceSet(iface := 1);  // Use Cellular
  END_IF;

END;
END_PROGRAM
```

4.2.21.27 rchInterfaceGet (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.62.00

This function reads the network interface used to connect to the RTCU Communication Hub.
This function works similarly to gwGetMedia.

Output:

iface : SINT
The network interface used to connect to the server. (see Network for available interfaces)

Returns: INT

1 - Success
0 - Not supported
-1 - Error reading media

Declaration:

```
FUNCTION rchInterfaceGet : INT;
VAR_INPUT
   iface : ACCESS SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   eth : BOOL;
END_VAR;

VAR_OUTPUT
   LED_GPRS : BOOL;
   LED_RCH  : BOOL;
END_VAR;

PROGRAM example;
VAR
   rch_iface : SINT;
END_VAR;

   // Init cellular
   gsmPower(power := ON);
   netOpen(iface := 1);

   // Init Ethernet
   netOpen(iface := 2);

   DebugMsg(message:="Program running");

BEGIN
   LED_GPRS := netConnected(iface := 1);
   LED_RCH  := gwConnected();

   // Change RCH interface?
   rchInterfaceGet(iface := rch_iface);
```

```
   IF rch_iface <> 2 AND eth THEN
      // Select interface
      rchInterfaceSet(iface := 2);  // Use Ethernet
   ELSIF rch_iface <> 1 AND NOT eth THEN
      // Select interface
      rchInterfaceSet(iface := 1);  // Use Cellular
  END_IF;

END;
END_PROGRAM
```

4.2.22 hsio: High-speed IO

4.2.22.1 HSIO: High-Speed IO

The High-Speed IO functions use interrupts to monitor the on-board input signals and reads the state significantly faster than it is possible to do directly from VPL.
This can be used e.g. to read incremental encoders.
See also the PCT function.

Note: Only one type of high-speed IO can be used on each signal at a time, including PCT.
Note: Generating too fast input on too many input signals for too long a time may affect the performance. Make sure to check the performance with a realistic workload.

Incremental encoders

Incremental encoders are linear or rotary encoders that are used to determine the movement of an object. The encoder will typically have two or three outputs, A, B and Z. A and B are used to report the position while Z is used for indexing the encoder, and is typically only active once for every full rotation.

- hsioEncoderEnable — Configures and enables an encoder.
- hsioEncoderDisable — Disables an encoder.
- hsioEncoderRead — Reads the current position of an encoder.
- hsioEncoderReset — Resets the current position of an encoder.

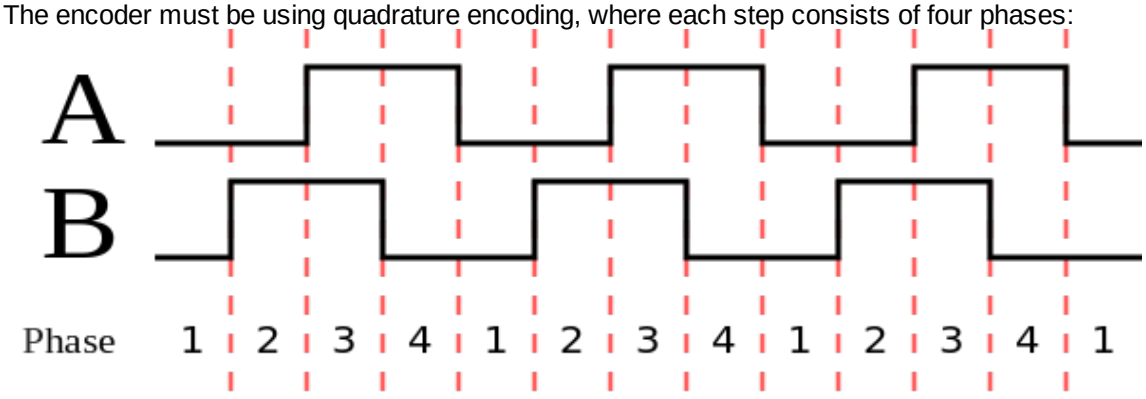
4.2.22.2 hsioEncoderEnable (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.52.00

---

hsioEncoderEnable configures and enables monitoring of an incremental encoder.
If the Z input is provided, it can be used to improve the positioning.

The encoder must be using quadrature encoding, where each step consists of four phases:



By examining the signals, it is possible to determine which direction the encoder is being moved and for how many steps, making it possible to determine the position.

When the Z-input is asserted while the encoder uses a mode that uses the Z-input, it will be registered and the position will be updated when the next step starts.

Note: Digital input 6-8 on the RTCU LX4 can not be used for encoders.

Input:

a : PTR
Address of the digital input signal to use as A (please see the example below).

b : PTR
Address of the digital input signal to use as B (please see the example below).

z : PTR
Address of the optional digital input signal to use as Z (please see the example below).

id : INT (1..2) *default 1*
Identifier of the encoder to configure.

mode : INT (0..3) *default 0*
The mode to use for handling the Z signal.
0 Z is not handled.
1 Reset on every Z.
Every time Z is observed, the position is reset to 0. This is useful if Z is only active once during the entire movement, e.g. a rotary encoder that rotates 360 degrees or less.
2 Reset on first Z.
The position will be reset to 0 the next time z is observed. This can e.g. be used in combination with an end-stop to zero the position.
3 Calibrate the position on every Z.
This will set the position to the nearest multiple of the number of steps provided, every time Z is asserted.
This will make sure that a few lost steps does not accumulate across multiple rotations of a rotary encoder.

steps : INT *default 0*
The number of steps on a full rotation. Only needed in mode 3.

Returns: INT

1	- Success
0	- Function is not supported.
-1	- Unspecified error.
-2	- Invalid signal provided. This can only be used on on-board high speed input signals.
-3	- Invalid mode.
-4	- The mode requires that steps has a valid value.
-5	- Invalid ID.
-6	- The encoder is already in use. Disable it first with hsioEncoderDisable.

Declaration:

```
FUNCTION hsioEncoderEnable : INT;
VAR_INPUT
   a     : PTR;
   b     : PTR;
   z     : PTR;
   id    : INT := 1;
   mode  : INT := 0;
   steps : INT := 0;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
VAR_INPUT
  a : BOOL;
  b : BOOL;
  z : BOOL;
```

```
END_VAR;

PROGRAM test;
BEGIN
   ...
   // Enable encoder
   hsioEncoderEnable(id := 1, a := ADDR(a), b := ADDR(b), z := ADDR(z), mode
:= 3, steps := 200);
   ...
END;

END_PROGRAM;
```

4.2.22.3 hsioEncoderDisable (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.52.00

---

hsioEncoderDisable disables monitoring of an encoder.

Input:

id : INT (1..2) ***default 1***
Identifier of the encoder to disable.

Returns: **INT**

1	- Success
0	- Function is not supported.
-1	- Unspecified error.
-5	- Invalid ID.

Declaration:

```
FUNCTION hsioEncoderDisable : INT;
VAR_INPUT
   id    : INT := 1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
BEGIN
   ...
   // Disable encoder
   hsioEncoderDisable(id := 1);
   ...
END;

END_PROGRAM;
```

4.2.22.4 hsioEncoderRead (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.52.00

hsioEncoderRead reads the current position of the encoder.

Input:

id : INT (1..2) ***default 1***
Identifier of the encoder to read the position of.

Returns: DINT

The current position of the encoder.
Returns 0 on error.

Declaration:

```
FUNCTION hsioEncoderRead : DINT;
VAR_INPUT
   id    : INT := 1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
BEGIN
   ...
   // Get position
   DebugFmt(message:="Position: \4", v4:= hsioEncoderRead(id := 1));
   ...
END;

END_PROGRAM;
```

4.2.22.5 hsioEncoderReset (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.52.00

hsioEncoderReset resets the position of an encoder.
By default it will reset it immediately. If wait is set to true, it will reset the position the next time Z is active.

Input:

id : INT (1..2) ***default 1***
Identifier of the encoder to reset.

wait : BOOL ***default FALSE***
Wait for the Z-input to be active before resetting. Can not be combined with mode 3.

Returns: INT

1	- Success
0	- Function is not supported.
-1	- Unspecified error.
-5	- Invalid ID.
-6	- The Z-input is used for calibration. It can not also be used for reset at the same time.

-7 - The encoder is not enabled or does not have Z.

Declaration:

```
FUNCTION hsioEncoderReset : INT;
VAR_INPUT
   id    : INT  := 1;
   wait  : BOOL := FALSE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
BEGIN
   ...
   // Reset encoder
   hsioEncoderReset(id := 1, wait := TRUE);
   ...
END;

END_PROGRAM;
```

4.2.23 json: JSON functions

4.2.23.1 json: JSON Functions

The JSON functions are used to create and manipulate JSON structures.
The JSON functions are only available in NX32L compilation mode.

The strings used with JSON must use UTF-8 encoding, so non-ASCII characters must be entered as special characters, see below.

Examples of how the JSON functions can be used can be found in the REST Example and the REST Basic Authentication Example.

The following functions are available:

- jsonCreateArray — Creates a new,empty JSON array.
- jsonCreateObject — Creates a new, empty JSON object.
- jsonFree — Releases a JSON object or array
- jsonGetType — Returns the type of a JSON value.
- jsonGetKey — Iterate through a JSON object.
- jsonArraySize — Get the size of a JSON array.
- jsonAddValue — Adds a JSON structure to the end of a JSON array.
- jsonAddValueString — Adds a string to the end of a JSON array.
- jsonAddValueInt — Adds an integer to the end of a JSON array.
- jsonAddValueFloat — Adds a floating point value to the end of a JSON array.
- jsonAddValueBool — Adds a boolean to the end of a JSON array.
- jsonAddValueNull — Adds a null-value to the end of a JSON array.
- jsonSetValue — Stores a JSON structure at a specific index in a JSON array or at a specific key in a JSON object.
- jsonSetValueString — Stores a string at a specific index in a JSON array or at a specific key in a JSON object.
- jsonSetValueInt — Stores an integer at a specific index in a JSON array or at a specific key in a JSON object.
- jsonSetValueFloat — Stores a floating point value at a specific index in a JSON array or at a specific key in a JSON object.
- jsonSetValueBool — Stores a boolean at a specific index in a JSON array or at a specific key in a JSON object.
- jsonSetValueNull — Stores a null-value at a specific index in a JSON array or at a specific key in a JSON object.
- jsonGetValue — Reads a JSON structure from the provided JSON structure
- jsonGetString — Reads a string from the provided JSON structure
- jsonGetInt — Reads an integer from the provided JSON structure
- jsonGetFloat — Reads a floating point value from the provided JSON structure
- jsonGetBool — Reads a boolean from the provided JSON structure
- jsonDeleteValue — Removes a value from a JSON structure.
- jsonToString — Creates a string representation of the JSON structure.
- jsonFromString — Creates a JSON structure from a string.
- jsonHandleStats — Retrieves information about JSON handles, to help with detecting handle leaks.

Using Unicode strings

The JSON API uses Unicode strings so that any text can be in the local language rather than in English.
While it is not possible to enter Unicode text directly into a string variable, it can be done as a string of special characters (see STRING).
This approach requires 3 steps for each character:
1. Find the character at the Unicode homepage (www.unicode.org).
2. Encode the character into UTF-8 (Wikipedia gives a good description of this here: en.wikipedia.org/wiki/Utf-8).
3. Type in the special characters from step 2 (e.g. "CF86" for the Greek character "phi").

4.2.23.2 jsonCreateArray (Function)

Architecture: NX32L
Device support: All
Firmware version: 2.10.00

---

jsonCreateArray creates a new, empty JSON array.

When the JSON array is no longer needed, it must be released using jsonFree.

Input:
None

Output:
o : SYSHANDLE
A handle to the JSON array.

Returns: INT
1 - Success
0 - Function is not supported.
-6 - Could not create JSON array, there may be too many in use.

Declaration:
```
FUNCTION jsonCreateArray : INT;
VAR_INPUT
   o      : ACCESS SYSHANDLE;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   arr : SYSHANDLE;
   obj : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Create array
```

```
   rc := jsonCreateArray(o := arr);
   // Add integer
   rc := jsonAddValueInt(o := arr, value := 123);
   // Create object
   rc := jsonCreateObject(o := obj);
   // Add integer to object
   rc := jsonSetValueInt(o := obj, key := "Number", value := 42);
   // Replace integer in array with object
   rc := jsonSetValue(o := arr, idx := 0, value := obj);
   // Release object handle
   rc := jsonFree(o := obj);
   ...
END;

END_PROGRAM;
```

4.2.23.3 jsonCreateObject (Function)

Architecture: NX32L
Device support: All
Firmware version: 2.10.00

---

jsonCreateObject creates a new, empty JSON object.

When the JSON object is no longer needed, it must be released using jsonFree.

Input:

None

Output:

o : SYSHANDLE
A handle to the JSON object.

Returns: INT

1 - Success
0 - Function is not supported.
-6 - Could not create JSON object, there may be too many in use.

Declaration:

```
FUNCTION jsonCreateObject : INT;
VAR_INPUT
   o     : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   arr : SYSHANDLE;
   obj : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Create array
```

```
   rc := jsonCreateArray(o := arr);
   // Add integer
   rc := jsonAddValueInt(o := arr, value := 123);
   // Create object
   rc := jsonCreateObject(o := obj);
   // Add integer to object
   rc := jsonSetValueInt(o := obj, key := "Number", value := 42);
   // Replace integer in array with object
   rc := jsonSetValue(o := arr, idx := 0, value := obj);
   // Release object handle
   rc := jsonFree(o := obj);
   ...
END;

END_PROGRAM;
```

4.2.23.4 jsonFree (Function)

Architecture: NX32L
Device support: All
Firmware version: 2.10.00

---

jsonFree releases the resources used by a JSON object or array.

Input:

o : SYSHANDLE
A handle to the JSON object or array to release. Will be made invalid once it has been released.

Returns: INT

1 - Success
0 - Function is not supported.
-3 - JSON structure was not found.

Declaration:

```
FUNCTION jsonFree : INT;
VAR_INPUT
   o     : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   arr : SYSHANDLE;
   obj : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Create array
   rc := jsonCreateArray(o := arr);
   // Add integer
   rc := jsonAddValueInt(o := arr, value := 123);
```

```
    // Create object
    rc := jsonCreateObject(o := obj);
    // Add integer to object
    rc := jsonSetValueInt(o := obj, key := "Number", value := 42);
    // Replace integer in array with object
    rc := jsonSetValue(o := arr, idx := 0, value := obj);
    // Release object handle
    rc := jsonFree(o := obj);
    ...
END;

END_PROGRAM;
```

4.2.23.5 jsonGetType (Function)

Architecture: NX32L
Device support: All
Firmware version: 2.10.00

---

jsonGetType returns the type of the provided JSON value.
This can be used to determine which function to call when handling unknown values.
jsonGetType can be used to get the type of the elements in a JSON array by changing the idx value.

Input:

o : SYSHANDLE
A handle to a JSON structure.

idx : INT *(Default -1)*
If idx is -1, the type of o will be returned.
If idx is not -1 and o is a JSON array, the type of o[idx] will be returned.

Returns: INT

7	- Null.
6	- Boolean. Can be read using jsonSetValueBool.
5	- Floating point number.
4	- Integer.
3	- String.
2	- JSON Array.
1	- JSON Object.
0	- Function is not supported.
-2	- JSON value not found.
-3	- Array element is not found, idx is outside the valid range
-4	- idx is not -1 and o is not an array

Declaration:

```
FUNCTION jsonGetType : INT;
VAR_INPUT
    o      : SYSHANDLE;
    idx    : INT := -1;
END_VAR;
```

Example:

```
// Dumps JSON structure to device output
FUNCTION INTERFACE dumpJSON;
VAR_INPUT
    json    : SYSHANDLE;
```

```
   prefix : STRING;
END_VAR;
END_FUNCTION;

// Dumps array to device output
FUNCTION dumpArray;
VAR_INPUT
   json   : SYSHANDLE;
   prefix : STRING;
END_VAR;
VAR
   type, i, rc : INT;

   tmp    : SYSHANDLE;
   key    : STRING;
   txt    : STRING;
   str    : STRING;
   d      : DINT;
   b      : BOOL;
   f      : DOUBLE;
END_VAR;

   FOR i := 0 TO jsonArraySize(o:=json) - 1 DO
      txt := strFormat(format:=prefix + "[\1]:", v1:=i);

      type := jsonGetType(o:=json, idx:= i);
      CASE type OF
         1,2:// Object or array
            DebugFmt(message:=txt);
            rc := jsonGetValue(o:=json, idx:=i, value := tmp);
            dumpJSON(json:=tmp, prefix := prefix);
            rc := jsonFree(o:=tmp);
         3: // string
            jsonGetString(o:=json, idx := i, value :=str);
            DebugFmt(message:=txt+":$""+str+"$"");
         4: // int
            jsonGetInt(o:=json, idx := i, value :=d);
            DebugFmt(message:=txt+":\4", v4:=d);
         5: // float
            jsonGetFloat(o:=json, idx := i, value :=f);
            DebugFmt(message:=txt+": "+doubleToStr(v:=f));
         6: // Bool
            jsonGetBool(o:=json, idx := i, value :=b);
            IF b THEN
               DebugFmt(message:=txt+": TRUE");
            ELSE
               DebugFmt(message:=txt+": FALSE");
            END_IF;
         7: //NULL
            DebugFmt(message:=txt+": NULL");
      END_CASE;
   END_FOR;
END_FUNCTION;

// Dumps object to device output
FUNCTION dumpObject;
VAR_INPUT
   json   : SYSHANDLE;
   prefix : STRING;
END_VAR;
VAR
   type, i, rc : INT;

   tmp    : SYSHANDLE;
```

```
   key     : STRING;
   txt     : STRING;
   str     : STRING;
   d       : DINT;
   b       : BOOL;
   f       : DOUBLE;
END_VAR;

   i := 0;
   REPEAT
      rc := jsonGetKey(o:=json, idx:= i, key:=key, type:=type);
      txt := prefix + "$""+key+"$"";
      IF rc = 1 THEN
         CASE type OF
         1,2:// Object or array
            DebugFmt(message:=txt);
            rc := jsonGetValue(o:=json, key:=key, value := tmp);
            dumpJSON(json:=tmp, prefix := prefix);
            rc := jsonFree(o:=tmp);

         3: // string
            jsonGetString(o:=json, key:=key, value :=str);
            DebugFmt(message:=txt+":$""+str+"$"");
         4: // int
            jsonGetInt(o:=json, key:=key, value :=d);
            DebugFmt(message:=txt+":\4", v4:=d);
         5: //Real
            jsonGetFloat(o:=json, key:=key, value :=f);
            DebugFmt(message:=txt+": "+doubleToStr(v:=f));
         6: // Bool
            jsonGetBool(o:=json, key:=key, value :=b);
            IF b THEN
               DebugFmt(message:=txt+": TRUE");
            ELSE
               DebugFmt(message:=txt+": FALSE");
            END_IF;
         7: // NULL
            DebugFmt(message:=txt+": NULL");
         END_CASE;
      END_IF;
      i := i+1;
   UNTIL  rc < 0
   END_REPEAT;
END_FUNCTION;

// Implementation of dumpJSON
FUNCTION IMPLEMENTATION dumpJSON;
VAR
   type: INT;
END_VAR;

   type := jsonGetType(o:=json, idx:=-1);
   IF type = 1 THEN // Object
      DebugFmt(message:=prefix+"object{");
      dumpObject(json:=json, prefix:=prefix+" ");
      DebugFmt(message:=prefix+"}");
   END_IF;

   IF type = 2 THEN // Array
      DebugFmt(message:=prefix+"array[");
      dumpArray(json:=json, prefix:=prefix+" ");
      DebugFmt(message:=prefix+"]");
   END_IF;
```

```
END_FUNCTION;
```

4.2.23.6 jsonGetKey (Function)

Architecture: NX32L
Device support: All
Firmware version: 2.10.00

---

jsonGetKey returns the name and type of a value in a JSON object.
This can be used to determine which function to call when handling unknown values.

Input:

o : SYSHANDLE
A handle to a JSON structure.

idx : INT *(Default 0)*
The index of the value to examine, starting at 0.

Output:

key : STRING
The name of the value.

type : INT
The type of the value:

- 7 - Null.
- 6 - Boolean. Can be read using jsonSetValueBool.
- 5 - Floating point number.
- 4 - Integer.
- 3 - String.
- 2 - JSON Array.
- 1 - JSON Object.

Returns: INT

- 1 - Success
- 0 - Function is not supported.
- -2 - JSON value not found.
- -3 - Array element is not found, idx is outside the valid range
- -4 - idx is not -1 and o is not an array

Declaration:

```
FUNCTION jsonGetKey : INT;
VAR_INPUT
   o     : SYSHANDLE;
   idx   : INT;
   key   : ACCESS STRING;
   type  : ACCESS INT;
END_VAR;
```

Example:

```
// Dumps JSON structure to device output
FUNCTION INTERFACE dumpJSON;
VAR_INPUT
   json   : SYSHANDLE;
```

```
   prefix : STRING;
END_VAR;
END_FUNCTION;

// Dumps array to device output
FUNCTION dumpArray;
VAR_INPUT
   json   : SYSHANDLE;
   prefix : STRING;
END_VAR;
VAR
   type, i, rc : INT;

   tmp    : SYSHANDLE;
   key    : STRING;
   txt    : STRING;
   str    : STRING;
   d      : DINT;
   b      : BOOL;
   f      : DOUBLE;
END_VAR;

   FOR i := 0 TO jsonArraySize(o:=json) - 1 DO
      txt := strFormat(format:=prefix + "[\1]:", v1:=i);

      type := jsonGetType(o:=json, idx:= i);
      CASE type OF
         1,2:// Object or array
            DebugFmt(message:=txt);
            rc := jsonGetValue(o:=json, idx:=i, value := tmp);
            dumpJSON(json:=tmp, prefix := prefix);
            rc := jsonFree(o:=tmp);
         3: // string
            jsonGetString(o:=json, idx := i, value :=str);
            DebugFmt(message:=txt+":$""+str+"$"");
         4: // int
            jsonGetInt(o:=json, idx := i, value :=d);
            DebugFmt(message:=txt+":\4", v4:=d);
         5: // float
            jsonGetFloat(o:=json, idx := i, value :=f);
            DebugFmt(message:=txt+": "+doubleToStr(v:=f));
         6: // Bool
            jsonGetBool(o:=json, idx := i, value :=b);
            IF b THEN
               DebugFmt(message:=txt+": TRUE");
            ELSE
               DebugFmt(message:=txt+": FALSE");
            END_IF;
         7: //NULL
            DebugFmt(message:=txt+": NULL");
      END_CASE;
   END_FOR;
END_FUNCTION;

// Dumps object to device output
FUNCTION dumpObject;
VAR_INPUT
   json   : SYSHANDLE;
   prefix : STRING;
END_VAR;
VAR
   type, i, rc : INT;

   tmp    : SYSHANDLE;
```

```
   key     : STRING;
   txt     : STRING;
   str     : STRING;
   d       : DINT;
   b       : BOOL;
   f       : DOUBLE;
END_VAR;

   i := 0;
   REPEAT
      rc := jsonGetKey(o:=json, idx:= i, key:=key, type:=type);
      txt := prefix + "$""+key+"$"";
      IF rc = 1 THEN
         CASE type OF
         1,2:// Object or array
            DebugFmt(message:=txt);
            rc := jsonGetValue(o:=json, key:=key, value := tmp);
            dumpJSON(json:=tmp, prefix := prefix);
            rc := jsonFree(o:=tmp);

         3: // string
            jsonGetString(o:=json, key:=key, value :=str);
            DebugFmt(message:=txt+":$""+str+"$"");
         4: // int
            jsonGetInt(o:=json, key:=key, value :=d);
            DebugFmt(message:=txt+":\4", v4:=d);
         5: //Real
            jsonGetFloat(o:=json, key:=key, value :=f);
            DebugFmt(message:=txt+": "+doubleToStr(v:=f));
         6: // Bool
            jsonGetBool(o:=json, key:=key, value :=b);
            IF b THEN
               DebugFmt(message:=txt+": TRUE");
            ELSE
               DebugFmt(message:=txt+": FALSE");
            END_IF;
         7: // NULL
            DebugFmt(message:=txt+": NULL");
         END_CASE;
      END_IF;
      i := i+1;
   UNTIL  rc < 0
   END_REPEAT;
END_FUNCTION;

// Implementation of dumpJSON
FUNCTION IMPLEMENTATION dumpJSON;
VAR
   type: INT;
END_VAR;

   type := jsonGetType(o:=json, idx:=-1);
   IF type = 1 THEN // Object
      DebugFmt(message:=prefix+"object{");
      dumpObject(json:=json, prefix:=prefix+" ");
      DebugFmt(message:=prefix+"}");
   END_IF;

   IF type = 2 THEN // Array
      DebugFmt(message:=prefix+"array[");
      dumpArray(json:=json, prefix:=prefix+" ");
      DebugFmt(message:=prefix+"]");
   END_IF;
```

```
END_FUNCTION;
```

4.2.23.7 jsonArraySize (Function)

Architecture: NX32L
Device support: All
Firmware version: 2.10.00

---

jsonArraySize returns the number of elements in the provided JSON array.

Input:

o : SYSHANDLE
A handle to a JSON array.

Returns: INT

>=0	- The number of elements in the array.
0	- Function is not supported.
-3	- Not a valid handle.
-4	- Not a valid JSON array.

Declaration:

```
FUNCTION jsonArraySize : INT;
VAR_INPUT
   o      : SYSHANDLE;
END_VAR;
```

Example:

```
// Dumps JSON structure to device output
FUNCTION INTERFACE dumpJSON;
VAR_INPUT
   json   : SYSHANDLE;
   prefix : STRING;
END_VAR;
END_FUNCTION;

// Dumps array to device output
FUNCTION dumpArray;
VAR_INPUT
   json   : SYSHANDLE;
   prefix : STRING;
END_VAR;
VAR
   type, i, rc : INT;

   tmp    : SYSHANDLE;
   key    : STRING;
   txt    : STRING;
   str    : STRING;
   d      : DINT;
   b      : BOOL;
   f      : DOUBLE;
END_VAR;

   FOR i := 0 TO jsonArraySize(o:=json) - 1 DO
      txt := strFormat(format:=prefix + "[\1]:", v1:=i);
```

```
      type := jsonGetType(o:=json, idx:= i);
      CASE type OF
         1,2:// Object or array
            DebugFmt(message:=txt);
            rc := jsonGetValue(o:=json, idx:=i, value := tmp);
            dumpJSON(json:=tmp, prefix := prefix);
            rc := jsonFree(o:=tmp);
         3: // string
            jsonGetString(o:=json, idx := i, value :=str);
            DebugFmt(message:=txt+":$""+str+"$"");
         4: // int
            jsonGetInt(o:=json, idx := i, value :=d);
            DebugFmt(message:=txt+":\4", v4:=d);
         5: // float
            jsonGetFloat(o:=json, idx := i, value :=f);
            DebugFmt(message:=txt+": "+doubleToStr(v:=f));
         6: // Bool
            jsonGetBool(o:=json, idx := i, value :=b);
            IF b THEN
               DebugFmt(message:=txt+": TRUE");
            ELSE
               DebugFmt(message:=txt+": FALSE");
            END_IF;
         7: //NULL
            DebugFmt(message:=txt+": NULL");
      END_CASE;
   END_FOR;
END_FUNCTION;

// Dumps object to device output
FUNCTION dumpObject;
VAR_INPUT
   json   : SYSHANDLE;
   prefix : STRING;
END_VAR;
VAR
   type, i, rc : INT;

   tmp    : SYSHANDLE;
   key    : STRING;
   txt    : STRING;
   str    : STRING;
   d      : DINT;
   b      : BOOL;
   f      : DOUBLE;
END_VAR;

   i := 0;
   REPEAT
      rc := jsonGetKey(o:=json, idx:= i, key:=key, type:=type);
      txt := prefix + "$""+key+"$"";
      IF rc = 1 THEN
         CASE type OF
         1,2:// Object or array
            DebugFmt(message:=txt);
            rc := jsonGetValue(o:=json, key:=key, value := tmp);
            dumpJSON(json:=tmp, prefix := prefix);
            rc := jsonFree(o:=tmp);

         3: // string
            jsonGetString(o:=json, key:=key, value :=str);
            DebugFmt(message:=txt+":$""+str+"$"");
         4: // int
            jsonGetInt(o:=json, key:=key, value :=d);
```

```
            DebugFmt(message:=txt+":\4", v4:=d);
         5: //Real
            jsonGetFloat(o:=json, key:=key, value :=f);
            DebugFmt(message:=txt+": "+doubleToStr(v:=f));
         6: // Bool
            jsonGetBool(o:=json, key:=key, value :=b);
            IF b THEN
               DebugFmt(message:=txt+": TRUE");
            ELSE
               DebugFmt(message:=txt+": FALSE");
            END_IF;
         7: // NULL
            DebugFmt(message:=txt+": NULL");
         END_CASE;
      END_IF;
      i := i+1;
   UNTIL  rc < 0
   END_REPEAT;
END_FUNCTION;

// Implementation of dumpJSON
FUNCTION IMPLEMENTATION dumpJSON;
VAR
   type: INT;
END_VAR;

   type := jsonGetType(o:=json, idx:=-1);
   IF type = 1 THEN // Object
      DebugFmt(message:=prefix+"object{");
      dumpObject(json:=json, prefix:=prefix+" ");
      DebugFmt(message:=prefix+"}");
   END_IF;

   IF type = 2 THEN // Array
      DebugFmt(message:=prefix+"array[");
      dumpArray(json:=json, prefix:=prefix+" ");
      DebugFmt(message:=prefix+"]");
   END_IF;

END_FUNCTION;
```

4.2.23.8 jsonAddValue (Function)

Architecture: NX32L
Device support: All
Firmware version: 2.10.00

---

jsonAddValue adds the provided JSON value to the end of the given JSON array, growing the size of the array by 1.

Input:

o : SYSHANDLE
A handle to the JSON array to add the value to.

value : SYSHANDLE
The value to add.

Returns: **INT**

1 - Success.
0 - Function is not supported.

-3 - Invalid handle.
-4 - Not an array
-7 - Could not find value
-99 - Failed to add value

Declaration:

```
FUNCTION jsonAddValue : INT;
VAR_INPUT
   o     : SYSHANDLE;
   value : SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   arr : SYSHANDLE;
   obj : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Create array
   rc := jsonCreateArray(o := arr);
   // Create object
   rc := jsonCreateObject(o := obj);
   // Add integer to object
   rc := jsonSetValueInt(o := obj, key := "Number", value := 42);
   // Add object to array
   rc := jsonAddValue(o := arr, value := obj);
   // Release object handle
   rc := jsonFree(o := obj);
   ...
END;

END_PROGRAM;
```

4.2.23.9 jsonAddValueString (Function)

Architecture: NX32L
Device support: All
Firmware version: 2.10.00

---

jsonAddValueString adds the provided string to the end of the given JSON array, growing the size of the array by 1.

Input:

o : SYSHANDLE
A handle to the JSON array to add the value to.

value : STRING
The UTF-8 string to add.

Returns: INT

1	- Success.
0	- Function is not supported.
-3	- Invalid handle.
-4	- Not an array
-6	- The string is not a valid UTF-8 string or there are not enough resources available to add it.
-99	- Failed to add value

Declaration:

```
FUNCTION jsonAddValueString : INT;
VAR_INPUT
   o     : SYSHANDLE;
   value : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   arr : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Create array
   rc := jsonCreateArray(o := arr);
   // Add string
   rc := jsonAddValueString(o := arr, value := "Test");
   ...
END;

END_PROGRAM;
```

4.2.23.10 jsonAddValueInt (Function)

```
Architecture:       NX32L
Device support:     All
Firmware version:   2.10.00
```

---

jsonAddValueInt adds the provided integer to the end of the given JSON array, growing the size of the array by 1.

Input:

o : SYSHANDLE
A handle to the JSON array to add the value to.

value : DINT
The value to add.

Returns: INT

1	- Success.
0	- Function is not supported.

-3 - Invalid handle.
-4 - Not an array
-6 - Not enough resources available to add value.
-99 - Failed to add value

Declaration:

```
FUNCTION jsonAddValueInt : INT;
VAR_INPUT
   o     : SYSHANDLE;
   value : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   arr : SYSHANDLE;
   obj : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Create array
   rc := jsonCreateArray(o := arr);
   // Add integer
   rc := jsonAddValueInt(o := arr, value := 123);
   // Create object
   rc := jsonCreateObject(o := obj);
   // Add integer to object
   rc := jsonSetValueInt(o := obj, key := "Number", value := 42);
   // Replace integer in array with object
   rc := jsonSetValue(o := arr, idx := 0, value := obj);
   // Release object handle
   rc := jsonFree(o := obj);
   ...
END;

END_PROGRAM;
```

4.2.23.11 jsonAddValueFloat (Function)

Architecture: NX32L
Device support: All
Firmware version: 2.10.00

jsonAddValueFloat adds the provided floating point value to the end of the given JSON array, growing the size of the array by 1.

Input:

o : SYSHANDLE
A handle to the JSON array to add the value to.

value : DOUBLE
The value to add.

Returns: INT

1	- Success.
0	- Function is not supported.
-3	- Invalid handle.
-4	- Not an array
-6	- Not enough resources available to add value.
-99	- Failed to add value

Declaration:

```
FUNCTION jsonAddValueFloat : INT;
VAR_INPUT
   o     : SYSHANDLE;
   value : DOUBLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   arr : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Create array
   rc := jsonCreateArray(o := arr);
   // Add floating point value
   rc := jsonAddValueFloat(o := arr, value := 12.3);
   ...
END;

END_PROGRAM;
```

4.2.23.12 jsonAddValueBool (Function)

Architecture: NX32L
Device support: All
Firmware version: 2.10.00

jsonAddValueBool adds the provided boolean value to the end of the given JSON array, growing the size of the array by 1.

Input:

o : SYSHANDLE
A handle to the JSON array to add the value to.

value : BOOL
The value to add.

Returns: INT

1	- Success.
0	- Function is not supported.
-3	- Invalid handle.
-4	- Not an array
-99	- Failed to add value

Declaration:

```
FUNCTION jsonAddValueBool : INT;
VAR_INPUT
   o     : SYSHANDLE;
   value : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   arr : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Create array
   rc := jsonCreateArray(o := arr);
   // Add boolean value
   rc := jsonAddValueBool(o := arr, value := TRUE);
   ...
END;

END_PROGRAM;
```

4.2.23.13 jsonAddValueNull (Function)

Architecture: NX32L
Device support: All
Firmware version: 2.10.00

---

jsonAddValueNull adds a null-value to the end of the given JSON array, growing the size of the array by 1.

Input:

o : SYSHANDLE
A handle to the JSON array to add the null-value to.

Returns: INT

1	- Success.
0	- Function is not supported.
-3	- Invalid handle.
-4	- Not an array
-99	- Failed to add value

Declaration:

```
FUNCTION jsonAddValueNull : INT;
VAR_INPUT
   o     : SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   arr : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Create array
   rc := jsonCreateArray(o := arr);
   // Add null-value
   rc := jsonAddValueNull(o := arr);
   ...
END;

END_PROGRAM;
```

4.2.23.114 jsonSetValue (Function)

Architecture: NX32L
Device support: All
Firmware version: 2.10.00

---

jsonSetValue stores the provided JSON value in a JSON object or array.

When working on a JSON object, set the key parameter to the name of the value to set.
When working on a JSON array, idx must be set to the index of the element to set, and the insert parameter can be used to control if it should replace the existing element.

Input:

o : SYSHANDLE
A handle to the JSON array or object to set the value on.

idx : INT *(Default -1)*
Index to use when working on a JSON array.
Leave at -1 when working on a JSON object.

key : STRING
The name of the value when working on a JSON object.

insert : BOOL *(Default FALSE)*
When FALSE, the existing value at the given position in the array will be replaced.
Set to TRUE to insert the value in the array at the given position and shift the elements after it one position towards the end.

value : SYSHANDLE
The value to add.

Returns: **INT**

1	- Success.
0	- Function is not supported.
-1	- Invalid key
-3	- Invalid handle.
-4	- The type of o does not match the expected type.
-5	- Array index is outside array.
-7	- Could not find value
-99	- Failed to set value

Declaration:

```
FUNCTION jsonSetValue : INT;
VAR_INPUT
   o     : SYSHANDLE;
   idx   : INT := -1;
   key   : STRING;
   insert: BOOL := FALSE;
   value : SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   arr : SYSHANDLE;
   obj : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Create array
   rc := jsonCreateArray(o := arr);
   // Add integer
   rc := jsonAddValueInt(o := arr, value := 123);
   // Create object
   rc := jsonCreateObject(o := obj);
   // Add string to object
   rc := jsonSetValueString(o := obj, key := "Name", value := "Test object");
   // Replace integer in array with object
   rc := jsonSetValue(o := arr, idx := 0, value := obj);
   // Release object handle
   rc := jsonFree(o := obj);
   ...
END;

END_PROGRAM;
```

4.2.23.115 jsonSetValueString (Function)

Architecture: **NX32L**
Device support: **All**
Firmware version: **2.10.00**

jsonSetValueString stores the provided string in a JSON object or array.

When working on a JSON object, set the key parameter to the name of the value to set.
When working on a JSON array, idx must be set to the index of the element to set, and the insert parameter can be used to control if it should replace the existing element.

Input:

o : SYSHANDLE
A handle to the JSON array or object to set the value on.

idx : INT *(Default -1)*
Index to use when working on a JSON array.
Leave at -1 when working on a JSON object.

key : STRING
The name of the value when working on a JSON object.

insert : BOOL *(Default FALSE)*
When FALSE, the existing value at the given position in the array will be replaced.
Set to TRUE to insert the value in the array at the given position and shift the elements after it one position towards the end.

value : STRING
The UTF-8 string to add.

Returns: INT

1	- Success.
0	- Function is not supported.
-1	- Invalid key
-3	- Invalid handle.
-4	- The type of o does not match the expected type.
-5	- Array index is outside array.
-6	- The string is not a valid UTF-8 string or there are not enough resources available to add it.
-99	- Failed to set value

Declaration:

```
FUNCTION jsonSetValueString : INT;
VAR_INPUT
   o     : SYSHANDLE;
   idx   : INT := -1;
   key   : STRING;
   insert: BOOL := FALSE;
   value : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   arr : SYSHANDLE;
   obj : SYSHANDLE;
END_VAR;
```

```
BEGIN
   ...
   // Create array
   rc := jsonCreateArray(o := arr);
   // Add integer
   rc := jsonAddValueInt(o := arr, value := 123);
   // Create object
   rc := jsonCreateObject(o := obj);
   // Add string to object
   rc := jsonSetValueString(o := obj, key := "Name", value := "Test object");
   // Replace integer in array with object
   rc := jsonSetValue(o := arr, idx := 0, value := obj);
   // Release object handle
   rc := jsonFree(o := obj);
   ...
END;

END_PROGRAM;
```

4.2.23.16 jsonSetValueInt (Function)

Architecture: NX32L
Device support: All
Firmware version: 2.10.00

---

jsonSetValueInt stores the provided integer in a JSON object or array.

When working on a JSON object, set the key parameter to the name of the value to set.
When working on a JSON array, idx must be set to the index of the element to set, and the insert parameter can be used to control if it should replace the existing element.

Input:

o : SYSHANDLE
A handle to the JSON array or object to set the value on.

idx : INT *(Default -1)*
Index to use when working on a JSON array.
Leave at -1 when working on a JSON object.

key : STRING
The name of the value when working on a JSON object.

insert : BOOL *(Default FALSE)*
When FALSE, the existing value at the given position in the array will be replaced.
Set to TRUE to insert the value in the array at the given position and shift the elements after it one position towards the end.

value : DINT
The value to add.

Returns: INT

1	- Success.
0	- Function is not supported.
-1	- Invalid key
-3	- Invalid handle.
-4	- The type of o does not match the expected type.

-5 - Array index is outside array.
-6 - Not enough resources available to add value.
-99 - Failed to set value

Declaration:

```
FUNCTION jsonSetValueInt : INT;
VAR_INPUT
   o      : SYSHANDLE;
   idx    : INT := -1;
   key    : STRING;
   insert: BOOL := FALSE;
   value : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   arr : SYSHANDLE;
   obj : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Create array
   rc := jsonCreateArray(o := arr);
   // Add integer
   rc := jsonAddValueInt(o := arr, value := 123);
   // Create object
   rc := jsonCreateObject(o := obj);
   // Add integer to object
   rc := jsonSetValueInt(o := obj, key := "Number", value := 42);
   // Replace integer in array with object
   rc := jsonSetValue(o := arr, idx := 0, value := obj);
   // Release object handle
   rc := jsonFree(o := obj);
   ...
END;

END_PROGRAM;
```

4.2.23.17 jsonSetValueFloat (Function)

Architecture: NX32L
Device support: All
Firmware version: 2.10.00

---

jsonSetValueFloat stores the provided floating point value in a JSON object or array.

When working on a JSON object, set the key parameter to the name of the value to set.
When working on a JSON array, idx must be set to the index of the element to set, and the insert parameter can be used to control if it should replace the existing element.

Input:

o : SYSHANDLE
A handle to the JSON array or object to set the value on.

idx : INT *(Default -1)*
Index to use when working on a JSON array.
Leave at -1 when working on a JSON object.

key : STRING
The name of the value when working on a JSON object.

insert : BOOL *(Default FALSE)*
When FALSE, the existing value at the given position in the array will be replaced.
Set to TRUE to insert the value in the array at the given position and shift the elements after it one position towards the end.

value : DOUBLE
The value to add.

Returns: INT

1	- Success.
0	- Function is not supported.
-1	- Invalid key
-3	- Invalid handle.
-4	- The type of o does not match the expected type.
-5	- Array index is outside array.
-6	- Not enough resources available to add value.
-99	- Failed to set value

Declaration:

```
FUNCTION jsonSetValueFloat : INT;
VAR_INPUT
   o     : SYSHANDLE;
   idx   : INT := -1;
   key   : STRING;
   insert: BOOL := FALSE;
   value : DOUBLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   arr : SYSHANDLE;
   obj : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Create array
   rc := jsonCreateArray(o := arr);
   // Add integer
   rc := jsonAddValueInt(o := arr, value := 123);
   // Create object
   rc := jsonCreateObject(o := obj);
   // Add float to object
```

```
    rc := jsonSetValueFloat(o := obj, key := "Number", value := 12.3);
    // Replace integer in array with object
    rc := jsonSetValue(o := arr, idx := 0, value := obj);
    // Release object handle
    rc := jsonFree(o := obj);
    ...
END;

END_PROGRAM;
```

4.2.23.18 jsonSetValueBool (Function)

Architecture: NX32L
Device support: All
Firmware version: 2.10.00

---

jsonSetValueBool stores the provided boolean in a JSON object or array.

When working on a JSON object, set the key parameter to the name of the value to set.
When working on a JSON array, idx must be set to the index of the element to set, and the insert parameter can be used to control if it should replace the existing element.

Input:

o : SYSHANDLE
A handle to the JSON array or object to set the value on.

idx : INT *(Default -1)*
Index to use when working on a JSON array.
Leave at -1 when working on a JSON object.

key : STRING
The name of the value when working on a JSON object.

insert : BOOL *(Default FALSE)*
When FALSE, the existing value at the given position in the array will be replaced.
Set to TRUE to insert the value in the array at the given position and shift the elements after it one position towards the end.

value : BOOL
The value to add.

Returns: INT

1	- Success.
0	- Function is not supported.
-1	- Invalid key
-3	- Invalid handle.
-4	- The type of o does not match the expected type.
-5	- Array index is outside array.
-99	- Failed to set value

Declaration:

```
FUNCTION jsonSetValueBool : INT;
VAR_INPUT
    o     : SYSHANDLE;
    idx   : INT := -1;
```

```
   key    : STRING;
   insert: BOOL := FALSE;
   value : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   arr : SYSHANDLE;
   obj : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Create array
   rc := jsonCreateArray(o := arr);
   // Add integer
   rc := jsonAddValueInt(o := arr, value := 123);
   // Create object
   rc := jsonCreateObject(o := obj);
   // Add boolean to object
   rc := jsonSetValueBool(o := obj, key := "Enabled", value := FALSE);
   // Replace integer in array with object
   rc := jsonSetValue(o := arr, idx := 0, value := obj);
   // Release object handle
   rc := jsonFree(o := obj);
   ...
END;

END_PROGRAM;
```

4.2.23.19 jsonSetValueNull (Function)

Architecture: NX32L
Device support: All
Firmware version: 2.10.00

jsonSetValueNull stores a null-value in a JSON object or array.

When working on a JSON object, set the key parameter to the name of the value to set.
When working on a JSON array, idx must be set to the index of the element to set, and the insert parameter can be used to control if it should replace the existing element.

Input:

o : SYSHANDLE
A handle to the JSON array or object to set the value on.

idx : INT *(Default -1)*
Index to use when working on a JSON array.
Leave at -1 when working on a JSON object.

key : STRING
The name of the value when working on a JSON object.

insert : BOOL *(Default FALSE)*
When FALSE, the existing value at the given position in the array will be replaced.
Set to TRUE to insert the value in the array at the given position and shift the elements after it one position towards the end.

Returns: INT

1	- Success.
0	- Function is not supported.
-1	- Invalid key
-3	- Invalid handle.
-4	- The type of o does not match the expected type.
-5	- Array index is outside array.
-99	- Failed to set value

Declaration:

```
FUNCTION jsonSetValueNull : INT;
VAR_INPUT
   o      : SYSHANDLE;
   idx    : INT := -1;
   key    : STRING;
   insert: BOOL := FALSE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   arr : SYSHANDLE;
   obj : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Create array
   rc := jsonCreateArray(o := arr);
   // Add integer
   rc := jsonAddValueInt(o := arr, value := 123);
   // Create object
   rc := jsonCreateObject(o := obj);
   // Add null to object
   rc := jsonSetValueNull(o := obj, key := "Object");
   // Replace integer in array with object
   rc := jsonSetValue(o := arr, idx := 0, value := obj);
   // Release object handle
   rc := jsonFree(o := obj);
   ...
END;

END_PROGRAM;
```

4.2.23.20 jsonGetValue (Function)

Architecture: NX32L
Device support: All
Firmware version: 2.10.00

---

jsonGetValue reads a JSON object or array from the provided JSON object or array.

When working on a JSON object, set the key parameter to the name of the value to get.
When working on a JSON array, idx must be set to the index of the element to get.

Input:

o : SYSHANDLE
A handle to the JSON array or object to work on.

idx : INT *(Default -1)*
Index to use when working on a JSON array.

key : STRING
The name of the value when working on a JSON object.

Output:

value : SYSHANDLE
The value of the element. Must be freed with jsonFree when no longer needed.

Returns: INT

1	- Success.
0	- Function is not supported.
-1	- Invalid key
-2	- Value not found
-3	- Invalid handle.
-4	- The type of o does not match the expected type.
-5	- Array index is outside array.

Declaration:

```
FUNCTION jsonGetValue : INT;
VAR_INPUT
   o     : SYSHANDLE;
   idx   : INT := -1;
   key   : STRING;
   value : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   str : STRING;
   arr : SYSHANDLE;
   obj : SYSHANDLE;
END_VAR;
```

```
BEGIN
   ...
   // Get object from array
   rc := jsonGetValue(o := arr, idx := 0, value := obj);
   // Get name from object
   rc := jsonGetString(o := obj, key := "Name", value := str);
   // Release object handle
   rc := jsonFree(o := obj);
   ...
END;

END_PROGRAM;
```

4.2.23.21 jsonGetString (Function)

Architecture: NX32L
Device support: All
Firmware version: 2.10.00

---

jsonGetString reads a string from the provided JSON object or array.

When working on a JSON object, set the key parameter to the name of the value to get.
When working on a JSON array, idx must be set to the index of the element to get.

Input:

o : SYSHANDLE
A handle to the JSON array or object to work on.

idx : INT *(Default -1)*
Index to use when working on a JSON array.

key : STRING
The name of the value when working on a JSON object.

Output:

value : STRING
The value of the element.

Returns: INT

1	- Success.
0	- Function is not supported.
-1	- Invalid key
-2	- Value not found
-3	- Invalid handle.
-4	- The type of o does not match the expected type.
-5	- Array index is outside array.

Declaration:

```
FUNCTION jsonGetString : INT;
VAR_INPUT
   o     : SYSHANDLE;
   idx   : INT := -1;
   key   : STRING;
   value : ACCESS STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   str : STRING;
   arr : SYSHANDLE;
   obj : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Get object from array
   rc := jsonGetValue(o := arr, idx := 0, value := obj);
   // Get name from object
   rc := jsonGetString(o := obj, key := "Name", value := str);
   // Release object handle
   rc := jsonFree(o := obj);
   ...
END;

END_PROGRAM;
```

4.2.23.22 jsonGetInt (Function)

Architecture: NX32L
Device support: All
Firmware version: 2.10.00

jsonGetInt reads an integer from the provided JSON object or array.

When working on a JSON object, set the key parameter to the name of the value to get.
When working on a JSON array, idx must be set to the index of the element to get.

Input:

o : SYSHANDLE
A handle to the JSON array or object to work on.

idx : INT *(Default -1)*
Index to use when working on a JSON array.

key : STRING
The name of the value when working on a JSON object.

Output:

value : DINT
The value of the element.

Returns: INT

1	- Success.
0	- Function is not supported.
-1	- Invalid key
-2	- Value not found
-3	- Invalid handle.

-4 - The type of o does not match the expected type.
-5 - Array index is outside array.

Declaration:

```
FUNCTION jsonGetInt : INT;
VAR_INPUT
   o     : SYSHANDLE;
   idx   : INT := -1;
   key   : STRING;
   value : ACCESS DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   val : DINT;
   arr : SYSHANDLE;
   obj : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Get object from array
   rc := jsonGetValue(o := arr, idx := 0, value := obj);
   // Get integer from object
   rc := jsonGetInt(o := obj, key := "Number", value := val);
   // Release object handle
   rc := jsonFree(o := obj);
   ...
END;

END_PROGRAM;
```

4.2.23.23 jsonGetFloat (Function)

Architecture: NX32L
Device support: All
Firmware version: 2.10.00

jsonGetFloat reads a floating point value from the provided JSON object or array.

When working on a JSON object, set the key parameter to the name of the value to get.
When working on a JSON array, idx must be set to the index of the element to get.

Input:

o : SYSHANDLE
A handle to the JSON array or object to work on.

idx : INT *(Default -1)*
Index to use when working on a JSON array.

key : STRING
The name of the value when working on a JSON object.

Output:
value : DOUBLE
The value of the element.

Returns: INT
1 - Success.
0 - Function is not supported.
-1 - Invalid key
-2 - Value not found
-3 - Invalid handle.
-4 - The type of o does not match the expected type.
-5 - Array index is outside array.

Declaration:
```
FUNCTION jsonGetFloat : INT;
VAR_INPUT
   o     : SYSHANDLE;
   idx   : INT := -1;
   key   : STRING;
   value : ACCESS DOUBLE;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   val : DOUBLE;
   arr : SYSHANDLE;
   obj : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Get object from array
   rc := jsonGetValue(o := arr, idx := 0, value := obj);
   // Get float from object
   rc := jsonGetFloat(o := obj, key := "Number", value := val);
   // Release object handle
   rc := jsonFree(o := obj);
   ...
END;

END_PROGRAM;
```

4.2.23.24 jsonGetBool (Function)

Architecture: NX32L
Device support: All
Firmware version: 2.10.00

---

jsonGetBool reads a boolean value from the provided JSON object or array.

When working on a JSON object, set the key parameter to the name of the value to get.
When working on a JSON array, idx must be set to the index of the element to get.

Input:

o : SYSHANDLE
A handle to the JSON array or object to work on.

idx : INT *(Default -1)*
Index to use when working on a JSON array.

key : STRING
The name of the value when working on a JSON object.

Output:

value : BOOL
The value of the element.

Returns: INT

1	- Success.
0	- Function is not supported.
-1	- Invalid key
-2	- Value not found
-3	- Invalid handle.
-4	- The type of o does not match the expected type.
-5	- Array index is outside array.

Declaration:

```
FUNCTION jsonGetBool : INT;
VAR_INPUT
   o     : SYSHANDLE;
   idx   : INT := -1;
   key   : STRING;
   value : ACCESS BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   val : BOOL;
   arr : SYSHANDLE;
   obj : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Get object from array
   rc := jsonGetValue(o := arr, idx := 0, value := obj);
   // Get boolean from object
   rc := jsonGetBool(o := obj, key := "Enabled", value := val);
   // Release object handle
   rc := jsonFree(o := obj);
   ...
END;
```

```
END_PROGRAM;
```

4.2.23.25 jsonDeleteValue (Function)

```
Architecture:       NX32L
Device support:     All
Firmware version:   2.10.00
```

jsonDeleteValue removes specified element from a JSON object or array.

When working on a JSON object, set the key parameter to the name of the value to remove.
When working on a JSON array, idx must be set to the index of the element to remove.

Input:

o : SYSHANDLE
A handle to the JSON array or object to remove the element from.

idx : INT *(Default -1)*
Index to use when working on a JSON array.
Leave at -1 when working on a JSON object.

key : STRING
The name of the value when working on a JSON object.

Returns: INT

1	- Success.
0	- Function is not supported.
-1	- Invalid key
-2	- Element not found.
-3	- Invalid handle.
-4	- The type of o does not match the expected type.
-5	- Array index is outside array.

Declaration:

```
FUNCTION jsonDeleteValue : INT;
VAR_INPUT
   o     : SYSHANDLE;
   idx   : INT := -1;
   key   : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   arr : SYSHANDLE;
END_VAR;

BEGIN
```

```
    ...
    // Delete element from array
    rc := jsonDeleteValue(o := arr, idx := 0);
    ...
END;

END_PROGRAM;
```

4.2.23.26 jsonToString (Function)

Architecture: NX32L
Device support: All
Firmware version: 2.10.00

---

jsonToString creates a string representation of the JSON structure.
If indent is set to 0, it will be a compact string with no indentation or newlines, otherwise it will use newlines between items and use indentation.

Input:

o : SYSHANDLE
A handle to the JSON structure.

indent: SINT *(0..31, Default 3)*
The number of spaces to use for indentation. If set to 0, the string will not use indentation and will be on a single line.

Output:

str : STRING
The generated string.

Returns: INT

1	- Success
0	- Function is not supported.
-1	- Invalid indentation.
-3	- Invalid handle.
-99	- Failed to create string, structure might be corrupt.

Declaration:

```
FUNCTION jsonToString : INT;
VAR_INPUT
    o      : SYSHANDLE;
    indent : INT := 3;
    str    : ACCESS STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
    rc  : INT;
    obj : SYSHANDLE;
    str : STRING;
END_VAR;
```

```
BEGIN
   ...
   // Create structure from string
   rc := jsonFromString(o := obj, str := "[{$"name$":$"Test Object$"}]");

   // Dump structure to device output
   rc := jsonToString(o := obj, indent:=3, str := str);
   DebugMsg(message:=str);
   ...
END;

END_PROGRAM;
```

4.2.23.27 jsonFromString (Function)

Architecture: NX32L
Device support: All
Firmware version: 2.10.00

---

jsonFromString creates a new JSON structure by parsing the given string.

When the JSON structure is no longer needed, it must be released using jsonFree.

When needing to create large JSON structures with mostly fixed data, one option is to use strTemplateCreate to add the dynamic data to a string with custom JSON data, which can then be used as input to jsonFromString.

Input:

str : STRING
JSON string to create the structure from.

Output:

o : SYSHANDLE
A handle to the new JSON structure.

Returns: INT

1	- Success
0	- Function is not supported.
-6	- Could not create JSON structure, there may be too many in use.
-99	- Failed to create JSON structure, string might not be valid JSON.

Declaration:

```
FUNCTION jsonFromString : INT;
VAR_INPUT
   str    : STRING;
   o      : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
```

```
VAR
   rc  : INT;
   obj : SYSHANDLE;
   str : STRING;
END_VAR;

BEGIN
   str := STRING_BEGIN
{
   "glossary":{
      "title":"example glossary",
      "GlossDiv":{
         "title":"S",
         "GlossList":{
            "GlossEntry":{
               "ID":"SGML",
               "SortAs":"SGML",
               "GlossTerm":"Standard Generalized Markup Language",
               "Acronym":"SGML",
               "Abbrev":"ISO 8879:1986",
               "GlossDef":{
                  "para":"A meta-markup language, used to create markup
languages such as DocBook.",
                  "GlossSeeAlso":[
                     "GML",
                     "XML"
                  ]
               },
               "GlossSee":"markup"
            }
         }
      }
   }
}
STRING_END;

   ...
   // Create structure from string
   rc := jsonFromString(o := obj, str := str);

   // Dump structure to device output
   rc := jsonToString(o := obj, indent:=3, str := str);
   DebugMsg(message:=str);
   ...
END;

END_PROGRAM;
```

4.2.23.28 jsonHandleStats (Functionblock)

Architecture: NX32L
Device support: All
Firmware version: 2.10.00

---

jsonHandleStats reports some information about the JSON handles, to help with detecting leaking handles.

Input:

None

Output:

status : INT
The status is 1 when the data is valid.

no_handles : INT
Number of handles currently in use.

Declaration:

```
FUNCTION_BLOCK jsonHandleStats;
VAR_OUTPUT
   status     : INT;
   no_handles : INT;
END_VAR;
```

Example:

Please see the "Examples - REST Example"

4.2.24 jwt: JSON Web Tokens

4.2.24.1 jwt: JSON Web Tokens

The JSON Web Tokens(JWT) functions are used to encode and decode JWT objects, which are used for e.g. authentication. JWT is specified in RFC 7519.
A JSON Web Token is a string that contains three base64url encoded parts separated by periods. The first part is the header includes information about which algorithm is used to generate the signature, the second part is the payload that specifies a number of claims. The last part is optional and is the signature of the token that verifies that the token is valid.

As the tokens can potentially be very long strings, these functions require that the project is built with Large String Support.

The following functions are used to work with the JWT objects:

- jwtCreate — Creates a new, empty JWT object.
- jwtFree — Releases a JWT object.
- jwtDecodeAsym — Creates a JWT object from a token using asymmetrical encryption.
- jwtDecodeSym — Creates a JWT object from a token using symmetrical encryption.
- jwtAlgGet — Read the used encryption algorithm.
- jwtAlgSetAsym — Set the encryption algorithm to an asymmetrical one.
- jwtAlgSetSym — Set the encryption algorithm to a symmetrical one.
- jwtEncode — Create an encoded JWT token from the JWT object.

The following functions are used to manipulate the JWT headers:

- jwtHeaderAdd — Adds a new string header to the JWT object
- jwtHeaderAddBool — Adds a new boolean header to the JWT object
- jwtHeaderAddDint — Adds a new integer header to the JWT object
- jwtHeaderAddJSON — Adds the headers from a JSON encoded string to the JWT object
- jwtHeaderDelete — Deletes a header from the JWT object
- jwtHeaderGet — Reads a string header from the JWT object.
- jwtHeaderGetBool — Reads a boolean header from the JWT object.
- jwtHeaderGetDint — Reads an integer header from the JWT object.
- jwtHeaderGetJSON — Reads a header from the JWT object as a JSON encoded string.

The following functions are used to manipulate the JWT claims:

- jwtClaimAdd — Adds a new string claim to the JWT object
- jwtClaimAddBool — Adds a new boolean claim to the JWT object
- jwtClaimAddDint — Adds a new integer claim to the JWT object
- jwtClaimAddJSON — Adds the claims from a JSON encoded string to the JWT object
- jwtClaimDelete — Deletes a claim from the JWT object
- jwtClaimGet — Reads a string claim from the JWT object.
- jwtClaimGetBool — Reads a boolean claim from the JWT object.
- jwtClaimGetDint — Reads an integer claim from the JWT object.
- jwtClaimGetJSON — Reads a claim from the JWT object as a JSON encoded string.

4.2.24.2 jwtCreate (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.70.00

---

jwtCreate creates a new, empty JWT object.

When the JWT object is no longer needed, it must be released using jwtFree.

Input:
None

Output:
jwt : SYSHANDLE
A handle to the JWT object.

Returns: **INT**
1 - Success
0 - Function is not supported.
-2 - Could not allocate JWT, there may be too many in use.
-99 - Failed to create JWT.

Declaration:

```
FUNCTION jwtCreate : INT;
VAR_INPUT
   jwt    : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   jwt : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Create JWT
   rc := jwtCreate(jwt := jwt);
   ...
END;

END_PROGRAM;
```

4.2.24.3 jwtFree (Function)

```
Architecture:       NX32L
Device support:     All
Firmware version:   1.70.00
```

jwtFree releases the resources used by the JWT object.

Input:

jwt : SYSHANDLE
A handle the JWT object to release. Will be made invalid once it has been released.

Returns: INT

1 - Success
0 - Function is not supported.

Declaration:

```
FUNCTION jwtFree : INT;
VAR_INPUT
   jwt   : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   jwt : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Release JWT
   rc := jwtFree(jwt := jwt);
   ...
END;

END_PROGRAM;
```

4.2.24.4 jwtDecodeAsym (Function)

```
Architecture:       NX32L
Device support:     All
Firmware version:   1.70.00
```

jwtDecodeAsym decodes a token into a JWT object.
jwtDecodeAsym is used for asymmetrical encryption and unencrypted tokens. For symmetrical encryption, see jwtDecodeSym.

When the JWT object is no longer needed, it must be released using jwtFree.

Note: To be sure that the signature was valid, it is important to check that it used the expected encryption algorithm with jwtAlgGet.

Input:

token : STRING
The token to decode.

cert : STRING
The name of the certificate to use for decoding the token. If not provided, no validation of the signature will be done.

Output:

jwt : SYSHANDLE
A handle to the decoded JWT object.

Returns: INT

1	- Success
0	- Function is not supported.
-1	- Invalid token.
-2	- Could not allocate JWT, there may be too many in use.
-4	- Could not find certificate.
-5	- Failed to parse certificate.
-99	- Failed to decode JWT.

Declaration:

```
FUNCTION jwtDecodeAsym : INT;
VAR_INPUT
   jwt   : ACCESS SYSHANDLE;
   token : STRING;
   cert  : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   jwt : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Decode JWT
   rc := jwtDecodeAsym(jwt := jwt,
      token :=
"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpv
aG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.ySDdaDkS5f0NlFCS2gg-
hwZ82kYbBjJ6kwL1iqVXPMF6cNprhQntupGJyXtmtETicNq82eV1zd8vtYwUhkJIUX-
V8NavW65E0AWS_Qdof2jyyIZangjBL5q7cV-
qDbRmnygaHwz4G5x4NyCWeoWfkvgzdBEoN98a62OwpmghrPhfKur7ZXhmXXyKro5V4fcTHxD5bLCDM
3wd8BREfs8AQHt4jxwYgZwgwCp1739rSUSW-
LRKNaOxQNKP062QvZYEEvwD0557Eo2Pj5gNGFqKMtYDaSfydIPhIVfZbsEeJ-
6ITtPB5hmMuXOONFT2ARgxi8qW26FSsz6ZKba2HMDtBg",
      cert := "enc_pub");
   ...
END;
```

```
END_PROGRAM;
```

4.2.24.5 jwtDecodeSym (Function)

```
Architecture:       NX32L
Device support:     All
Firmware version:   1.70.00
```

---

jwtDecodeSym decodes a token into a JWT object.
jwtDecodeSym is used for symmetrical encryption and unencrypted tokens. For asymmetrical encryption, see jwtDecodeAsym.

When the JWT object is no longer needed, it must be released using jwtFree.

Note: To be sure that the signature was valid, it is important to check that it used the expected encryption algorithm with jwtAlgGet.

Input:

token : STRING
The token to decode.

key : PTR
The key to use for decoding the token. If not provided, no validation of the signature will be done.

key_len : INT
The size of the key.

Output:

jwt : SYSHANDLE
A handle to the decoded JWT object.

Returns: INT

```
   1   - Success
   0   - Function is not supported.
  -1   - Invalid token.
  -2   - Could not allocate JWT, there may be too many in use.
 -99   - Failed to decode JWT.
```

Declaration:

```
FUNCTION jwtDecodeSym : INT;
VAR_INPUT
   jwt     : ACCESS SYSHANDLE;
   token   : STRING;
   key     : PTR;
   key_len : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM ex;
// These are the local variables of the program block
VAR
   rc       : INT;
   str      : STRING;
   jwt      : SYSHANDLE;
```

```
   key      : ARRAY[1..32] OF BYTE;
   key_len  : INT;
   token    : STRING;

END_VAR;

   // The next code will only be executed once after the program starts

   // Set up key
   key_len := strLen(str:="your-256-bit-secret");
   strToMemory(dst:=ADDR(key), str:="your-256-bit-secret", len := key_len);

   token :=
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpv
aG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
";

   // Create JWT object from token
   rc := jwtDecodeSym(jwt:=jwt, token:=token, key := ADDR(key), key_len :=
key_len);
   DebugFmt(message:="jwtDecodeSym(): \1, ", v1:=rc);

   IF rc = 1 THEN
      // Show the used algorithm
      rc := jwtAlgGet(jwt:=jwt);
      DebugFmt(message:="jwtGetAlg: \1", v1:=rc);

      // Print header
      rc := jwtHeaderGetJSON(jwt:=jwt, value:=str);
      DebugFmt(message:="jwtHeaderGetJSON: \1, "+str, v1:=rc);

      // Print claim
      rc := jwtClaimGetJSON(jwt:=jwt, value:=str);
      DebugFmt(message:="jwtClaimGetJSON: \1, "+str, v1:=rc);

      // clean up
      rc := jwtFree(jwt:=jwt);
      DebugFmt(message:="jwtFree: \1", v1:=rc);
   END_IF;

BEGIN
// Code from this point until END will be executed repeatedly


END;
END_PROGRAM;
```

4.2.24.6 jwtAlgGet (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.70.00

---

jwtAlgGet returns the encryption algorithm used by the JWT object.

Should be used every time a JWT has been decoded with jwtDecodeAsym and jwtDecodeSym to validate that it was correctly signed.
Can also be used to get the value set with jwtAlgSetAsym and jwtAlgSetSym.

Input:

jwt : SYSHANDLE
The JWT object to check.

Returns: INT

The encryption algorithm used:

9	- ES512
8	- ES384
7	- ES256
6	- RS512
5	- RS384
4	- RS256
3	- HS512
2	- HS384
1	- HS256
0	- No encryption
-2	- Could not find JWT object.

Declaration:

```
FUNCTION jwtAlgGet : INT;
VAR_INPUT
   jwt   : SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   jwt : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Get algorithm
   rc := jwtAlgGet(jwt := jwt);
   ...
END;

END_PROGRAM;
```

4.2.24.7 jwtAlgSetAsym (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.70.00

---

jwtAlgSetAsym configures the JWT object to use an asymmetrical encryption.
Mainly for use with JWT objects created with jwtCreate.
To configure a symmetrical encryption, see jwtAlgSetSym.

Input:

jwt : SYSHANDLE
The JWT object to set the encryption algorithm for.

type : INT *default 0*
The kind of algorithm to use:

0 - No encryption
4 - RS256
5 - RS384
6 - RS512
7 - ES256
8 - ES384
9 - ES512

cert: STRING
The name of a certificate that contains the private key to use for encoding of the token.

pass: STRING
The password for the certificate.

Returns: INT

1 - Success
0 - Not supported.
-2 - Could not find JWT object.
-3 - Unsupported algorithm.
-4 - Could not find certificate or certificate was not provided for algorithm that required it.
-5 - Failed to parse certificate
-99 - Failed to set algorithm.

Declaration:

```
FUNCTION jwtAlgSetAsym : INT;
VAR_INPUT
   type  : INT;
   jwt   : SYSHANDLE;
   cert  : STRING;
   pass  : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   jwt : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Set algorithm
   rc := jwtAlgSetAsym(jwt:=jwt, type := 4, cert := "enc_key", pass :=
"test");
   ...
END;

END_PROGRAM;
```

4.2.24.8 jwtAlgSetSym (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.70.00

jwtAlgSetSym configures the JWT object to use a symmetrical encryption.
Mainly for use with JWT objects created with jwtCreate.
To configure an asymmetrical encryption, see jwtAlgSetAsym.

Input:

jwt : SYSHANDLE
The JWT object to set the encryption algorithm for.

type : INT *default 0*
The kind of algorithm to use:
0 - No encryption
1 - HS256
2 - HS384
3 - HS512

key: PTR
The key to use for encoding the token.

key_len : INT
The size of the key.

Returns: INT

1 - Success
0 - Not supported.
-2 - Could not find JWT object.
-3 - Unsupported algorithm.
-99 - Failed to set algorithm.

Declaration:

```
FUNCTION jwtAlgSetSym : INT;
VAR_INPUT
   type    : INT;
   jwt     : SYSHANDLE;
   key     : PTR;
   key_len : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc      : INT;
   jwt     : SYSHANDLE;
   key     : ARRAY[0..20] OF BYTE;
   key_len : INT;
END_VAR;

BEGIN
   ...
   // Prepare key
   key_len := strlen(str:="your-256-bit-secret");
   rc := strToMemory(dst:=ADDR(key), str:="your-256-bit-secret", len :=
key_len);
```

```
   // Set algorithm
   rc := jwtAlgSetSym(jwt:=jwt, type := 1, key := ADDR(key), key_len :=
key_len);
   ...
END;

END_PROGRAM;
```

4.2.24.9 jwtEncode (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.70.00

---

jwtEncode encodes a JWT object into a token.
It uses the encryption algorithm specified in jwtAlgSetAsym and jwtAlgSetSym.

Input:

jwt : SYSHANDLE
A handle to the JWT object to encode.´

Output:

token : STRING
The generated token.

Returns: INT

1	- Success
0	- Function is not supported.
-2	- Could not find JWT.
-99	- Failed to encode JWT.

Declaration:

```
FUNCTION jwtEncode : INT;
VAR_INPUT
   jwt     : SYSHANDLE;
   token   : ACCESS STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM ex;
// These are the local variables of the program block
VAR
   rc       : INT;
   str      : STRING;
   jwt      : SYSHANDLE;
   key      : ARRAY[1..32] OF BYTE;
   key_len  : INT;

END_VAR;

   // The next code will only be executed once after the program starts

   // Set up key
   key_len := strLen(str:="your-256-bit-secret");
   strToMemory(dst:=ADDR(key), str:="your-256-bit-secret", len := key_len);
```

```
   // Create empty JWT object with default header
   rc := jwtCreate(jwt:=jwt);
   DebugFmt(message:="jwtCreate: \1", v1:=rc);

   // Add claim content
   rc := jwtClaimAddDint(jwt:=jwt, name:="iat", value:=1516239022);
   DebugFmt(message:="jwtClaimAddDint: \1", v1:=rc);

   rc := jwtClaimAdd(jwt:=jwt, name:="name", value:="John Doe");

   DebugFmt(message:="jwtClaimAdd: \1", v1:=rc);

   rc := jwtClaimAddDint(jwt:=jwt, name:="sub", value:=1234567890);

   DebugFmt(message:="jwtClaimAddDint: \1", v1:=rc);

   // set the encryption type
   rc := jwtAlgSetSym(jwt:=jwt, type := 1, key := ADDR(key), key_len :=
key_len);
   DebugFmt(message:="jwtSetAlgSym: \1", v1:=rc);

   // Generate token from the JWT object
   rc := jwtEncode(jwt:=jwt, token:=str);
   DebugFmt(message:="jwtEncode: \1, ", v1:=rc);

   DebugMsg(message:="--- Token ---");
   DebugMsg(message:=str);

   // clean up
   rc := jwtFree(jwt:=jwt);

BEGIN
// Code from this point until END will be executed repeatedly


END;
END_PROGRAM;
```

4.2.24.10 jwtHeaderAdd (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.70.00

---

jwtHeaderAdd adds a new string header to the JWT object.
If the header already exists, it must be deleted first with jwtHeaderDelete.

Input:

jwt : SYSHANDLE
A handle to the JWT object.

name : STRING
The name of the header to add.

value : STRING
The string to add.

Returns: INT

1	- Success
0	- Function is not supported.
-1	- Invalid name.
-2	- Could not find JWT object.
-6	- Header already exists.
-99	- Failed to add header.

Declaration:

```
FUNCTION jwtHeaderAdd : INT;
VAR_INPUT
   jwt   : SYSHANDLE;
   name  : STRING;
   value : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   jwt : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Add header
   rc := jwtHeaderAdd(jwt := jwt, name := "name", value := "John Doe");
   ...
END;

END_PROGRAM;
```

4.2.24.11 jwtHeaderAddBool (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.70.00

---

jwtHeaderAddBool adds a new boolean header to the JWT object.
If the header already exists, it must be deleted first with jwtHeaderDelete.

Input:

jwt : SYSHANDLE
A handle to the JWT object.

name : STRING
The name of the header to add.

value : BOOL
The value to add.

Returns: INT

1 - Success
0 - Function is not supported.
-1 - Invalid name.
-2 - Could not find JWT object.
-6 - Header already exists.
-99 - Failed to add header.

Declaration:

```
FUNCTION jwtHeaderAddBool : INT;
VAR_INPUT
   jwt   : SYSHANDLE;
   name  : STRING;
   value : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   jwt : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Add header
   rc := jwtHeaderAddBool(jwt := jwt, name := "admin", value := TRUE);
   ...
END;

END_PROGRAM;
```

4.2.24.12 jwtHeaderAddDint (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.70.00

---

jwtHeaderAddDint adds a new integer header to the JWT object.
If the header already exists, it must be deleted first with jwtHeaderDelete.

Input:

jwt : SYSHANDLE
A handle to the JWT object.

name : STRING
The name of the header to add.

value : DINT
The value to add.

Returns: INT

1 - Success

0 - Function is not supported.
-1 - Invalid name.
-2 - Could not find JWT object.
-6 - Header already exists.
-99 - Failed to add header.

Declaration:

```
FUNCTION jwtHeaderAddDint : INT;
VAR_INPUT
   jwt   : SYSHANDLE;
   name  : STRING;
   value : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   jwt : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Add header
   rc := jwtHeaderAddDint(jwt := jwt, name := "admin", value := 42);
   ...
END;

END_PROGRAM;
```

4.2.24.13 jwtHeaderAddJSON (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.70.00

---

jwtHeaderAddJSON adds headers from the provided JSON encoded string to the JWT object. It will overwrite existing headers.

Input:

jwt : SYSHANDLE
A handle to the JWT object.

value : STRING
The JSON encoded string with the headers to add.

Returns: INT

1 - Success
0 - Function is not supported.
-1 - Invalid name.
-2 - Could not find JWT object.
-6 - Header already exists.

-99 - Failed to add header.

Declaration:

```
FUNCTION jwtHeaderAddJSON : INT;
VAR_INPUT
   jwt   : SYSHANDLE;
   value : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   jwt : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Add header
   rc := jwtHeaderAddJSON(jwt := jwt, value := "{$"map$@:
{$"a$":47,$"b$":5}}");
   ...
END;

END_PROGRAM;
```

4.2.24.14 jwtHeaderDelete (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.70.00

---

jwtHeaderDelete deletes a header from the JWT object.

Input:

jwt : SYSHANDLE
A handle to the JWT object.

name : STRING
The name of the header to delete. If empty, all headers will be deleted.

Returns: INT

1 - Success
0 - Function is not supported.
-2 - Could not find JWT object.
-99 - Failed to delete header.

Declaration:

```
FUNCTION jwtHeaderDelete : INT;
VAR_INPUT
   jwt   : SYSHANDLE;
```

```
   name   : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   jwt : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Delete header
   rc := jwtHeaderDelete(jwt := jwt, name := "name");
   ...
END;

END_PROGRAM;
```

4.2.24.15 jwtHeaderGet (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.70.00

jwtHeaderGet reads a string header from the JWT object.

Input:

jwt : SYSHANDLE
A handle to the JWT object.

name : STRING
The name of the header to read.

Output:

value : STRING
The value of the header.

Returns: INT

1 - Success
0 - Function is not supported.
-1 - Invalid name.
-2 - Could not find JWT object.
-4 - Could not find header.

Declaration:

```
FUNCTION jwtHeaderGet : INT;
VAR_INPUT
   jwt   : SYSHANDLE;
   name  : STRING;
   value : ACCESS STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   str : STRING;
   jwt : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Read header
   rc := jwtHeaderGet(jwt := jwt, name := "name", value := str);
   ...
END;

END_PROGRAM;
```

4.2.24.16 jwtHeaderGetBool (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.70.00

jwtHeaderGet reads a boolean header from the JWT object.

Input:

jwt : SYSHANDLE
A handle to the JWT object.

name : STRING
The name of the header to read.

Output:

value : BOOL
The value of the header.

Returns: INT

1	- Success
0	- Function is not supported.
-1	- Invalid name.
-2	- Could not find JWT object.
-4	- Could not find header.

Declaration:

```
FUNCTION jwtHeaderGetBool : INT;
VAR_INPUT
   jwt   : SYSHANDLE;
   name  : STRING;
   value : ACCESS BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   val : BOOL;
   jwt : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Read header
   rc := jwtHeaderGetBool(jwt := jwt, name := "name", value := val);
   ...
END;

END_PROGRAM;
```

4.2.24.17 jwtHeaderGetDint (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.70.00

jwtHeaderGetDint reads an integer header from the JWT object.

Input:

jwt : SYSHANDLE
A handle to the JWT object.

name : STRING
The name of the header to read.

Output:

value : DINT
The value of the header.

Returns: INT

1	- Success
0	- Function is not supported.
-1	- Invalid name.
-2	- Could not find JWT object.
-4	- Could not find header.

Declaration:

```
FUNCTION jwtHeaderGetDint : INT;
VAR_INPUT
   jwt   : SYSHANDLE;
   name  : STRING;
   value : ACCESS DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   val : DINT;
   jwt : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Read header
   rc := jwtHeaderGetDint(jwt := jwt, name := "name", value := val);
   ...
END;

END_PROGRAM;
```

4.2.24.18 jwtHeaderGetJSON (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.70.00

jwtHeaderGetJSON reads a header from the JWT object as a JSON encoded string
If no header name is provided, the entire header is returned.

Input:

jwt : SYSHANDLE
A handle to the JWT object.

name : STRING
The name of the header to read. Use an empty string to get the entire header.

Output:

value : STRING
The value of the header.

Returns: INT

1	- Success
0	- Function is not supported.
-1	- Invalid name.
-2	- Could not find JWT object.
-4	- Could not find header.

Declaration:

```
FUNCTION jwtHeaderGetJSON : INT;
VAR_INPUT
   jwt   : SYSHANDLE;
   name  : STRING;
   value : ACCESS STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   str : STRING;
   jwt : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Read header
   rc := jwtHeaderGetJSON(jwt := jwt, name := "map", value := str);
   ...
END;

END_PROGRAM;
```

4.2.24.19 jwtClaimAdd (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.70.00

jwtClaimAdd adds a new string claim to the JWT object.
If the claim already exists, it must be deleted first with jwtClaimDelete.

Input:

jwt : SYSHANDLE
A handle to the JWT object.

name : STRING
The name of the claim to add.

value : STRING
The string to add.

Returns: INT

1	- Success
0	- Function is not supported.
-1	- Invalid name.
-2	- Could not find JWT object.
-6	- Header already exists.
-99	- Failed to add claim.

Declaration:

```
FUNCTION jwtHeaderAdd : INT;
VAR_INPUT
   jwt   : SYSHANDLE;
   name  : STRING;
   value : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   jwt : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Add claim
   rc := jwtClaimAdd(jwt := jwt, name := "name", value := "John Doe");
   ...
END;

END_PROGRAM;
```

4.2.24.20 jwtClaimAddBool (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.70.00

jwtClaimAddBool adds a new boolean claim to the JWT object.
If the claim already exists, it must be deleted first with jwtClaimDelete.

Input:

jwt : SYSHANDLE
A handle to the JWT object.

name : STRING
The name of the claim to add.

value : BOOL
The value to add.

Returns: INT

1	- Success
0	- Function is not supported.
-1	- Invalid name.
-2	- Could not find JWT object.
-6	- Claim already exists.
-99	- Failed to add claim.

Declaration:

```
FUNCTION jwtClaimAddBool : INT;
VAR_INPUT
   jwt   : SYSHANDLE;
   name  : STRING;
   value : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   jwt : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Add claim
   rc := jwtClaimAddBool(jwt := jwt, name := "admin", value := TRUE);
   ...
END;

END_PROGRAM;
```

4.2.24.21 jwtClaimAddDint (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.70.00

jwtHeaderAddDint adds a new integer claim to the JWT object.
If the claim already exists, it must be deleted first with jwtClaimDelete.

Input:

jwt : SYSHANDLE
A handle to the JWT object.

name : STRING
The name of the claim to add.

value : DINT
The value to add.

Returns: INT

1	- Success
0	- Function is not supported.
-1	- Invalid name.
-2	- Could not find JWT object.
-6	- Claim already exists.
-99	- Failed to add claim.

Declaration:

```
FUNCTION jwtClaimAddDint : INT;
VAR_INPUT
   jwt   : SYSHANDLE;
   name  : STRING;
   value : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   jwt : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Add claim
   rc := jwtClaimAddDint(jwt := jwt, name := "admin", value := 42);
   ...
END;

END_PROGRAM;
```

4.2.24.22 jwtClaimAddJSON (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.70.00

jwtClaimAddJSON adds claims from the provided JSON encoded string to the JWT object.
It will overwrite existing claims.

Input:

jwt : SYSHANDLE
A handle to the JWT object.

value : STRING
The JSON encoded string with the claims to add.

Returns: INT

1	- Success
0	- Function is not supported.
-1	- Invalid name.
-2	- Could not find JWT object.
-6	- Claim already exists.
-99	- Failed to add claim.

Declaration:

```
FUNCTION jwtClaimAddJSON : INT;
VAR_INPUT
   jwt   : SYSHANDLE;
   value : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
```

```
   rc  : INT;
   jwt : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Add claim
   rc := jwtClaimAddJSON(jwt := jwt, value := "{$"map$@:{$"a$":47,$"b$":5}}");
   ...
END;

END_PROGRAM;
```

4.2.24.23 jwtClaimDelete (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.70.00

---

jwtClaimDelete deletes a claim from the JWT object.

Input:

jwt : SYSHANDLE
A handle to the JWT object.

name : STRING
The name of the claim to delete. If empty, all claims will be deleted.

Returns: INT

1	- Success
0	- Function is not supported.
-2	- Could not find JWT object.
-99	- Failed to delete claim.

Declaration:

```
FUNCTION jwtClaimDelete : INT;
VAR_INPUT
   jwt   : SYSHANDLE;
   name  : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   jwt : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Delete claim
   rc := jwtClaimDelete(jwt := jwt, name := "name");
   ...
```

```
END;

END_PROGRAM;
```

4.2.24.24 jwtClaimGet (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.70.00

---

jwtClaimGet reads a string claim from the JWT object.

Input:

jwt : SYSHANDLE
A handle to the JWT object.

name : STRING
The name of the claim to read.

Output:

value : STRING
The value of the claim.

Returns: INT

1	- Success
0	- Function is not supported.
-1	- Invalid name.
-2	- Could not find JWT object.
-4	- Could not find claim.

Declaration:

```
FUNCTION jwtClaimGet : INT;
VAR_INPUT
   jwt   : SYSHANDLE;
   name  : STRING;
   value : ACCESS STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   str : STRING;
   jwt : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Read claim
   rc := jwtClaimGet(jwt := jwt, name := "name", value := str);
   ...
END;
```

```
END_PROGRAM;
```

4.2.24.25 jwtClaimGetBool (Function)

```
Architecture:       NX32L
Device support:     All
Firmware version:   1.70.00
```

---

jwtClaimGet reads a boolean claim from the JWT object.

Input:

jwt : SYSHANDLE
A handle to the JWT object.

name : STRING
The name of the claim to read.

Output:

value : BOOL
The value of the claim.

Returns: INT

1	- Success
0	- Function is not supported.
-1	- Invalid name.
-2	- Could not find JWT object.
-4	- Could not find claim.

Declaration:

```
FUNCTION jwtClaimGetBool : INT;
VAR_INPUT
   jwt   : SYSHANDLE;
   name  : STRING;
   value : ACCESS BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   val : BOOL;
   jwt : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Read claim
   rc := jwtClaimGetBool(jwt := jwt, name := "name", value := val);
   ...
END;
```

```
END_PROGRAM;
```

4.2.24.26 jwtClaimGetDint (Function)

```
Architecture:       NX32L
Device support:     All
Firmware version:   1.70.00
```

---

jwtClaimGetDint reads an integer claim from the JWT object.

Input:

jwt : SYSHANDLE
A handle to the JWT object.

name : STRING
The name of the claim to read.

Output:

value : DINT
The value of the claim.

Returns: INT

1	- Success
0	- Function is not supported.
-1	- Invalid name.
-2	- Could not find JWT object.
-4	- Could not find claim.

Declaration:

```
FUNCTION jwtClaimGetDint : INT;
VAR_INPUT
   jwt   : SYSHANDLE;
   name  : STRING;
   value : ACCESS DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   val : DINT;
   jwt : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Read claim
   rc := jwtClaimGetDint(jwt := jwt, name := "name", value := val);
   ...
END;
```

```
END_PROGRAM;
```

4.2.24.27 jwtClaimGetJSON (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.70.00

---

jwtClaimGetJSON reads a claim from the JWT object as a JSON encoded string
If no claim name is provided, the entire claim is returned.

Input:

jwt : SYSHANDLE
A handle to the JWT object.

name : STRING
The name of the claim to read. Use an empty string to get the entire claim.

Output:

value : STRING
The value of the claim.

Returns: INT

1 - Success
0 - Function is not supported.
-1 - Invalid name.
-2 - Could not find JWT object.
-4 - Could not find claim.

Declaration:

```
FUNCTION jwtClaimGetJSON : INT;
VAR_INPUT
   jwt   : SYSHANDLE;
   name  : STRING;
   value : ACCESS STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   str : STRING;
   jwt : SYSHANDLE;
END_VAR;

BEGIN
   ...
   // Read claim
   rc := jwtClaimGetJSON(jwt := jwt, name := "map", value := str);
   ...
END;
```

```
END_PROGRAM;
```

4.2.25 mbus: M-Bus

4.2.25.1 mbus: M-Bus

M-Bus is a series of European standards for remote reading of water, gas, and electricity meters (EN 13757).

This API has support for two different types of M-Bus: Wired M-Bus and Wireless M-Bus.

Wired M-Bus uses two wires to communicate with a number of devices that might be powered by the master.
The master sends a request and the slave devices respond.
The slave devices can be addressed using either a short primary address which is an integer typically in the range 1-250, or the secondary address which is a unique 16-byte string. See mbusScan for details about the format of the secondary address.

Wireless M-Bus uses RF to transmit the messages.
The slaves send data periodically without the master needing to do anything.
Wireless M-Bus only uses secondary addresses.
Wireless M-Bus data is often encrypted, so the correct decryption key is needed to parse the data.
When a Wireless M-Bus frame is received, it is kept in an internal buffer until it is either read with mbusRecordSlaveInfo or mbusReceive or it is overwritten by a newer message when the buffer is full.

The following functions are used to access both types of M-Bus:

- mbusOpen — Opens a connection to an M-Bus interface,
- mbusClose — Closes an M-Bus connection.
- mbusRecordSlaveInfo — Get information about slave device.
- mbusRecordCount — Get number of records on slave device.
- mbusRecordGetInfo — Get information about a record.
- mbusRecordGetType — Get record data type.
- mbusRecordGetBuffer — Read record value into a buffer.
- mbusRecordGetInt — Read record value as a DINT.
- mbusRecordGetIntLarge — Read large record value as two DINTs
- mbusRecordGetFloat — Read record value as a float.
- mbusRecordGetString — Read record value as a string.
- mbusRecordGetLinsec — Read record value as a linsec value.
- mbusSend — Send a raw M-Bus frame.
- mbusReceive — Receive a raw M-Bus frame.

The following functions are only used for wired M-Bus:

- mbusBaudSet — Change the baud rate for an M-Bus connection.
- mbusScan — Scan for M-bus slave devices.
- mbusScanStop — Stop the M-Bus scan.
- mbusSlaveConfigBaud — Change baud rate on slave device.
- mbusSlaveConfigAddress — Change primary address on slave device.
- mbusSlaveConfigReset — Send application reset to slave device.
- mbusDataRequest — Request data from slave device.
- mbusSelectSecondary — Select a secondary address to be made available as primary address 253, for use with mbusSend.

The following functions are only used for wireless M-Bus:

- mbusSlaveRegister — Register a wireless M-Bus slave device with optional decryption key.

• mbusSlaveUnregister	Remove registration.
• mbusFilterEnable	Control if data from all or only registered slaves should be received.
• mbusDataReceive	Receive a frame from wireless M-Bus
• mbusSetSenderAddress	Set address to use as sender when sending Wireless M-Bus frames with mbusSend.

4.2.25.2 mbusOpen (Function)

```
Architecture:       NX32L
Device support:     LX4, NX-400
Firmware version:   1.94.00
```

This function is used to open a connection to an M-Bus or Wireless M-Bus interface.

Input:

type : SINT *default 1*
The type of connection to create:
1 - On-board M-Bus interface (Only supported on the RTCU LX4)
2 - Wireless M-Bus

baud : DINT (300,600,1200,2400,4800,9600,19200,38400) *(default 2400)*
The baud rate to use for Wired M-Bus.
Note that not all baud rates are supported on all interfaces.

mode : SINT *default _MBUS_MODE_T1*
The operation mode to use for Wireless M-Bus:
_MBUS_MODE_T1 - T1
_MBUS_MODE_T2 - T2
_MBUS_MODE_S1 - S1
_MBUS_MODE_S2 - S2
_MBUS_MODE_R - R
_MBUS_MODE_T1_C - Reception of both T1 and C
_MBUS_MODE_T2_C - Reception of both T2 and C

Output:

handle : SYSHANDLE
A handle to the connection

Returns: INT

1 - Success
0 - Not supported
-1 - Invalid handle(handle is in use)
-2 - No more connections available
-3 - Interface is already in use
-4 - Invalid baud rate
-5 - Invalid interface type.
-8 - Failed to open connection

Declaration:

```
FUNCTION mbusOpen : INT;
VAR_INPUT
   type     : SINT := 1;
   baud     : DINT := 2400;
   mode     : SINT := _MBUS_MODE_T1;
   handle   : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   mb : SYSHANDLE;
   rc : INT;
END_VAR;

BEGIN
   ...
   // Open wired M-Bus interface
   rc := mbusOpen(handle := mb);
   ...
END;
END_PROGRAM;
```

4.2.25.3 mbusClose (Function)

Architecture: NX32L
Device support: LX4,NX-400
Firmware version: 1.94.00

This function is used to close an M-Bus connection.

Input:

handle : SYSHANDLE
A handle to the connection to close.

Returns: INT

1 - Success
0 - Not supported
-1 - Invalid handle

Declaration:

```
FUNCTION mbusClose : INT;
VAR_INPUT
   handle   : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
```

```
VAR
   mb : SYSHANDLE;
   rc : INT;
END_VAR;

BEGIN
   ...
   // Open wired M-Bus interface
   rc := mbusOpen(handle := mb);
   ...
   // Close wired M-Bus interface
   rc := mbusClose(handle := mb);
   ...
END;
END_PROGRAM;
```

4.2.25.4 mbusBaudSet (Function)

Architecture: NX32L
Device support: LX4
Firmware version: 1.94.00

This function is used to change the baud rate of a wired M-Bus connection.

Input:

handle : SYSHANDLE
A handle to the connection

baud : DINT (300,600,1200,2400,4800,9600,19200,38400) *(default 2400)*
The new baud rate to use.

Returns: INT

1	- Success
0	- Not supported
-1	- Invalid handle
-4	- Invalid baud rate
-5	- Invalid interface type.
-20	- Generic error

Declaration:

```
FUNCTION mbusBaudSet : INT;
VAR_INPUT
   handle   : SYSHANDLE;
   baud     : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   mb : SYSHANDLE;
   rc : INT;
END_VAR;

BEGIN
```

```
    ...
    // Open wired M-Bus interface @2400 baud
    rc := mbusOpen(handle := mb);
    ...
    // Increase speed to 9600 baud
    rc := mbusBaudSet(handle := mb, baud := 9600);
    ...
END;
END_PROGRAM;
```

4.2.25.5 mbusScan (Function)

Architecture: NX32L
Device support: LX4
Firmware version: 1.94.00

This function is used to scan for slaves on wired M-Bus.
The search for primary addresses is done by checking every address. Depending on the baud rate, this can take a few minutes.

The search for secondary addresses is done using a mask, where only the address space containing devices will be scanned. Depending on the baudrate and the devices on the bus, the scan can take many minutes.

Input:

handle : SYSHANDLE
A handle to the connection

smode : SINT *default 1*
The scan mode determining the address type to scan for:
1 - Primary address (0-250)
2 - Secondary (16 character id)

mask : STRING *default "FFFFFFFFFFFFFFFF"*
String containing the mask to use for filtering the addresses in scan mode 2.
The mask must be 16 characters long and must use 'F' as placeholder for unknown values.
The mask contains:
- 4 bytes with the device ID, encoded as BCD.
- 2 bytes for the manufacturer ID as binary.
- 1 byte for the device version as binary.
- 1 byte for the device media as binary.

For a device with device ID 12345678, manufacturer ID 0x1C36, version 123 and medium 7, the secondary address would be "12345678361C7B07".
To search for a device ID 12345678, use mask "12345678FFFFFFFF".
To search for devices starting with "123", use mask "123FFFFFFFFFFFFF".
For the device ID, each nibble can be replaced with 'F', the rest of the fields can only use wildcards for the entire field, i.e. 'FF' for version or medium, 'FFFF' for manufacturer ID.

cb_found : CALLBACK
The callback function to call for every found slave. (See the mbusScanFoundCallback callback declaration below)

cb_progress : CALLBACK
Callback function to call before every search to indicate the progress. (See the mbusScanProgressCallback callback declaration below)

Returns: INT

1 - Success
0 - Not supported
-1 - Invalid handle
-5 - Invalid interface type
-7 - Invalid mask.
-9 - Communication error

Declaration:

```
FUNCTION mbusScan : INT;
VAR_INPUT
   handle      : SYSHANDLE;
   cb_found    : CALLBACK;
   cb_progress : CALLBACK;
   mask        : STRING := "FFFFFFFFFFFFFFFF";
   smode       : SINT := 1;
END_VAR;

FUNCTION CALLBACK mbusScanFoundCallback;
VAR_INPUT
   handle      : SYSHANDLE; // Handle to the connection.
   secondary   : STRING; // The secondary address of the slave. Only available
for smode 2.
   primary     : INT; // The primary address of the slave.
END_VAR;
END_FUNCTION;

FUNCTION CALLBACK mbusScanProgressCallback;
VAR_INPUT
   handle      : SYSHANDLE; // Handle to the connection.
   secondary   : STRING; // The mask of the secondary address currently being
scanned.
   primary     : INT; // The primary address currently being scanned. -1 for
secondary scan.
END_VAR;
END_FUNCTION;
```

Example:

```
INCLUDE rtcu.inc

FUNCTION CALLBACK OnScan;
VAR_INPUT
   handle      : SYSHANDLE;
   secondary   : STRING;
   primary     : INT;
END_VAR;
   DebugFmt(message:="device found: \1: "+secondary, v1:=primary);
END_FUNCTION;

FUNCTION CALLBACK OnProgress;
VAR_INPUT
   handle      : SYSHANDLE;
   secondary   : STRING;
   primary     : INT;
END_VAR;
   DebugFmt(message:="scanning: \1: "+secondary, v1:=primary);
END_FUNCTION;
```

```
PROGRAM test;
VAR
   mb : SYSHANDLE;
   rc : INT;
END_VAR;

   ...
   // Open wired M-Bus interface
   rc := mbusOpen(handle := mb);
   ...
   // Start scan for any secondary address
   rc := mbusScan(handle:=mb, cb_found:=@OnScan, cb_progress:=@OnProgress,
smode:=2, mask:="FFFFFFFFFFFFFFFF");
   DebugFmt(message:="mbusScan(): \1", v1:=rc);
BEGIN

   ...
END;
END_PROGRAM;
```

4.2.25.6 mbusScanStop (Function)

Architecture: NX32L
Device support: LX4
Firmware version: 1.94.00

This function is used to stop an M-Bus scan started with mbusScan.

Input:

handle : SYSHANDLE
A handle to the connection to stop scanning on.

Returns: INT

1	- Success
0	- Not supported
-1	- Invalid handle
-9	- Communication error

Declaration:

```
FUNCTION mbusScanStop : INT;
VAR_INPUT
   handle   : SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

FUNCTION CALLBACK OnScan;
VAR_INPUT
   handle      : SYSHANDLE;
   secondary   : STRING;
   primary     : INT;
END_VAR;
   DebugFmt(message:="device found: \1: "+secondary, v1:=primary);
END_FUNCTION;
```

```
FUNCTION CALLBACK OnProgress;
VAR_INPUT
   handle      : SYSHANDLE;
   secondary   : STRING;
   primary     : INT;
END_VAR;
   DebugFmt(message:="scanning: \1: "+secondary, v1:=primary);
END_FUNCTION;

PROGRAM test;
VAR
   mb : SYSHANDLE;
   rc : INT;
END_VAR;

   ...
   // Open wired M-Bus interface
   rc := mbusOpen(handle := mb);
   ...
   // Start scan for any secondary address
   rc := mbusScan(handle:=mb, cb_found:=@OnScan, cb_progress:=@OnProgress,
smode:=2, mask:="FFFFFFFFFFFFFFFF");
   DebugFmt(message:="mbusScan(): \1", v1:=rc);
BEGIN

   ...
END;
END_PROGRAM;
```

4.2.25.7 mbusSlaveConfigBaud (Function)

Architecture: NX32L
Device support: LX4
Firmware version: 1.94.00

This function is used to change the baud rate of a wired M-Bus slave.

Input:

handle : SYSHANDLE
A handle to the connection

baud : DINT (300,600,1200,2400,4800,9600,19200,38400)
The new baud rate to use.

pri_addr : INT *default -1*
The primary address for the slave to configure. Use -1 to use the secondary address instead.

sec_addr : STRING
The secondary address for the slave to configure. Used if the primary address is -1.

Returns: **INT**

1	- Success
0	- Not supported
-1	- Invalid handle
-4	- Invalid baud rate
-5	- Invalid interface type.
-6	- Invalid address

-9 - Communication error
-10 - Communication timeout
-11 - Invalid reply

Declaration:

```
FUNCTION mbusSlaveConfigBaud : INT;
VAR_INPUT
   handle   : SYSHANDLE;
   baud     : DINT;
   pri_addr : INT := -1;
   sec_addr : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   mb : SYSHANDLE;
   rc : INT;
END_VAR;

BEGIN
   ...
   // Open wired M-Bus interface @2400 baud
   rc := mbusOpen(handle := mb);
   ...
   // Change baud on device with primary address 11 to 9600 baud
   rc := mbusSlaveConfigBaud(handle:=mb, pri_addr:=11, baud:=9600);
   ...
   // Change speed to 9600 baud to be able to communicate with the device
   rc := mbusBaudSet(handle := mb, baud := 9600);
   ...
END;
END_PROGRAM;
```

4.2.25.8 mbusSlaveConfigAddress (Function)

Architecture: NX32L
Device support: LX4
Firmware version: 1.94.00

This function is used to change the primary address of a wired M-Bus slave.

Input:

handle : SYSHANDLE
A handle to the connection

new_addr : INT (1..250)
The new address to use.

pri_addr : INT *default -1*
The primary address for the slave to configure. Use -1 to use the secondary address instead.

sec_addr : STRING
The secondary address for the slave to configure. Used if the primary address is -1.

Returns: **INT**

1	- Success
0	- Not supported
-1	- Invalid handle
-5	- Invalid interface type.
-6	- Invalid address
-9	- Communication error
-10	- Communication timeout
-11	- Invalid reply

Declaration:

```
FUNCTION mbusSlaveConfigAddress : INT;
VAR_INPUT
   handle   : SYSHANDLE;
   pri_addr : INT := -1;
   sec_addr : STRING;
   new_addr : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   mb : SYSHANDLE;
   rc : INT;
END_VAR;

BEGIN
   ...
   // Open wired M-Bus interface @2400 baud
   rc := mbusOpen(handle := mb);
   ...
   // Change primary address for the device with secondary address
637145452d2c1b16
   rc := mbusSlaveConfigAddress(handle:=mb, new_addr:=11,
sec_addr:="637145452d2c1b16");
   ...
END;
END_PROGRAM;
```

4.2.25.9 mbusSlaveConfigReset (Function)

Architecture: NX32L
Device support: LX4
Firmware version: 1.94.00

This function sends an application reset request to the wired M-Bus slave.

Input:

handle : SYSHANDLE
A handle to the connection

pri_addr : INT ***default -1***
The primary address for the slave. Use -1 to use the secondary address instead.

sec_addr : STRING
The secondary address for the slave. Used if the primary address is -1.

subcode : INT *default -1*
Optional subcode to include in the request. It is not used if it is -1.

Returns: INT

1	- Success
0	- Not supported
-1	- Invalid handle
-5	- Invalid interface type.
-6	- Invalid address
-9	- Communication error
-10	- Communication timeout
-11	- Invalid reply

Declaration:

```
FUNCTION mbusSlaveConfigReset : INT;
VAR_INPUT
   handle   : SYSHANDLE;
   pri_addr : INT := -1;
   sec_addr : STRING;
   subcode  : INT := -1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   mb : SYSHANDLE;
   rc : INT;
END_VAR;

BEGIN
   ...
   // Open wired M-Bus interface @2400 baud
   rc := mbusOpen(handle := mb);
   ...
   // Send reset request to device with primary address 11, with subcode 5.
   rc := mbusSlaveConfigReset(handle:=mb, pri_addr:=11, subcode:=5);
   ...
END;
END_PROGRAM;
```

4.2.25.10 mbusDataRequest (Function)

Architecture: NX32L
Device support: LX4
Firmware version: 1.94.00

This function is used to request data from a wired M-Bus slave.
The slave information can be read with mbusRecordSlaveInfo.
The meter data can then be read using mbusRecordGetInfo.

Input:

handle : SYSHANDLE
A handle to the connection

pri_addr : INT *default -1*
The primary address for the slave. Use -1 to use the secondary address instead.

sec_addr : STRING
The secondary address for the slave. Used if the primary address is -1.

Returns: INT

1	- Success
0	- Not supported
-1	- Invalid handle
-5	- Invalid interface type.
-6	- Invalid address
-9	- Communication error

Declaration:

```
FUNCTION mbusDataRequest : INT;
VAR_INPUT
   handle   : SYSHANDLE;
   pri_addr : INT := -1;
   sec_addr : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   mb   : SYSHANDLE;
   rc   : INT;
   info : mbusRecordSlaveInfo;
END_VAR;

   ...
   // Open wired M-Bus interface @2400 baud
   rc := mbusOpen(handle := mb);
   ...
BEGIN
   // Request data from device
   rc := mbusDataRequest(handle := mb, pri_addr := 11);
   DebugFmt(message := "mbusDataRequest=\1", v1 := rc);

   info(handle:=mb);
   IF info.ready THEN
      DebugFmt(message:="Info for \4:", v4:=info.id);
      DebugFmt(message:=" Enc:  \1", v1:=info.enc_state);
      DebugFmt(message:=" Man:  " + info.manufacturer);
      DebugFmt(message:=" Ver:  \1", v1:=info.version);
      DebugFmt(message:=" Med:  \1", v1:=info.medium);
      DebugFmt(message:=" AN :  \1", v1:=info.accessnumber);
      DebugFmt(message:=" Sta:  \1", v1:=info.status);
      DebugFmt(message:=" Addr: " + info.sec_addr);
      DebugFmt(message:=" Sig:  \1", v1:=info.signal);
   ELSE
      DebugMsg(message:="no response");
   END_IF;
   ...
```

```
END;
END_PROGRAM;
```

4.2.25.11 mbusDataReceive (Function)

```
Architecture:       NX32L
Device support:     LX4,NX-400
Firmware version:   1.94.00
```

This function is used to receive data from a wireless M-Bus slave
The slave information can be read with mbusRecordSlaveInfo.
The meter data can then be read using mbusRecordGetInfo.

Input:

handle : SYSHANDLE
A handle to the connection

timeout : INT *default 0*
Number of seconds to wait for data from Wireless M-Bus. Use 0 seconds to return immediately if there is no data in the buffer.

filter : STRING
Specifies the address or address filter to receive frames from. If not specified, it will just receive the first frame in the buffer.

Returns: INT

1	- Success
0	- Not supported
-1	- Invalid handle
-5	- Invalid interface type.
-7	- Invalid filter.
-9	- Communication error
-10	- Timeout before data was received.
-14	- Invalid timeout value.

Declaration:

```
FUNCTION mbusDataReceive : INT;
VAR_INPUT
   handle   : SYSHANDLE;
   timeout  : INT := 0;
   filter   : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   mb   : SYSHANDLE;
   rc   : INT;
   info : mbusRecordSlaveInfo;
```

```
END_VAR;

   ...
   // Open wireless M-Bus interface in T1+C mode
   rc := mbusOpen(type:=2, handle:=mb, mode:=_MBUS_MODE_T1_C);
   DebugFmt(message:="mbusOpen(): \1", v1:=rc);
   ...
BEGIN
   // Wait for 15 seconds for data from 1234567893153303
   rc := mbusDataReceive(handle:=mb, filter:="1234567893153303", timeout:=15);
   DebugFmt(message:="mbusDataReceive: \1", v1:=rc);

   IF rc = 1 THEN
      info(handle:=mb);
      IF info.ready THEN
         DebugFmt(message:="Info for \4:", v4:=info.id);
         DebugFmt(message:=" Enc:  \1", v1:=info.enc_state);
         DebugFmt(message:=" Man:  "+info.manufacturer);
         DebugFmt(message:=" Ver:  \1", v1:=info.version);
         DebugFmt(message:=" Med:  \1", v1:=info.medium);
         DebugFmt(message:=" AN :  \1", v1:=info.accessnumber);
         DebugFmt(message:=" Sta:  \1", v1:=info.status);
         DebugFmt(message:=" Addr: "+info.sec_addr);
         DebugFmt(message:=" Sig:  \1", v1:=info.signal);
      END_IF;
      ...
   END_IF;

   ...
END;
END_PROGRAM;
```

4.2.25.12 mbusRecordSlaveInfo (Functionblock)

Architecture: NX32L
Device support: LX4,NX-400
Firmware version: 1.94.00

This function block will provide information about a slave.

For M-Bus this information must be requested with mbusDataRequest.

For Wireless M-Bus, the data must have been received with mbusDataReceive.

Input:

handle : SYSHANDLE
A handle to the connection

Output:

id : DINT
The device id / serial number of the slave.

manufacturer : STRING
The EN 61107 manufacturer ID as a string.

version : INT
The version number of the slave.

medium : INT
The medium of the received data.

pri_addr: INT
The primary address of the slave. -1 for wireless M-Bus.

sec_addr: STRING
The secondary address of the slave.

accessnumber : INT
The access number of the package.

enc_state: USINT
The encryption state of the package.
0 - Not encrypted
1 - Encrypted, decryption was successful.
2 - Decryption failed - the key may be wrong.
3 - Decryption key not found.
4 - Unknown encryption type.

signal : SINT
The signal strength of the package in dB. Only available for Wireless M-Bus. 0 if not available.

status: INT
Status field from slave.

ready : BOOL
TRUE if the information is available, FALSE if not.

Declaration:

```
FUNCTION_BLOCK mbusRecordSlaveInfo;
VAR_INPUT
   handle  : SYSHANDLE;
END_VAR;
VAR_OUTPUT
   id              : INT;
   manufacturer    : STRING;
   version         : INT;
   medium          : INT;
   pri_addr        : INT;
   sec_addr        : STRING;
   accessnumber    : INT;
   enc_state       : USINT;
   signal          : SINT;
   status          : INT;
   ready           : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   mb    : SYSHANDLE;
   rc    : INT;
```

```
   info : mbusRecordSlaveInfo;
END_VAR;
   ...
   // Open wired M-Bus interface @2400 baud
   rc := mbusOpen(handle := mb);
   ...

BEGIN
   // Request data from device
   rc := mbusDataRequest(handle := mb, pri_addr := 11);
   DebugFmt(message := "mbusDataRequest=\1", v1 := rc);

   info(handle:=mb);
   IF info.ready THEN
      DebugFmt(message:="Info for \4:", v4:=info.id);
      DebugFmt(message:=" Enc:  \1", v1:=info.enc_state);
      DebugFmt(message:=" Man:  " + info.manufacturer);
      DebugFmt(message:=" Ver:  \1", v1:=info.version);
      DebugFmt(message:=" Med:  \1", v1:=info.medium);
      DebugFmt(message:=" AN :  \1", v1:=info.accessnumber);
      DebugFmt(message:=" Sta:  \1", v1:=info.status);
      DebugFmt(message:=" Addr: " + info.sec_addr);
      DebugFmt(message:=" Sig:  \1", v1:=info.signal);
   ELSE
      DebugMsg(message:="no response");
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.25.113 mbusRecordCount (Function)

Architecture: NX32L
Device support: LX4,NX-400
Firmware version: 1.94.00

This function returns the number of records available from the slave, requested with mbusDataRequest.

Input:

handle : SYSHANDLE
A handle to the connection

Returns: INT

>0	- The number of records
0	- Not supported.
-1	- Invalid handle
-3	- No data found.

Declaration:

```
FUNCTION mbusRecordCount : INT;
VAR_INPUT
   handle   : SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   mb : SYSHANDLE;
   recInfo   : mbusRecordGetInfo;
END_VAR;

FUNCTION DumpRecords;
VAR
   rc    : INT;
   count : INT;
   i     : INT;
   str, unit : STRING;
   scale : FLOAT;
   type  : INT;
   d     : DINT;
END_VAR;
   count := mbusRecordCount(handle:=mb);
   DebugFmt(message:=" Records: \1", v1:=count);


   FOR i := 0 TO count DO
      recInfo(handle:=mb, index:=i);
      IF recInfo.ready THEN
         str := "  Record "+intToStr(v:=i)+", Type: " +
intToStr(v:=recInfo.type);

         scale:=1.0;
         unit:="";

         str := str + ", VIF: "+sintToHex(v:=recInfo.VIF)+", VIFE: "
            + sintToHex(v:=recInfo.VIFE[1])+" "+sintToHex(v:=recInfo.VIFE[2]);
         DebugFmt(message:=str);
         DebugFmt(message:="   Addr: "+recInfo.address);
         DebugFmt(message:="   Dev: \1, Func: \2, Tar \4, Stor:
"+dintToStr(v:=recInfo.storage),
                           v1:=recInfo.device, v2:=recInfo.func,
v4:=recInfo.tariff);

         DebugFmt(message:="   Type: "+recInfo.text);

         type := mbusRecordGetType(handle:=mb, index:=i);
         DebugFmt(message:="   Data Type: \1", v1:=type);

         IF recInfo.VIF = 16#14 THEN
            // Volume [1e-2  m^3]
            unit := "m^3";
            scale := 0.01;
         END_IF;


         IF type = _MBUS_TYPE_INT32 THEN
            rc := mbusRecordGetInt(handle:=mb, index:=i, value:=d);
            DebugMsg(message:="   Value:     "+floatToStr(v:=FLOAT(d)*scale)+"
"+unit);
         END_IF;

      END_IF;
   END_FOR;
END_FUNCTION;

...
```

4.2.25.14 mbusRecordGetInfo (Functionblock)

```
Architecture:       NX32L
Device support:     LX4,NX-400
Firmware version:   1.94.00
```

This function block will provide information about the records requested with mbusDataRequest or received with mbusDataReceive.

The VIF describes the unit and multiplier for the value, e.g. a VIF of 0x15 indicates that the value is in W and must be multiplied by 10.
Please refer to the M-Bus documentation for details about the VIF/VIFE tables etc.

Input:

handle : SYSHANDLE
A handle to the connection

index : INT
The index of the record to read, starting with index 1. mbusRecordCount can be used to get the total number of records.

Output:

type : SINT
The record type:
- 1 - Fixed data structure.
- 2 - Variable data structure.
- 3 - Application error. Error code can be found as value in the record.

address: STRING
The address associated with this record. For simple packages it will match the slave address, but other times, the record may have been forwarded from another meter with this address.

VIF : USINT
Value Information Field, indicates the type of value the record contains, e.g. the unit and multiplier.
Bit 7 is the extension bit. If it is set, the VIF is extended with the contents of VIFE[1].
If bit 0-6 is less than 0x7B, the primary VIF table must be used.
If the VIF is 0xFD or 0xFB, the extension VIF tables must be used to look up the value of VIFE[1].
If the VIF is 0x7F or 0xFF, the data is manufacturer specific.
If the VIF is 0x7C/0xFC, a custom string is used for the VIF, which can be found in the text variable.

VIFE : ARRAY [1..10] OF USINT
VIF Extension, indicates the type of value the record contains, e.g. the unit and multiplier, if the extension bit is set in the VIF.
If the extension bit is set in VIFE[n], VIFE[n+1] is valid.

text : STRING
String describing the unit and multiplier. If VIF is 0x7C, this was sent from the slave, otherwise this is the result of a table lookup, which does not know all the manufacturer specific values.

func : SINT
The type of value.
- 0 - Instantaneous value

1 - Minimum value
2 - Maximum value
3 - Value during error state

device : INT
Index of the sub device the record belongs to.

tariff : DINT
The tariff value for the record.

storage : DINT
The storage value for the record (lower 32 bits).

storage_upper : UINT
The storage value for the record (upper 9 bits, will normally be 0).

ready : BOOL
TRUE if the information is available, FALSE if not.

Declaration:

```
FUNCTION_BLOCK mbusRecordGetInfo;
VAR_INPUT
   handle  : SYSHANDLE;
   index   : INT;
END_VAR;
VAR_OUTPUT
   tariff   : DINT;
   storage  : DINT;
   storage_upper: UINT;
   text     : STRING;
   address  : STRING;
   device   : INT;
   func     : SINT;
   type     : SINT;
   VIFE     : ARRAY[1..10] OF USINT;
   VIF      : USINT;
   ready    : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   mb : SYSHANDLE;
   recInfo   : mbusRecordGetInfo;
END_VAR;

FUNCTION DumpRecords;
VAR
   rc    : INT;
   count : INT;
   i     : INT;
   str, unit : STRING;
   scale : FLOAT;
   type  : INT;
   d     : DINT;
END_VAR;
   count := mbusRecordCount(handle:=mb);
   DebugFmt(message:=" Records: \1", v1:=count);
```

```
   FOR i := 0 TO count DO
      recInfo(handle:=mb, index:=i);
      IF recInfo.ready THEN
         str := "  Record "+intToStr(v:=i)+", Type: " +
intToStr(v:=recInfo.type);

         scale:=1.0;
         unit:="";

         str := str + ", VIF: "+sintToHex(v:=recInfo.VIF)+", VIFE: "
            + sintToHex(v:=recInfo.VIFE[1])+" "+sintToHex(v:=recInfo.VIFE[2]);
         DebugFmt(message:=str);
         DebugFmt(message:="   Addr: "+recInfo.address);
         DebugFmt(message:="   Dev: \1, Func: \2, Tar \4, Stor:
"+dintToStr(v:=recInfo.storage),
                           v1:=recInfo.device, v2:=recInfo.func,
v4:=recInfo.tariff);

         DebugFmt(message:="   Type: "+recInfo.text);

         type := mbusRecordGetType(handle:=mb, index:=i);
         DebugFmt(message:="   Data Type: \1", v1:=type);

         IF recInfo.VIF = 16#14 THEN
            // Volume [1e-2  m^3]
            unit := "m^3";
            scale := 0.01;
         END_IF;


         IF type = _MBUS_TYPE_INT32 THEN
            rc := mbusRecordGetInt(handle:=mb, index:=i, value:=d);
            DebugMsg(message:="   Value:     "+floatToStr(v:=FLOAT(d)*scale)+"
"+unit);
         END_IF;

      END_IF;
   END_FOR;
END_FUNCTION;

...
```

4.2.25.15 mbusRecordGetType (Function)

Architecture: NX32L
Device support: LX4,NX-400
Firmware version: 1.94.00

This function returns the type of the record, indicating which functions can be used to read it.

Input:

handle : SYSHANDLE
A handle to the connection

index : INT
The index of the record.

Returns: INT

_MBUS_TYPE_TIME	- Time, can be read with mbusRecordGetLinsec and mbusRecordGetBuffer.
_MBUS_TYPE_TEXT	- Text, can be read with mbusRecordGetString and mbusRecordGetBuffer.
_MBUS_TYPE_FLOAT	- Float, can be read with mbusRecordGetFloat and mbusRecordGetBuffer.
_MBUS_TYPE_INT64	- Large integer, can be read with mbusRecordGetIntLarge and mbusRecordGetBuffer.
_MBUS_TYPE_INT32	- Integer, can be read with mbusRecordGetInt and mbusRecordGetBuffer.
_MBUS_TYPE_BUFFER	- Buffer, can be read with mbusRecordGetBuffer.
0	- Not supported / unknown type.
-1	- Invalid handle
-3	- No data found.

Declaration:

```
FUNCTION mbusRecordGetType : INT;
VAR_INPUT
   handle   : SYSHANDLE;
   index    : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   mb : SYSHANDLE;
   recInfo   : mbusRecordGetInfo;
END_VAR;

FUNCTION DumpRecords;
VAR
   rc    : INT;
   count : INT;
   i     : INT;
   str, unit : STRING;
   scale : FLOAT;
   type  : INT;
   d     : DINT;
END_VAR;
   count := mbusRecordCount(handle:=mb);
   DebugFmt(message:=" Records: \1", v1:=count);


   FOR i := 0 TO count DO
      recInfo(handle:=mb, index:=i);
      IF recInfo.ready THEN
         str := "  Record "+intToStr(v:=i)+", Type: " +
intToStr(v:=recInfo.type);

         scale:=1.0;
         unit:="";

         str := str + ", VIF: "+sintToHex(v:=recInfo.VIF)+", VIFE: "
            + sintToHex(v:=recInfo.VIFE[1])+" "+sintToHex(v:=recInfo.VIFE[2]);
         DebugFmt(message:=str);
         DebugFmt(message:="   Addr: "+recInfo.address);
```

```
            DebugFmt(message:="   Dev: \1, Func: \2, Tar \4, Stor:
"+dintToStr(v:=recInfo.storage),
                                 v1:=recInfo.device, v2:=recInfo.func,
v4:=recInfo.tariff);

            DebugFmt(message:="   Type: "+recInfo.text);

            type := mbusRecordGetType(handle:=mb, index:=i);
            DebugFmt(message:="   Data Type: \1", v1:=type);

            IF recInfo.VIF = 16#14 THEN
               // Volume [1e-2  m^3]
               unit := "m^3";
               scale := 0.01;
            END_IF;


            IF type = _MBUS_TYPE_INT32 THEN
               rc := mbusRecordGetInt(handle:=mb, index:=i, value:=d);
               DebugMsg(message:="   Value:     "+floatToStr(v:=FLOAT(d)*scale)+"
"+unit);
            END_IF;

         END_IF;
      END_FOR;
END_FUNCTION;

...
```

4.2.25.116 mbusRecordGetBuffer (Function)

Architecture: NX32L
Device support: LX4,NX-400
Firmware version: 1.94.00

This function returns the value of a record as a buffer.

Input:

handle : SYSHANDLE
A handle to the connection

index : INT
The index of the record.

data : PTR
Pointer to the buffer to store the value in.

maxsize : INT
The size of the buffer.

Output:

size : INT
The number of bytes stored in the buffer.

Returns: INT

1 - Success,

0 - Not supported.
-1 - Invalid handle
-3 - No data found.
-6 - Buffer is to small. Size will contain the needed size.

Declaration:

```
FUNCTION mbusRecordGetBuffer : INT;
VAR_INPUT
   handle   : SYSHANDLE;
   index    : INT;
   data     : PTR;
   maxsize  : INT;
   size     : ACCESS INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   mb  : SYSHANDLE;
   rc  : INT;
   i   : INT;
   len : INT;
   buf : ARRAY[1..255] OF USINT;
END_VAR;

BEGIN
   ...
   // Get buffer from record
   rc := mbusRecordGetBuffer(handle:=handle, index:=i, data:=ADDR(buf),
maxsize:=sizeof(buf), size:=len);
   ...
END;
END_PROGRAM;
```

4.2.25.117 mbusRecordGetInt (Function)

Architecture: NX32L
Device support: LX4,NX-400
Firmware version: 1.94.00

This function returns the value of a record as a DINT.

Input:

handle : SYSHANDLE
A handle to the connection

index : INT
The index of the record.

Output:

value : DINT
The value of the record.

Returns: INT

1 - Success,
0 - Not supported.
-1 - Invalid handle
-3 - No data found.
-4 - Wrong data type.

Declaration:

```
FUNCTION mbusRecordGetInt : INT;
VAR_INPUT
   handle   : SYSHANDLE;
   index    : INT;
   value    : ACCESS DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   mb : SYSHANDLE;
   recInfo   : mbusRecordGetInfo;
END_VAR;

FUNCTION DumpRecords;
VAR
   rc    : INT;
   count : INT;
   i     : INT;
   str, unit : STRING;
   scale : FLOAT;
   type  : INT;
   d     : DINT;
END_VAR;
   count := mbusRecordCount(handle:=mb);
   DebugFmt(message:=" Records: \1", v1:=count);


   FOR i := 0 TO count DO
      recInfo(handle:=mb, index:=i);
      IF recInfo.ready THEN
         str := "  Record "+intToStr(v:=i)+", Type: " +
intToStr(v:=recInfo.type);

         scale:=1.0;
         unit:="";

         str := str + ", VIF: "+sintToHex(v:=recInfo.VIF)+", VIFE: "
            + sintToHex(v:=recInfo.VIFE[1])+" "+sintToHex(v:=recInfo.VIFE[2]);
         DebugFmt(message:=str);
         DebugFmt(message:="   Addr: "+recInfo.address);
         DebugFmt(message:="   Dev: \1, Func: \2, Tar \4, Stor:
"+dintToStr(v:=recInfo.storage),
                           v1:=recInfo.device, v2:=recInfo.func,
v4:=recInfo.tariff);

         DebugFmt(message:="   Type: "+recInfo.text);

         type := mbusRecordGetType(handle:=mb, index:=i);
         DebugFmt(message:="   Data Type: \1", v1:=type);
```

```
            IF recInfo.VIF = 16#14 THEN
               // Volume [1e-2  m^3]
               unit := "m^3";
               scale := 0.01;
            END_IF;


            IF type = _MBUS_TYPE_INT32 THEN
               rc := mbusRecordGetInt(handle:=mb, index:=i, value:=d);
               DebugMsg(message:="   Value:     "+floatToStr(v:=FLOAT(d)*scale)+"
"+unit);
            END_IF;

         END_IF;
      END_FOR;
END_FUNCTION;

...
```

4.2.25.118 mbusRecordGetIntLarge (Function)

Architecture: NX32L
Device support: LX4,NX-400
Firmware version: 1.94.00

This function returns the value of a record containing an integer of up to 64 bit as two DINTs.

Input:

handle : SYSHANDLE
A handle to the connection

index : INT
The index of the record.

Output:

value_lo : DINT
The lower 32 bits of the record value.

value_hi : DINT
The upper 32 bits of the record value.

Returns: INT

1	- Success,
0	- Not supported.
-1	- Invalid handle
-3	- No data found.
-4	- Wrong data type.

Declaration:

```
FUNCTION mbusRecordGetIntLarge : INT;
VAR_INPUT
   handle    : SYSHANDLE;
   index     : INT;
   value_lo : ACCESS DINT;
```

```
   value_hi : ACCESS DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   mb       : SYSHANDLE;
   rc       : INT;
   i        : INT;
   d_l, d_h : DINT;
END_VAR;

BEGIN
   ...
   // Get 64 bit value from record
   rc := mbusRecordGetIntLarge(handle:=handle, index:=i, value_lo:=d_l,
value_hi:=d_h);
   ...
END;
END_PROGRAM;
```

4.2.25.119 mbusRecordGetFloat (Function)

Architecture: NX32L
Device support: LX4,NX-400
Firmware version: 1.94.00

This function returns the value of a record as a FLOAT.

Input:

handle : SYSHANDLE
A handle to the connection

index : INT
The index of the record.

Output:

value : FLOAT
The value of the record.

Returns: INT

1	- Success,
0	- Not supported.
-1	- Invalid handle
-3	- No data found.
-4	- Wrong data type.

Declaration:

```
FUNCTION mbusRecordGetInt : INT;
VAR_INPUT
   handle    : SYSHANDLE;
   index     : INT;
```

```
   value    : ACCESS FLOAT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   mb  : SYSHANDLE;
   rc  : INT;
   i   : INT;
   val : FLOAT;
END_VAR;

BEGIN
   ...
   // Get value from record
   rc := mbusRecordGetFloat(handle:=handle, index:=i, value:=val);
   ...
END;
END_PROGRAM;
```

4.2.25.20 mbusRecordGetString (Function)

Architecture: NX32L
Device support: LX4,NX-400
Firmware version: 1.94.00

This function returns the value of a record as a STRING.

Input:

handle : SYSHANDLE
A handle to the connection

index : INT
The index of the record.

Output:

value : STRING
The value of the record.

Returns: INT

 1 - Success,
 0 - Not supported.
-1 - Invalid handle
-3 - No data found.
-4 - Wrong data type.

Declaration:

```
FUNCTION mbusRecordGetString : INT;
VAR_INPUT
   handle   : SYSHANDLE;
   index    : INT;
```

```
   value    : ACCESS STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   mb  : SYSHANDLE;
   rc  : INT;
   i   : INT;
   str : STRING;
END_VAR;

BEGIN
   ...
   // Get string from record
   rc := mbusRecordGetString(handle:=handle, index:=i, value:=str);
   ...
END;
END_PROGRAM;
```

4.2.25.21 mbusRecordGetLinsec (Function)

Architecture: NX32L
Device support: LX4,NX-400
Firmware version: 1.94.00

This function returns the value of a record as a linsec value.

Input:

handle : SYSHANDLE
A handle to the connection

index : INT
The index of the record.

Output:

value : DINT
The value of the record.

Returns: INT

1	- Success,
0	- Not supported.
-1	- Invalid handle
-3	- No data found.
-4	- Wrong data type.

Declaration:

```
FUNCTION mbusRecordGetLinsec : INT;
VAR_INPUT
   handle   : SYSHANDLE;
   index    : INT;
```

```
   value    : ACCESS DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   mb     : SYSHANDLE;
   rc     : INT;
   i      : INT;
   linsec : DINT;
END_VAR;

BEGIN
   ...
   // Get time from record
   rc := mbusRecordGetLinsec(handle:=handle, index:=i, value:=linsec);
   ...
END;
END_PROGRAM;
```

4.2.25.22 mbusSend (Function)

Architecture: NX32L
Device support: LX4,NX-400
Firmware version: 1.94.00

This function is used to send raw M-Bus frames.
To send raw wired M-Bus frames to a slave that can not use the normal primary address, mbusSelectSecondary can be used to select the address to make it available as primary address 253.

When sending wireless M-Bus frames, the frame data must start with the Control field followed by the CI field and the rest of the data, skipping large parts of the header.
To change the address used when sending wireless M-Bus frames, use mbusSetSenderAddress.

Input:

handle : SYSHANDLE
A handle to the connection

frame : PTR
The frame to send

size : INT
The size of the frame in bytes.

Returns: **INT**

1	- Success
0	- Not supported
-1	- Invalid handle
-9	- Communication error

-13 - Invalid frame.

Declaration:

```
FUNCTION mbusSend : INT;
VAR_INPUT
   handle   : SYSHANDLE;
   frame    : PTR;
   size     : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
// Uncomment math.inc to add math library support.
//INCLUDE math.inc

//  These are the global variables of the program
VAR
   mb   : SYSHANDLE;
END_VAR;

// Convert hex string into SINT array
FUNCTION ParseString:INT;
VAR_INPUT
   str : STRING;
   dst : PTR;
END_VAR;
VAR
   i   : INT;
   pos : INT;
   arr : ARRAY[0..300] OF SINT;
END_VAR;
   i:=1;
   pos := 0;
   WHILE i < strLen(str := str) DO
      IF strMid(str := str, start := i, length := 1) <> " " THEN
         arr[pos]:=hexToSint(hex:=strMid(str:=str, start:=i, length:=2));
         i := i + 2;
         pos := pos + 1;
      ELSE
         i := i + 1;
      END_IF;
   END_WHILE;
   memcpy(dst:=dst, src:=ADDR(arr), len:=pos);
   ParseString:=pos;
END_FUNCTION;

// Send packet from OMS Annex N.2.3: gas meter with internal radio, security
profile B
FUNCTION SendN23;
VAR
   rc       : INT;
   len      : INT;
   tx_frame : ARRAY [1..300] OF USINT;
END_VAR;
   // Hex string with data from example N.2.3:
   len := ParseString(dst := ADDR(tx_frame), str:=
      // C
      "44"+
      // ELL
      "8c2075"+
      // AFL
      "900f002c25b30a000021924d4f2fb66e01"+
```

```
      // TPL
      "7a7500200710"+
      // Encrypted TPL/APL
      " 9058475f4bc91df878b80a1b0f98b629 "+
      // Encrypted APL
      "024aac727942bfc549233c0140829b93"
   );

   // Set sender address to address from DLL
   rc := mbusSetSenderAddress(handle:=mb, sec_addr:="1234567893153303");
   DebugFmt(message:="mbusSetSenderAddress(): \1", v1:=rc);

   rc := mbusSend(handle:=mb, frame := ADDR(tx_frame), size := len);
   DebugFmt(message:="mbusSend(): \1", v1:=rc);

END_FUNCTION;


PROGRAM sender;
// These are the local variables of the program block
VAR
   rc : INT;
END_VAR;
// The next code will only be executed once after the program starts
   rc := mbusOpen(type:=2, handle:=mb);
   DebugFmt(message:="mbusOpen(): \1", v1:=rc);

BEGIN
// Code from this point until END will be executed repeatedly

   SendN23();

   Sleep(delay:=10000);

END;
END_PROGRAM;
```

4.2.25.23 mbusReceive (Function)

Architecture: NX32L
Device support: LX4,NX-400
Firmware version: 1.94.00

This function is used to receive raw M-Bus frames.
For M-Bus, the frame must be requested using mbusDataRequest or mbusSend, and this function will block for a moment while it tries to read the response.
Wireless M-Bus devices may automatically send the frames and the timeout parameter can be used to determine how long this function should wait for a response.
The filter parameter can be used to only receive frames from a specific wireless M-Bus device.

Input:

handle : SYSHANDLE
A handle to the connection

frame : PTR
The buffer to store the frame in.

maxsize : INT
The size of the buffer in bytes.

filter : STRING
For Wireless M-Bus only: Specifies the address or address filter to receive frames from.

timeout : INT *Default 0*
Number of seconds to wait for a Wireless M-Bus frame. Use 0 seconds to return immediately if there is no data in the buffer.

Output:

size : INT
The number of bytes received.

Returns: INT

1	- Success
0	- Not supported
-1	- Invalid handle
-2	- Frame is too small.
-7	- Invalid filter.
-9	- Communication error
-10	- Timeout before data was received.
-14	- Invalid timeout value.

Declaration:

```
FUNCTION mbusReceive : INT;
VAR_INPUT
   handle   : SYSHANDLE;
   frame    : PTR;
   maxsize  : INT;
   filter   : STRING;
   timeout  : INT:=0;
   size     : ACCESS INT;
END_VAR;
```

Example:

```
// Dump all received frames from ELS meters, to e.g. help with debugging.
INCLUDE rtcu.inc
// Uncomment math.inc to add math library support.
//INCLUDE math.inc

//  These are the global variables of the program
VAR
   mb : SYSHANDLE;

END_VAR;

PROGRAM wmbus_raw_rec;
// These are the local variables of the program block
VAR
   rc       : INT;
   len      : INT;
   rx_frame : ARRAY [1..300] OF USINT;
   str      : STRING;
   i        : INT;
END_VAR;
// The next code will only be executed once after the program starts

   rc := mbusOpen(type:=2, handle:=mb, mode:=_MBUS_MODE_T1_C);
   DebugFmt(message:="mbusOpen(): \1", v1:=rc);
```

```
   // Receive from all devices
   rc := mbusFilterEnable(handle:=mb, enable:=FALSE);
   DebugFmt(message:="mbusFilterEnable(): \1", v1:=rc);

BEGIN
   // Code from this point until END will be executed repeatedly

   // Filter will only receive from ELS meters (16#1593)
   rc := mbusReceive(handle:=mb, filter:="FFFFFFFF9315FFFF", frame :=
ADDR(rx_frame), maxsize:=300, size := len, timeout:=30);
   DebugFmt(message:="mbusReceive(): \1, len=\2", v1:=rc, v2:=len);
   IF rc = 1 THEN
      str :="";
      FOR i := 1 TO len DO
         str := str + sintToHex(v:=rx_frame[i])+" ";
         // Break up string after 32 bytes
         IF i MOD 32 = 0 THEN
            DebugMsg(message:=str);
            str := "";
         END_IF;
      END_FOR;
      DebugMsg(message:=str);
   END_IF;
   Sleep(delay:=1000);
END;

END_PROGRAM;
```

4.2.25.24 mbusSelectSecondary (Function)

Architecture: NX32L
Device support: LX4
Firmware version: 1.94.00

This function is used to select a slave based on the secondary address and makes it available using primary address 253.
This can be used in combination with mbusSend to send raw M-Bus frames when the normal primary address can not be used.

Input:

handle : SYSHANDLE
A handle to the connection

sec_addr : STRING
The secondary address for the slave. See mbusScan for details about the format of the secondary address.

Returns: INT

1	- Success
0	- Not supported
-1	- Invalid handle
-5	- Invalid interface type.
-6	- Invalid address
-9	- Communication error

Declaration:

```
FUNCTION mbusSelectSecondary : INT;
VAR_INPUT
```

```
   handle    : SYSHANDLE;
   sec_addr  : STRING;
END_VAR;
```

Example:

```
   ...
   // Make the device with secondary address 1234567812345678 available as
primary address 253.
   rc := mbusSelectSecondary(handle:=mb, sec_addr := "1234567812345678");
   DebugFmt(message:="SelectSecondary(): \1", v1:=rc);
   ...
```

4.2.25.25 mbusSetSenderAddress (Function)

Architecture: NX32L
Device support: LX4,NX-400
Firmware version: 1.94.00

This function is used to change the address used when sending Wireless M-Bus frames with mbusSend.

Input:

handle : SYSHANDLE
A handle to the connection

sec_addr : STRING
The address to use when sending.

Returns: INT

1	- Success
0	- Not supported
-1	- Invalid handle
-5	- Invalid interface type.
-6	- Invalid address
-9	- Communication error

Declaration:

```
FUNCTION mbusSetSenderAddress : INT;
VAR_INPUT
   handle    : SYSHANDLE;
   sec_addr  : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
// Uncomment math.inc to add math library support.
//INCLUDE math.inc

//  These are the global variables of the program
VAR
   mb   : SYSHANDLE;
END_VAR;
```

```
// Convert hex string into SINT array
FUNCTION ParseString:INT;
VAR_INPUT
   str : STRING;
   dst : PTR;
END_VAR;
VAR
   i   : INT;
   pos : INT;
   arr : ARRAY[0..300] OF SINT;
END_VAR;
   i:=1;
   pos := 0;
   WHILE i < strLen(str := str) DO
      IF strMid(str := str, start := i, length := 1) <> " " THEN
         arr[pos]:=hexToSint(hex:=strMid(str:=str, start:=i, length:=2));
         i := i + 2;
         pos := pos + 1;
      ELSE
         i := i + 1;
      END_IF;
   END_WHILE;
   memcpy(dst:=dst, src:=ADDR(arr), len:=pos);
   ParseString:=pos;
END_FUNCTION;

// Send packet from OMS Annex N.2.3: gas meter with internal radio, security
profile B
FUNCTION SendN23;
VAR
   rc       : INT;
   len      : INT;
   tx_frame : ARRAY [1..300] OF USINT;
END_VAR;
   // Hex string with data from example N.2.3:
   len := ParseString(dst := ADDR(tx_frame), str:=
      // C
      "44"+
      // ELL
      "8c2075"+
      // AFL
      "900f002c25b30a000021924d4f2fb66e01"+
      // TPL
      "7a7500200710"+
      // Encrypted TPL/APL
      " 9058475f4bc91df878b80a1b0f98b629 "+
      // Encrypted APL
      "024aac727942bfc549233c0140829b93"
   );

   // Set sender address to address from DLL
   rc := mbusSetSenderAddress(handle:=mb, sec_addr:="1234567893153303");
   DebugFmt(message:="mbusSetSenderAddress(): \1", v1:=rc);

   rc := mbusSend(handle:=mb, frame := ADDR(tx_frame), size := len);
   DebugFmt(message:="mbusSend(): \1", v1:=rc);

END_FUNCTION;


PROGRAM sender;
// These are the local variables of the program block
VAR
```

```
   rc : INT;
END_VAR;
// The next code will only be executed once after the program starts
   rc := mbusOpen(type:=2, handle:=mb);
   DebugFmt(message:="mbusOpen(): \1", v1:=rc);

BEGIN
// Code from this point until END will be executed repeatedly

   SendN23();

   Sleep(delay:=10000);

END;
END_PROGRAM;
```

4.2.25.26 mbusSlaveRegister (Function)

Architecture: NX32L
Device support: LX4,NX-400
Firmware version: 1.94.00

This function registers a slave device.
Registered device can be decrypted if the correct key is provided.
Only data from registered devices is received if the receive filter is enabled with mbusFilterEnable.
The registration is persistent and must be removed with mbusSlaveUnregister.

Input:

handle : SYSHANDLE
A handle to the connection

index : USINT (1..64)
The index of the registration.

sec_addr : STRING
The address to the slave to register. See mbusScan for a description of the address format.

key : PTR
Pointer to a 16-byte array containing the key needed to decrypt the data.

Returns: INT

1	- Success.
0	- Not supported.
-1	- Invalid handle
-2	- Index out of range.
-5	- Invalid interface type.
-6	- Invalid address.
-9	- Communication error

Declaration:

```
FUNCTION mbusSlaveRegister : INT;
VAR_INPUT
   handle   : SYSHANDLE;
   key      : PTR;
   sec_addr : STRING;
   index    : USINT;
END_VAR;
```

Example:

```
FUNCTION RegisterDevices;
VAR
   rc  : INT;
   key : ARRAY[1..16] OF USINT;
END_VAR;

   // wM-Bus Meter with integrated radio and Security profile B from OMS
specification Annex N.2.3.
   key[1] := 16#00;
   key[2] := 16#01;
   key[3] := 16#02;
   key[4] := 16#03;
   key[5] := 16#04;
   key[6] := 16#05;
   key[7] := 16#06;
   key[8] := 16#07;
   key[9] := 16#08;
   key[10] := 16#09;
   key[11] := 16#0a;
   key[12] := 16#0b;
   key[13] := 16#0c;
   key[14] := 16#0d;
   key[15] := 16#0e;
   key[16] := 16#0f;

   rc := mbusSlaveRegister(handle:=mb, sec_addr:="1234567893153303", index:=1,
key:=ADDR(key));
   DebugFmt(message:="mbusSlaveRegister(): \1", v1:=rc);
END_FUNCTION;
```

4.2.25.27 mbusSlaveUnregister (Function)

Architecture: NX32L
Device support: LX4,NX-400
Firmware version: 1.94.00

This function removes a slave device registration created with mbusSlaveRegister.

Input:

handle : SYSHANDLE
A handle to the connection

index : USINT (1..64)
The index of the registration to remove.

Returns: INT

1 - Success.
0 - Not supported.
-1 - Invalid handle
-2 - Index out of range.
-5 - Invalid interface type.
-9 - Communication error

Declaration:

```
FUNCTION mbusSlaveUnregister : INT;
VAR_INPUT
   handle    : SYSHANDLE;
   index     : USINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   mb      : SYSHANDLE;
   rc      : INT;
END_VAR;

BEGIN
   ...
   // Remove registration for slave index 1
   rc := mbusSlaveUnregister(handle:=mb, index:=1);
   DebugFmt(message:="mbusSlaveUnRegister(): \1", v1:=rc);
   ...
END;
END_PROGRAM;
```

4.2.25.28 mbusFilterEnable (Function)

Architecture: NX32L
Device support: LX4,NX-400
Firmware version: 1.94.00

This function controls which message to receive from a Wireless M-Bus interface.
When enabled, only messages from slaves registered with mbusSlaveRegister are received.

Input:

handle : SYSHANDLE
A handle to the connection

enable : BOOL *Default TRUE*
Set to true to only receive from registered slaves, set to false to receive all messages.

Returns: INT

1	- Success.
0	- Not supported.
-1	- Invalid handle
-5	- Invalid interface type.
-9	- Communication error

Declaration:

```
FUNCTION mbusFilterEnable : INT;
VAR_INPUT
   handle    : SYSHANDLE;
   enable    : BOOL := TRUE;
END_VAR;
```

Example:

```
...
// Only receive data from registered devices
rc := mbusFilterEnable(handle:=mb, enable:=TRUE);
DebugFmt(message:="mbusFilterEnable(): \1", v1:=rc);
...
```

4.2.26 mdt: Mobile Data Terminal

4.2.26.1 mdt: Mobile Data Terminal

The Mobile Data Terminal (MDT) is an accessory compromised of an LCD display with keys for advanced user interaction with the RTCU device.
The MDT is connected to the RTCU device by using the RS232 port on the device.
For more information on the MDT see the data sheet and technical manual available from your supplier.

The following functions are used to access to the MDT:

• mdtOpen	Opens the MDT interface.
• mdtPower	Controls power to the MDT.
• mdtStandby	Controls MDT standby.
• mdtWrite	Writes text at the current position.
• mdtGotoXY	Sets the current position.
• mdtCurrentX	Returns the X-coordinate of the current position.
• mdtCurrentY	Returns the Y-coordinate of the current position.
• mdtScrollDown	Moves the text in the display one line down.
• mdtScrollUp	Moves the text in the display one line up.
• mdtGetKey	Waits for a key press event.
• mdtClear	Clears the display.
• mdtClearLine	Clears a line in the display.
• mdtCursor	Shows or hides the cursor.
• mdtBacklight	Sets the intensity of the backlight.
• mdtBeep	Makes the MDT emit a short beep.
• mdtContrast	Sets the contrast of the display.
• mdtDefineChar	Defines a custom character in the MDT font.
• mdtProfile	Retrieves MDT profile info.

4.2.26.2 mdtOpen (Function)

```
Architecture:       X32 / NX32 / NX32L
Device support:     All
Firmware version:   1.02 / 1.00.00
```

This function starts up the connection to the MDT by using the serial port specified.
It must be called before any communication with the MDT can occur. Note that the MDT does not have to actually be connected when this function is called.

When the MDT is opened, there will be no need for further intervention from the VPL program to keep the session up.

Note:
Only port=1 will support the mdtPower() functionality and using port=0 will not work with the standard interface cable.
For information on how to use port=0, please contact your supplier.

Note:
If used with NMP, navOpen() must be called before mdtOpen() by using the same port.
mdtPower() will work regardless of the port.

Input:
port : SINT (0/1) *default 1*
This selects which serial port to use (see serial port enumeration).

Returns: INT

1 - Invalid port.
0 - Success.

Declaration:

```
FUNCTION mdtOpen : INT;
VAR_INPUT
   port : SINT := 1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc : INT;
END_VAR;

BEGIN
   ...
   // Open MDT
   rc := mdtOpen(port := 1);
   ...
END;
END_PROGRAM;
```

4.2.26.3 mdtPower (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.02 / 1.00.00

This function controls the power of the MDT. By using this function, the VPL program can switch ON and OFF the MDT.
The mdtPower() function must be called before the MDT can be used

By switching the power OFF to the MDT, it is not possible to receive a "PowerUP" key press message.
Please also see the mdtStandby() function.

Note:
Only when the mdtOpen has been called with port=1 will the mdtPower() functionality, as described, be available unless the MDT is part of an NMP device.

Input:

power : BOOL
TRUE: Turns the MDT on.
FALSE: Turns the MDT off.

Returns: INT

0 - Success.
-1 - MDT not present or communication error.
-2 - MDT not open.

Declaration:

```
FUNCTION mdtPower : INT;
VAR_INPUT
   power : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc : INT;
END_VAR;

BEGIN
   ...
   // Turn MDT on
   rc := mdtPower(power := ON);
   ...
END;
END_PROGRAM;
```

4.2.26.4 mdtStandby (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.02 / 1.00.00

This function controls the power of the MDT. By using this function, the VPL program can set the MDT on standby or wake it from standby - for example in case of the "PowerUP" key press event returned from mdtGetKey().
Also see mdtPower().

Input:

enable : BOOL
TRUE: Sets the MDT on standby.
FALSE: Wakes the MDT from standby.

Returns: **INT**

0 - Success.
-1 - MDT not present or communication error.
-2 - MDT not open.

Declaration:

```
FUNCTION mdtStandby : INT;
VAR_INPUT
   enable : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc : INT;
END_VAR;
```

```
BEGIN
   ...
   // Set MDT on standby
   rc := mdtStandby(enable := ON);
   ...
END;
END_PROGRAM;
```

4.2.26.5 mdtWrite (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.02 / 1.00.00

This function writes a text message on the display at the current write position.
If a text message tries to print past the display limits, the remaining text is ignored and the current write position remains at the end of the line.

It is possible to add certain attributes to the text printed in the display by using the following control sequences:

\H	Starts highlighted text.
\h	Stops highlighted text.
\U	Starts underlined text.
\u	Stops underlined text.
\BL1	Starts blink mode 1 (the text alternates between visible and hidden).
\bl1	Stops blink mode 1.
\BL2	Starts blink mode 2 (the text alternates between normal and highlighted).
\bl2	Stops blink mode 2.

Once the text attribute is started, all text printed will be affected until it is stopped.

Note:
If the string passed to mdtWrite contains the "$" character, it must be duplicated ("$$") to avoid interference with the low-level communication protocol used between the RTCU and MDT.
Please further observe the restrictions when using the "$" characters in strings.
Failing to observe these precautions may result in the write operation failing.

Input:

message : STRING
Text string to print.

Returns: INT

0 - Success.
-1 - MDT not present or communication error.
-2 - MDT not open.

Declaration:

```
FUNCTION mdtWrite : INT;
VAR_INPUT
   message : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
```

```
PROGRAM test;
VAR
   rc : INT;
END_VAR;

BEGIN
   ...
   // Write test
   mdtGotoXY(x := 5, y := 2);
   mdtWrite(message := "Highlight \Htest\h text");
   mdtGotoXY(x := 5, y := 4);
   mdtWrite(message := "Underline \Utest\u text");
   mdtGotoXY(x := 5, y := 6);
   mdtWrite(message := "Blink 1   \BL1test\bl1 text");
   mdtGotoXY(x := 5, y := 8);
   rc := mdtWrite(message := "Blink 2   \BL2test\bl2 text");
   ...
END;
END_PROGRAM;
```

4.2.26.6 mdtGotoXY (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.02 / 1.00.00

Sets the current write position. This is the position where the next write to the display will take place.
To get the actual range of (x,y) for the connected MDT, it is recommended to call mdtProfile().

Input:

x : INT (1..28)
X position (column) - 1 is leftmost.

y : INT (1..11)
Y position (row) - 1 is topmost.

Returns: INT

1	- Position outside display.
0	- Success.
-1	- MDT not present or communication error.
-2	- MDT not open.

Declaration:

```
FUNCTION mdtGotoXY : INT;
VAR_INPUT
   x : INT;
   y : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc : INT;
END_VAR;

BEGIN
```

```
    ...
    // Move current position
    rc := mdtGotoXY(x := 15, y := 4);
    ...
END;
END_PROGRAM;
```

4.2.26.7 mdtCurrentX (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.02 / 1.00.00

This function will return the x-coordinate of the current position.

Input:
None.

Returns: **INT**
Current x-coordinate
-1 - MDT not present or communication error.
-2 - MDT not open.

Declaration:

```
FUNCTION mdtCurrentX : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
    x : INT;
    y : INT;
END_VAR;

BEGIN
    ...
    // Get curent position
    x := mdtCurrentX();
    y := mdtCurrentY();
    ...
END;
END_PROGRAM;
```

4.2.26.8 mdtCurrentY (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.02 / 1.00.00

This function will return the y-coordinate of the current position.

Input:
None.

Returns: **INT**
Current y-coordinate.
-1 - MDT not present or communication error.

-2 - MDT not open.

Declaration:

```
FUNCTION mdtCurrentY : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   x : INT;
   y : INT;
END_VAR;

BEGIN
   ...
   // Get curent position
   x := mdtCurrentX();
   y := mdtCurrentY();
   ...
END;
END_PROGRAM;
```

4.2.26.9 mdtScrollDown (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.02 / 1.00.00

This function moves the contents of the display one line down. The line that moves out of the display is deleted and an empty line is added at the top.
After this function has been called, the current position is at the upper left corner (1, 1).

Input:

None.

Returns: INT

0 - Success.
-1 - MDT not present or communication error.
-2 - MDT not open.

Declaration:

```
FUNCTION mdtScrollDown : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc : INT;
END_VAR;

BEGIN
   ...
   // Scroll display
   rc := mdtScrollDown();
   ...
```

```
END;
END_PROGRAM;
```

4.2.26.10 mdtScrollUp (Function)

```
Architecture:       X32 / NX32 / NX32L
Device support:     All
Firmware version:   1.02 / 1.00.00
```

This function moves the contents of the display one line up. The line that moves out of the display is deleted and an empty line is added at the bottom.
After this function has been called, the current position is at the lower left corner (1, 8).

Input:

None.

Returns: INT

0 - Success.
-1 - MDT not present or communication error.
-2 - MDT not open.

Declaration:

```
FUNCTION mdtScrollUp : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc : INT;
END_VAR;

BEGIN
   ...
   // Scroll display
   rc := mdtScrollUp();
   ...
END;
END_PROGRAM;
```

4.2.26.11 mdtGetKey (Function)

```
Architecture:       X32 / NX32 / NX32L
Device support:     All
Firmware version:   1.02 / 1.00.00
```

This waits for a key press from the MDT. The function waits for a maximum number of milliseconds, and if no key press has occurred within the timeout period, the function returns 0 which indicates the timeout. If a key press has been detected before the timeout period, a number identifying the actual key press is returned.
A small buffer is present in the system so that all key presses are received by the application.

Input:

timeout : INT (0..32767) ***Default 3000***
Timeout period in milliseconds to wait.

Returns: INT

127	- PowerUP key pressed (only generated when the MDT is in standby mode and the power key is pressed).
126	- Power key pressed.
125	- Back key pressed.
124	- Enter key pressed.
123	- Right key pressed.
122	- Left key pressed.
121	- Down key pressed.
120	- Up key pressed.
105	- F6 key pressed.
104	- F5 key pressed.
103	- F4 key pressed.
102	- F3 key pressed.
101	- F2 key pressed.
100	- F1 key pressed.
21	- # key pressed (MDT-200 and NMP only).
20	- * key pressed (MDT-200 and NMP only).
19	- "9" key pressed (MDT-200 and NMP only).
18	- "8" key pressed (MDT-200 and NMP only).
17	- "7" key pressed (MDT-200 and NMP only).
16	- "6" key pressed (MDT-200 and NMP only).
15	- "5" key pressed (MDT-200 and NMP only).
14	- "4" key pressed (MDT-200 and NMP only).
13	- "3" key pressed (MDT-200 and NMP only).
12	- "2" key pressed (MDT-200 and NMP only).
11	- "1" key pressed (MDT-200 and NMP only).
10	- "0" key pressed (MDT-200 and NMP only).
0	- Timeout.
-2	- MDT not open.

Declaration:

```
FUNCTION mdtGetKey : INT;
VAR_INPUT
   timeout : INT := 3000;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   key : INT;
END_VAR;

BEGIN
   ...
   // Wait for key
   key := mdtGetKey(timeout := 10000);
   ...
END;
END_PROGRAM;
```

4.2.26.12 mdtClear (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.02 / 1.00.00

Clears the contents in the display. After this call, the current write position will be set to upper left corner (1,1).

Input:
None.

Returns: INT
0 - Success.
-1 - MDT not present or communication error.
-2 - MDT not open.

Declaration:
```
FUNCTION mdtClear : INT;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Clear display
   mdtClear();
   ...
END;
END_PROGRAM;
```

4.2.26.13 mdtClearLine (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.02 / 1.00.00

Clears the contents of the line with the current write position in the display. After this call, the current write position will be set to left of the line.

Input:
None.

Returns: INT
0 - Success.
-1 - MDT not present or communication error.
-2 - MDT not open.

Declaration:
```
FUNCTION mdtClearLine : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Clear line
   mdtClearLine();
   ...
END;
END_PROGRAM;
```

4.2.26.14 mdtCursor (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.02 / 1.00.00

This function is used to enable or disable the cursor.

Input:

enable : BOOL
TRUE: - Shows cursor.
FALSE: - Hides cursor.

Returns: INT

0 - Success.
-1 - MDT not present or communication error.
-2 - MDT not open.

Declaration:

```
FUNCTION mdtCursor : INT;
VAR_INPUT
   enable : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc : INT;
END_VAR;

BEGIN
   ...
   // Show cursor
   rc := mdtCursor(enable := ON);
   ...
END;
END_PROGRAM;
```

4.2.26.15 mdtBacklight (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.02 / 1.00.00

Controls the intensity of the backlight in the display.

Input:

intensity : SINT (0..10)
The intensity of the backlight (0 = off, 10=highest intensity).

Returns: INT

1 - Intensity out of bounds.
0 - Success.
-1 - MDT not present or communication error.
-2 - MDT not open.

Declaration:

```
FUNCTION mdtBacklight : INT;
VAR_INPUT
   intensity : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Set the backlight intensity
   mdtBacklight(intensity := 10);
   ...
END;
END_PROGRAM;
```

4.2.26.16 mdtBeep (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.02 / 1.00.00

This function makes the MDT emit a short beep.

Input:

None.

Returns: INT

0 - Success.
-1 - MDT not present or communication error.
-2 - MDT not open.

Declaration:

```
FUNCTION mdtBeep : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Sound a beep
   mdtBeep();
   ...
END;
END_PROGRAM;
```

4.2.26.17 mdtContrast (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.03 / 1.00.00

This controls the contrast in the display.

Input:

intensity : SINT (0..10)
The intensity of the backlight (0 = lowest contrast, 10=highest contrast).

Returns: INT

1	- Intensity out of bounds.
0	- Success.
-1	- MDT not present or communication error.
-2	- MDT not open.

Declaration:

```
FUNCTION mdtContrast : INT;
VAR_INPUT
   intensity : SINT;
END_VAR;
```

Example:

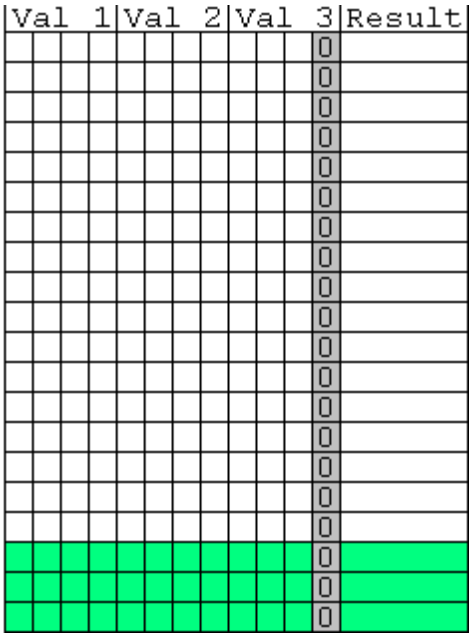
```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Set the contrast
   mdtContrast(intensity := 10);
   ...
END;
END_PROGRAM;
```

4.2.26.18 mdtDefineChar (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.07 / 1.00.00

This function is used to define a character in the MDT font. **This feature is not available on the MDT-100**.

The font in the MDT-200 uses characters 11 pixels wide by 20 pixels high.
To help visualize and calculate the character map, use a table with 12 * 20 cells with each representing a pixel.



(Figure 1. Empty character pixel map)

The 12th column (with gray background in Figure 1) is not used in the character and is only included to ease the calculations. This must be filled with 0's (zeros).
The 3 last rows (with green background in Figure 1) is below the writing line and should only be used if characters similar to "p", "g", or "q" are defined.

Draw the character in the table using 1's (ones, shows pixel) and 0's (zeros, hides pixel) as shown in Figure 2.

Val 1	Val 2	Val 3	Result
0 0 0 0 0 0 0 0 0 0 0	0	0	000
0 0 0 0 0 0 0 0 0 0 0	0	0	000
0 0 0 0 0 0 0 0 0 0 0	0	0	000
0 0 1 1 1 1 1 1 0 0 0	0	0	3F0
0 0 1 1 1 1 1 1 0 0 0	0	0	3F0
0 1 1 0 0 0 0 1 1 0 0	0	0	618
1 1 0 0 1 1 0 0 1 1 0	0	0	CCC
1 1 0 1 0 0 1 0 1 1 0	0	0	D2C
1 1 0 0 0 0 1 0 1 1 0	0	0	C2C
1 1 0 1 0 0 1 0 1 1 0	0	0	CCC
1 1 0 0 1 1 0 0 1 1 0	0	0	D2C
0 1 1 0 0 0 0 1 1 0 0	0	0	CCC
0 0 1 1 1 1 1 1 0 0 0	0	0	3F0
0 0 1 1 1 1 1 1 0 0 0	0	0	3F0
0 0 0 0 0 0 0 0 0 0 0	0	0	000
0 0 0 0 0 0 0 0 0 0 0	0	0	000
0 0 0 0 0 0 0 0 0 0 0	0	0	000
0 0 0 0 0 0 0 0 0 0 0	0	0	000
0 0 0 0 0 0 0 0 0 0 0	0	0	000
0 0 0 0 0 0 0 0 0 0 0	0	0	000

*(Figure 2. Sample character: **copyleft**)*

Once the table is filled, calculate the 3 hex digits for each line.
The map string in the function contains the hex digits from the top-left and down.
For the sample character in Figure 2., the map string would look like this:
"`0000000003F03F0618CCCD2CC2CD2CCCC6183F03F0000000000000000000`".

Note: only uppercase characters can be used and no spacing characters can be included.

Input:

index : INT (0..255)
The index of the character to define.

map : STRING
The calculated map of the character.

Returns: INT

0 - Success.
-1 - MDT not present or communication error.
-2 - MDT not open.

Declaration:

```
FUNCTION mdtDefineChar : INT;
VAR_INPUT
   index : INT;
   map   : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

mdtDefineChar
(index:=33
,map:="0000000003F03F0618CCCD2CC2CD2CCCC6183F03F0000000000000000000");
```

```
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.26.19 mdtProfile (Functionblock)

```
Architecture:       X32 / NX32 / NX32L
Device support:     All
Firmware version:   1.07 / 1.00.00
```

mdtProfile returns the profile of the MDT connected.

Input:

None.

Output:

type : INT

0 - No MDT connected.
1 - MDT-100 connected.
2 - MDT-200 connected.
3 - NMP connected.

version : INT
Version number of the firmware in the MDT. Version will be scaled by 100 - version 3.00 will be returned as 300.

pos_max_x : INT
The maximum x position.

pos_max_y : INT
The maximum y position.

Declaration:

```
FUNCTION_BLOCK mdtProfile;
VAR_OUTPUT
   type      : INT;
   version   : INT;
   pos_max_x : INT;
   pos_max_y : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   mdtInfo : mdtProfile;
END_VAR;

PROGRAM MDTExample;

...

BEGIN
   ...
   mdtInfo();
   IF mdtInfo.type > 0 THEN
```

```
        DebugMsg(message:="MDT connected!");
        DebugFmt(message:="  Type=\1",v1:=mdtInfo.type);
        DebugFmt(message:="  Ver= \1",v1:=mdtInfo.version);
        DebugFmt(message:="  MaxX=\1",v1:=mdtInfo.pos_max_x);
        DebugFmt(message:="  MaxY=\1",v1:=mdtInfo.pos_max_y);
    END_IF;
    ...
END;
END_PROGRAM;
```

4.2.27 misc: Miscellaneous functions

4.2.27.1 Misc: Miscellaneous Functions

The miscellaneous functions are functions that cannot be categorized with the other groups of Platform Support Functions:

• Sleep	Makes the device sleep for a number of milliseconds.
• DeepSleep	Makes the device sleep for a number of milliseconds by entering power saving sleep mode.
• PowerDown	Powers the device down for a certain number of seconds.
• HostConnected	Returns the status for connection to the RTCU IDE.
• memioWrite	Writes directly to memory I/O system.
• memioRead	Reads directly from memory I/O system.
• memioWriteX	Writes directly to memory I/O system.
• memioReadX	Reads directly from memory I/O system.

4.2.27.2 Sleep (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

Sleep is a simple function that will take the specified number of milliseconds to complete before it returns control to the caller. This is a quick way of introducing delays in a program. However, a cleaner and better way is often to use one of the different timers available, such as the TON timer.

Input:

delay : INT (0..32767)
Number of milliseconds to wait before returning.

Returns:

None.

Declaration:

```
FUNCTION Sleep;
VAR_INPUT
   delay : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Wait 1000 mSec (1 second)
   Sleep(delay := 1000);
   ...
END;

END_PROGRAM;
```

4.2.27.3 DeepSleep (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.02 / 1.00.00

DeepSleep will stop the processor of the RTCU device for a number of milliseconds - thereby lowering the power consumption considerably.

This function is equivalent to pmDeepSleep().

Note:
During deep sleep mode the timers (TP, TON, TOF, etc.) will not be updated and the time spent in deep sleep mode will not be counted.
The real-time clock functions (clockNow, clockGet, etc.) will be updated correctly.

The DeepSleep function will degrade to a Sleep() operation when the RTCU IDE (or any other RACP1-compatible host) is connected to the device. This is to ensure stable communication without interruptions.

Input:

delay : INT (0..32767)
Number of milliseconds to sleep.
The actual deep sleep period will be adjusted into steps of 250 ms.

Returns:

None.

Declaration:

```
FUNCTION DeepSleep;
VAR_INPUT
   delay : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Go in low-power mode for 5000 mSecs
   DeepSleep(delay := 5000);
   ...
END;

END_PROGRAM;
```

4.2.27.4 PowerDown (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

PowerDown will switch the power completely off to the RTCU and subsequently switch it on again after a number of seconds. When the device starts again, the program starts from the beginning - just as if the power had been reconnected.

On devices that support an **IGNITION** input, the device will also power up if this input goes high while the device is in power down mode. If the **IGNITION** input is high when the PowerDown() function is called, an error code will be returned.

Input:

seconds : DINT (0..2147483648)
Number of seconds to sleep.
-1 will keep the device powered down forever (until the power is recycled or the ignition input is triggered).

Returns: INT

If the power-down succeeds, the device will power down for the specified time.
If the function does not succeed, it will return with:

1 **-** Power-down not possible because ignition is active.
2 **-** Power-down not successful - probably because the device has been forced on.

Declaration:

```
FUNCTION PowerDown : INT;
VAR_INPUT
   seconds : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Switch power off, and turn it on again after 60 seconds
   PowerDown(seconds := 60);
   ...
END;

END_PROGRAM;
```

4.2.27.5 HostConnected (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This function checks whether the RTCU module is connected to a PC with the RTCU IDE program running (or any other RACP1-capable program).
The HostConnected function will not indicate when the RTCU IDE (or any other RACP2-capable program) is connected to the device via an IP-network.

Input:

None.

Returns: _BOOL

TRUE: Connected to a PC with the RTCU IDE running.

FALSE: Not connected.

Declaration:

```
FUNCTION HostConnected : BOOL;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Check for connection to RTCU IDE
   IF HostConnected() THEN
      // RTCU is connected to the RTCU IDE
      ...
   ELSE
      // RTCU is NOT connected to a PC with RTCU IDE running
      ...
   END_IF;

END;

END_PROGRAM;
```

4.2.27.6 memioWrite (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

---

memioWrite gives direct access to 4096 memory I/O locations. The memory I/O is normally only accessed indirectly by assigning variables in the programs VAR_INPUT/VAR_OUTPUT section to different memory locations by using the Job configuration dialog.

Input:

index : INT (1..4096)
Index number on the memory IO system to write the value in.

value : DINT
Value that is to be written to the specified memory location.

Returns:

None.

Declaration:

```
FUNCTION memioWrite;
VAR_INPUT
   index  : INT; // index to write to (1..4096)
   value  : DINT;// Value to write
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
```

```
// Write the value 4711 to memory location 1
memioWrite(index:=1, value:=4711);

// And read the value back again...
debugFmt(message:="Location 1 is \4", v4:=memioRead(index:=1));

BEGIN
   ...
END;

END_PROGRAM;
```

4.2.27.7 memioRead (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

---

memioRead gives direct access to 4096 memory I/O locations. The memory I/O is normally only accessed indirectly by assigning variables in the programs VAR_INPUT/VAR_OUTPUT section to different memory locations by using the Job configuration dialog.

Input:

index : INT (1..4096)
Index number on the memory IO system to read from.

Returns: DINT

Value that is read from the specified memory location.

Declaration:

```
FUNCTION memioRead : DINT;
VAR_INPUT
   index : INT; // index to read from (1..4096)
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Write the value 4711 to memory location 1
memioWrite(index:=1, value:=4711);

// And read the value back again...
debugFmt(message:="Location 1 is \4", v4:=memioRead(index:=1));

BEGIN
   ...
END;

END_PROGRAM;
```

4.2.27.8 memioWriteX (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.20 / 1.00.00

---

memioWriteX works similar to memioWrite but allows access to multiple elements in one highly efficient operation.
The use of memioWriteX is recommended in cases where a range of memory I/O locations must be written sequentially.

Also see memioReadX.

Input:

index : INT (1..4096)
Memory I/O system Index number where the write operation will start.

mem : PTR
Address to read the data that will be written into the memory I/O area.

len : INT
Number of 32-bit elements to be written into the memory I/O area.

Returns: INT

0	Operation successful.
1	Invalid parameters.
2	Index range was out of bounds.

Declaration:

```
FUNCTION memioWriteX : INT;
VAR_INPUT
   index  : INT;  // index to write to (1..4096).
   mem    : PTR;  // address to read from.
   len    : DINT; // number of elements to write.
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   buffer : ARRAY[1..10] OF DINT;
END_VAR;

...

// Write the content of the 'buffer' directly into I/O space starting at
location 120:
memioWriteX(index:=120, mem:=ADDR(buffer), len:=10);

...

END_PROGRAM;
```

4.2.27.9 memioReadX (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.20 / 1.00.00

---

memioReadX works similar to memioRead but allows access to multiple elements in one highly efficient operation.
The use of memioReadX is recommended in cases where a range of memory I/O locations must be read sequentially.

Also see memioWriteX.

Input:

index : INT (1..4096)
Memory I/O system Index number where the read operation will start.

mem : PTR
Address to write the data that will be read from the memory I/O area.

len : INT
Number of 32-bit elements to be read from the memory I/O area.

Returns: INT

0 Operation successful.
1 Invalid parameters.
2 Index range was out of bounds.

Declaration:

```
FUNCTION memioReadX : INT;
VAR_INPUT
   index  : INT;  // index to read from (1..4096).
   mem    : PTR;  // address to read from.
   len    : DINT; // number of elements to read.
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   buffer : ARRAY[1..10] OF DINT;
END_VAR;

...

// Read 10 elements directly from I/O space starting at location 120, and
place the data into 'buffer':
memioReadX(index:=120, mem:=ADDR(buffer), len:=10);

...

END_PROGRAM;
```

4.2.28 modbus: Functions for MODBUS

4.2.28.1 modbus: MODBUS communication

MODBUS is a messaging protocol for master/slave communication between devices connected on different types of buses or networks and has been the serial de facto standard of the industry since 1979.

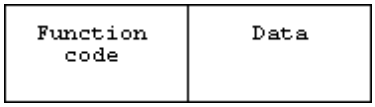
The MODBUS API is an alternative to the I/O Extension feature which gives the application full control over the MODBUS communication - thus allowing custom control over all requests and replies.
The MODBUS API supports the RTU protocol version, the ASCII protocol version, and the TCP protocol version.

Please note it is not possible to use the MODBUS API, MODBUS ASCII API and I/O Extension on the same port at the same time as they are equivalent to each other.

The following functions are used to access the MODBUS network.

- ioDeviceEnable — Enables or disables a device in the I/O Extension.
- ioInputLatchSet — Configure if inputs of devices in the I/O Extension should be latched.
- ioGetStatus — Gets the status for a device in the I/O Extension.
- ioNetRemove — Disables a net in the I/O Extension until device reset.
- ioNetEnable — Enables or disables a net in the I/O Extension.
- ioNetConfig — Configure an disabled I/O Extension net.
- ioSetMode — Sets the update mode for the I/O Extension devices.
- ioSynchronize — Synchronizes the I/O Extension devices.
- ioWaitException — Waits for an exception in the I/O Extension.
- modbusOpen — Opens the MODBUS RTU network on a serial port.
- modbusOpenX — Opens a MODBUS network.
- modbusClose — Closes a MODBUS network.
- modbusReceive — Receives a MODBUS message.
- modbusSend — Sends a MODBUS message.
- modbusWaitData — Waits for data received or timeout.
- modasciiOpen — Opens the MODBUS ASCII network on a serial port.
- modasciiClose — Closes a MODBUS ASCII network.
- modasciiReceive — Receives a MODBUS ASCII message.
- modasciiSend — Sends a MODBUS ASCII message.
- modasciiWaitData — Waits for ASCII data received or timeout.

The MODBUS message is made up of 2 fields: the function code and the function data.



The function code field is coded in one byte. Valid codes are in the range of 1...255 decimal (the range 128 – 255 is reserved and used for exception responses).
When a message is sent from a master to a slave device, the function code field tells the slave what kind of action to perform.

The data field of messages sent from a master to a slave devices contains additional information that the server uses to take the action defined by the function code.
This can include items like discrete and register addresses, the quantity of items to be handled, and the count of actual data bytes in the field.

The data field may be nonexistent (of zero length) in certain kinds of requests - in this case the server does not require any additional information.
The function code alone specifies the action.

Consult the MODBUS protocol for more information on MODBUS messages. For the commands used and supported by the I/O Extension, please see the MODBUS commands section.

The implementation of MODBUS has the following limitations:

- A maximum of 3 MODBUS networks can be used. (shared with I/O extension)
- A maximum of 2 MODBUS networks can use the same protocol. (RTU, ASCII or TCP)
- A maximum of 16 devices (32 devices under the NX32L architecture) can be connected to a TCP MODBUS network.

4.2.28.2 ioDeviceEnable (Function)

```
Architecture:       X32 / NX32 / NX32L
Device support:     All devices with RS485
Firmware version:   1.50 / 1.00.00
```

This function is used to enable or disable an I/O Extension device.
When a device is disabled, the firmware will no longer communicate with the device and the inputs and/or outputs on the device are no longer updated.
While a device is disabled, its status will not change and no new exceptions are raised.

The network ID is for the I/O Extension net that is found in the RTCU IDE, not the MODBUS connection ID returned by modbusOpen.

Input:

net_id : SINT
The network ID of the I/O Extension network.

dev_addr : INT
The address of the device.

enable : BOOL
TRUE: The I/O on the device is updated.
FALSE: The I/O on the device is no longer updated.

Returns: INT

0 - Success.
1 - The I/O net is not found.
2 - The I/O device is not found.

Declaration:

```
FUNCTION ioDeviceEnable : INT;
VAR_INPUT
   net_id   : SINT;
   dev_addr : INT;
   enable   : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM ModbusExample;
```

```
BEGIN
   ...
   IF cmd = 5 THEN
      // Disable device
      ioDeviceEnable(net_id := 1, dev_addr := device, enable := FALSE);
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.28.3 ioInputLatchSet (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with RS485
Firmware version: 4.66 / 1.08.00

This function is used to configure whether inputs (both Analog and Digital) of devices in I/O Extension, will be latched (keeping the last value) or reset (set to zero) when communication fails.
The default behavior is to latch the inputs.

Input:

reset : BOOL
TRUE: The inputs on devices is reset.
FALSE: The inputs on devices is latched.

Returns:

None.

Declaration:

```
FUNCTION ioInputLatchSet;
VAR_INPUT
   reset : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM ModbusExample;

   ...
   // Latch device inputs
   ioInputLatchSet(reset := FALSE);
   ...

BEGIN
END;
END_PROGRAM;
```

4.2.28.4 ioGetStatus (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with RS485
Firmware version: 1.50 / 1.00.00

This function will return the status code of an I/O Extension exception that has been raised by an I/O device.
While it is possible to use ioGetStatus to continuously poll the device for changes to its status, it is recommended to only poll the device after ioWaitException has detected an exception from the device.

The network ID is for the I/O Extension net that is found in the RTCU IDE, not the MODBUS connection ID returned by modbusOpen.

Input:

net_id : SINT
The network ID of the I/O Extension network.

dev_addr : INT
The address of the device.

Returns: INT

0 - The device is present.
1 - The modbus net is not open.
2 - The device is not present (no response from device).
3 - The I/O configuration is wrong (MODBUS exception received).
4 - Communication error (illegal/corrupted package received).

Declaration:

```
FUNCTION ioGetStatus : INT;
VAR_INPUT
   net_id   : SINT;
   dev_addr : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

THREAD_BLOCK modbusMonitor;
VAR
   device : INT;
   status : INT;
END_VAR;

   DebugMsg(message := "I/O monitor thread running");

WHILE TRUE DO
   device := ioWaitException(net_id := 1, timeout := -1);
   IF device > 0 THEN
      // Get status
      status := ioGetStatus(net_id := 1, dev_addr := device);
      // Show message
      CASE status OF
         0: DebugFmt(message := "Device \1 present!", v1 := device);
         2: DebugFmt(message := "Device \1 not present!", v1 := device);
         3: DebugFmt(message := "Device \1 configuration wrong!", v1 :=
device);
         4: DebugFmt(message := "Device \1 communication error!", v1 :=
device);
      ELSE
         DebugFmt(message := "Device \1 unknown exception (\2)!", v1 :=
device, v2 := status);
```

```
      END_CASE;
   END_IF;
END_WHILE;
END_THREAD_BLOCK;

PROGRAM ModbusExample;
VAR
   mbusMon : modbusMonitor;
END_VAR;

   mbusMon();

BEGIN
   ...
END;
END_PROGRAM;
```

4.2.28.5 ioNetRemove (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with RS485
Firmware version: 2.84 / 1.00.00

This function is used to disable an I/O Extension net until the next restart of the RTCU.
When a net is disabled, the firmware will release the serial port used and disable all the devices that are connected to the net.
Similar to the ioDeviceEnable function, the inputs and/or outputs of the devices are no longer updated, the status of the devices will not change and no new exceptions are raised.

The network ID is for the I/O Extension net that is found in the RTCU IDE, not the MODBUS connection ID returned by modbusOpen.

Input:

net_id : SINT
The network ID of the I/O Extension network.

Returns: INT

0 - Success.
1 - The I/O net is not found.

Declaration:

```
FUNCTION ioNetRemove : INT;
VAR_INPUT
   net_id   : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM ModbusExample;

   // Disable net
   ioNetRemove(net_id := 1);
   ...

BEGIN
   ...
```

```
END;
END_PROGRAM;
```

4.2.28.6 ioNetEnable (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with RS485
Firmware version: 4.56 / 1.00.00

This function is used to enable or disable an I/O Extension net.
When a net is disabled, the firmware will release the serial port used, and disable update of all the devices that are connected to the net.
Similar to the ioDeviceEnable function, the inputs and/or outputs of the devices are no longer updated, the status of the devices will not change and no new exceptions are raised.

The network ID is for the I/O Extension net that is found in the RTCU IDE, not the MODBUS connection ID returned by modbusOpen.

Input:

net_id : SINT
The network ID of the I/O Extension network.

enable : BOOL
TRUE: The I/O on devices connected on the net is updated.
FALSE: The I/O on devices connected on the net is no longer updated.

Returns: INT

0 - Success.
1 - The I/O net is not found.
2 - Could not open modbus net.
3 - Could not open serial port.

Declaration:

```
FUNCTION ioNetEnable : INT;
VAR_INPUT
   net_id   : SINT;
   enable   : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM ModbusExample;

   // Disable net
   ioNetEnable(net_id := 1, enable := OFF);
   ...

BEGIN
   ...
END;
END_PROGRAM;
```

4.2.28.7 ioNetConfig (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with RS485
Firmware version: 4.56 / 1.00.00

This function is used to change the configuration of an I/O Extension net.
The changes will be applied when the I/O Extension net is enabled with the ioNetEnable function.

The network ID is for the I/O Extension net that is found in the RTCU IDE, not the MODBUS connection ID returned by modbusOpen.

Input:

net_id : SINT
The network ID of the I/O Extension network.

port : SINT (-1,0..2) *(default -1)*
Selects which serial port to use (see serial port enumeration).
This parameter is ignored if set to -1.

baud : DINT (-1,9600,19200,38400,57600) *(default -1)*
Selects the desired baud rate.
This parameter is ignored if set to -1.

bit : SINT (-1,7/8) *(default -1)*
Selects the number of bits per character.
Note**:** 7 bit without parity bit only works on LX devices.
This parameter is ignored if set to -1.

parity : SINT (-1,0..2) *(default -1)*
Selects the desired parity, 0 is none, 1 is even, and 2 is odd.
This parameter is ignored if set to -1.

stop : SINT (-1,1/2) *(default -1)*
Selects number of stop bits.
This parameter is ignored if set to -1.

Returns: INT

0 - Success.
1 - The I/O net is not found.
2 - No changes to the configuration.
3 - Illegal parameter.

Declaration:

```
FUNCTION ioNetConfig : INT;
VAR_INPUT
   net_id   : SINT;
   port     : DINT := -1;
   baud     : DINT := -1;
   bit      : SINT := -1;
   parity   : SINT := -1;
   stop     : SINT := -1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM ModbusExample;

   // Move I/O Extension net 1 to serial port 2
   ioNetEnable(net_id := 1, enable := OFF);
   ioNetConfig(net_id := 1, port := 2);
   ioNetEnable(net_id := 1, enable := ON);

   ...

BEGIN
   ...
END;
END_PROGRAM;
```

4.2.28.8 ioSetMode (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with RS485
Firmware version: 2.10 / 1.00.00

This function will change how external I/O devices are updated.
There are two different update modes:

Asynchronous (0, Default)

The update of the external I/O devices is started in the BEGIN/END construction and is performed asynchronously with the onboard I/O.
This ensures that the communication will not slow down execution of the application while the variables bound to inputs in external I/O devices are not updated until the next instance of the BEGIN/END construction (possibly longer if the BEGIN/END loop is executed faster than the update communication).

Synchronous (1)

The external I/O devices are never updated in the BEGIN/END construction.
The update is performed with the ioSynchronize function which does not return until after the communication has been completed.
Note: the VPL variables bound to in- and outputs are not updated by the ioSynchronize function. This only happens in the BEGIN/END construction.

Note: UPDATEIO and UPDATEOUT work like the BEGIN/END construction for updating the external I/O devices.

Input:

mode : INT (0/1, *Default 0)*
The I/O Extension update mode.
0 Asynchronous update of I/O Extension devices.
1 Synchronous update of I/O Extension devices with ioSynchronize.

Returns:

0 - The mode is set.
1 - Unknown mode.

Declaration:

```
FUNCTION ioSetMode;
VAR_INPUT
```

```
   mode : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM ModbusExample;

ioSetMode(mode := 1);

BEGIN
   ioSynchronize();   // Update the external I/O
   ...
END;
END_PROGRAM;
```

4.2.28.9 ioSynchronize (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with RS485
Firmware version: 2.10 / 1.00.00

This function will update all the in- and outputs on the external I/O devices.
This function is required to update the external I/O devices when using synchronous update mode (see ioSetMode).

Note: this function does not update the VPL variables bound to in- and outputs. To update the variables you need the BEGIN/END construction or the UPDATEIO construction.

Input:

None.

Returns:

0 - The external I/O devices are updated.
1 - ioSynchronize does not work in the current update mode.

Declaration:

```
FUNCTION ioSynchronize;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM ModbusExample;

ioSetMode(mode := 1);

BEGIN
   ioSynchronize();   // Update the external I/O
   ...
END;
END_PROGRAM;
```

4.2.28.10 ioWaitException (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with RS485
Firmware version: 1.50 / 1.00.00

This function will wait for an I/O Extension exception or an optional timeout.
When an exception is raised, use ioGetStatus to get the reason for the exception.

The network ID is for the I/O Extension net that is found in the RTCU IDE, not the MODBUS connection ID returned by modbusOpen.

Note: the function will block the thread while waiting.

Input:

net_id : SINT
The network ID of the I/O Extension network.

timeout : INT (-1,0..32000) *(default -1)*
Timeout period in milliseconds to wait.
0 - Disable timeout. The function will return immediately.
-1 - Wait forever. The function will only return only when an exception is raised.

Returns: INT

The ID of the device that raised the exception.
0 - The I/O Extension network does not exist.
-1 - No exception raised within the specified period.

Declaration:

```
FUNCTION ioWaitException : INT;
VAR_INPUT
   net_id  : SINT;
   timeout : INT := -1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

THREAD_BLOCK modbusMonitor;
VAR
   device : INT;
   status : INT;
END_VAR;

   DebugMsg(message := "I/O monitor thread running");

WHILE TRUE DO
   device := ioWaitException(net_id := 1, timeout := -1);
   IF device > 0 THEN
      // Get status
      status := ioGetStatus(net_id := 1, dev_addr := device);
      // Show message
      CASE status OF
         0: DebugFmt(message := "Device \1 present!", v1 := device);
         2: DebugFmt(message := "Device \1 not present!", v1 := device);
         3: DebugFmt(message := "Device \1 configuration wrong!", v1 :=
```

```
device);
         4: DebugFmt(message := "Device \1 communication error!", v1 :=
device);
      ELSE
         DebugFmt(message := "Device \1 unknown exception (\2)!", v1 :=
device, v2 := status);
      END_CASE;
   END_IF;
END_WHILE;
END_THREAD_BLOCK;

PROGRAM ModbusExample;
VAR
   mbusMon : modbusMonitor;
END_VAR;

   mbusMon();

BEGIN
   ...
END;
END_PROGRAM;
```

4.2.28.11 modbusOpen (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with RS485
Firmware version: 1.50 / 1.00.00

This function will open a connection to a MODBUS RTU network.

The device supports both master mode, where the device controls the MODBUS network and polls the slaves, and slave mode, where the device waits for commands from the master.

Input:

mode : SINT (0/1)
0 - The RTCU device is master on the MODBUS network.
1 - The RTCU device is a slave on the MODBUS network.

unit_id : INT (1..247)
The address of the RTCU device on the MODBUS network.
Note: this parameter is ignored if mode is set to master.

port : DINT (0..2) *(default 0)*
Selects which serial port to use (see serial port enumeration).

baud : DINT (9600,19200,38400,57600,115200) *(default 115200)*
Selects the desired baud rate.

bit : SINT (7/8) *(default 8)*
Selects the number of bits per character.
Note**:** 7 bit without parity bit only works on LX devices.

parity : SINT (0..2) *(default 1)*
Selects the desired parity, 0 is none, 1 is even, and 2 is odd.

stopbit : SINT (1/2) *(default 1)*
Selects number of stop bits.

Note: Only 1 stop bit is supported on the RS485 ports on NX devices.

Returns: INT

ID of the MODBUS connection.
- -2 - Illegal unit_id selected.
- -5 - Illegal serial port.
- -6 - All MODBUS connections are in use.
- -7 - Illegal serial port parameters.

Declaration:

```
FUNCTION modbusOpen : INT;
VAR_INPUT
   mode    : INT;
   unit_id : INT;
   port    : DINT := 0;
   baud    : DINT := 115200;
   bit     : SINT := 8;
   parity  : SINT := 1;
   stopbit : SINT := 1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   mbNet : INT;
   mbRcv : modbusReceive;
END_VAR;

FUNCTION setINT;
VAR_INPUT
   adr : PTR;
   v   : INT;
END_VAR;
   memcpy(dst := adr + 1, src := ADDR(v)    , len := 1);
   memcpy(dst := adr    , src := ADDR(v) + 1, len := 1);
END_FUNCTION;

FUNCTION_BLOCK modbusReadDiscreteInputs;
VAR_INPUT
   net_id  : INT;
   unit_id : INT;
   index   : INT;
   length  : INT;
END_VAR;
VAR_OUTPUT
   status : INT;
   size   : INT;
   input  : ARRAY[1..32] OF BOOL;
END_VAR;
VAR
   rc, i  : INT;
   buf    : ARRAY[1..253] OF SINT;
   mask   : SINT;
END_VAR;

   // Check length
   IF length < 1 OR length > 32 THEN
      status := 1;
      RETURN;
   END_IF;
```

```
   IF index < 1 THEN
      status := 2;
      RETURN;
   END_IF;

   // Build command
   buf[1] := 16#02;
   setINT(adr := ADDR(buf[2]), v := index - 1);
   setINT(adr := ADDR(buf[4]), v := length);

   // Send command
   rc := modbusSend(net_id := net_id, unit_id := unit_id, frame := ADDR(buf),
size := 5);
   IF rc <> 0 THEN
      status := 3;
      RETURN;
   END_IF;

   // Wait for reply
   IF NOT modbusWaitData(net_id := net_id, timeout := 5000) THEN
      status := 4;
      RETURN;
   END_IF;

   // Read reply
   mbRcv(net_id := net_id, frame := ADDR(buf), maxsize := SIZEOF(buf));
   IF mbRcv.status <> 0 THEN
      status := 5;
      RETURN;
   END_IF;
   IF mbRcv.unit_id <> unit_id THEN
      status := 6;
      RETURN;
   END_IF;
   IF buf[1] <> 16#02 THEN
      status := 7;
      RETURN;
   END_IF;

   // Parse reply
   mask := 1;
   FOR i := 1 TO length DO
      input[i] := BOOL(buf[3 + ((i - 1) / 8)] AND mask);
      mask := shl8(in := mask, n := 1);
      IF mask = 0 THEN mask := 1; END_IF;
   END_FOR;
   size   := length;
   status := 0;
END_FUNCTION_BLOCK;

PROGRAM ModbusExample;
VAR
   dinRead : modbusReadDiscreteInputs;
   linsec  : DINT;
   i       : INT;
END_VAR;

   DebugMsg(message := "Initializing...");
   mbNet := modbusOpen(port := 2, baud := 19200, parity := 0, mode := 0);
   DebugFmt(message := "modbus...\1", v1 := mbNet);

BEGIN
   IF clockNow() > linsec THEN
      DebugMsg(message := "-----------------------------");
```

```
      DebugMsg(message := "Reading inputs:");
      dinRead(net_id := mbNet, unit_id := 9, index := 1, length := 4);
      DebugFmt(message := "-status=\1", v1 := dinRead.status);
      IF dinRead.status = 0 THEN
         FOR i := 1 TO dinRead.size DO
            DebugFmt(message := "-input[\1]=\2", v1 := i, v2 :=
INT(dinRead.input[i]));
         END_FOR;
      END_IF;
      // Set next reading
      linsec := clockNow() + 5;
   END_IF;
END;
END_PROGRAM;
```

4.2.28.12 modbusOpenX (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with RS485 or LAN
Firmware version: 4.50 / 1.00.00

This function will open a connection to a MODBUS network.
The connection uses either the RTU, ASCII or TCP protocol.

The device supports both master mode, where the device controls the MODBUS network and polls the slaves, and slave mode, where the device waits for commands from the master.
Please note that the TCP protocol can only be used via the LAN network interface, which must be opened before communication is possible.

For NX32L devices, all interfaces can be used with the TCP protocol (from firmware version 1.08.00).

Input:

mode : SINT

1 - RTU protocol
2 - ASCII protocol
3 - TCP protocol

unit_id : INT (0..247)
The address of the RTCU device on the MODBUS network.
If this parameter is set to zero, the RTCU device will be the master on the network.

port : DINT *(default -1)*
When port is set to -1, the default port for the selected mode is used.
mode = 1 Selects which serial port to use (see serial port enumeration, default is port 0).
mode = 2 Selects which serial port to use (see serial port enumeration, default is port 0).
mode = 3 Selects which IP port to listen for connections from Masters on. (default is port 502)

iface : SINT *(default 0)*
The network interface from which incoming TCP connections are accepted. (See Network)
The network interface must be present and connected, when modbusOpenX is called if set.
This parameter is ignored if mode != 3 or if the RTCU device is the master.

(from firmware version 1.32.00).

con_timeout : UINT (0..30000) ***(default 1000)***
The number of milliseconds to try to connect to a slave before timing out.
This parameter is ignored if mode != 3 or if the RTCU device is not the master.

baud : DINT (9600,19200,38400,57600,115200) ***(default 115200)***
Selects the desired baud rate.
This parameter is ignored if mode 3 is selected.

bit : SINT (7/8) ***(default 8)***
Selects the number of bits per character.
This parameter is ignored if mode 3 is selected.
Note**:** 7 bit without parity bit only works on LX devices.

parity : SINT (0..2) ***(default 1)***
Selects the desired parity, 0 is none, 1 is even, and 2 is odd.
This parameter is ignored if mode 3 is selected.

stopbit : SINT (1/2) ***(default 1)***
Selects number of stop bits.
This parameter is ignored if mode 3 is selected.

Note: Only 1 stop bit is supported on the RS485 ports on NX devices.

rs485 : SINT (-1..1) ***(default -1)***
-1 - Auto detect. First try to use RS485 then RS232.
0 - Use RS232
1 - Use RS485
This parameter is ignored if mode 3 is selected.

allow1 : DINT ***(default 0)***
Remote IP address accepted. 0 if all is allowed.
This parameter is ignored if mode != 3 or if the RTCU device is the master.

allow2 : DINT ***(default 0)***
Remote IP address accepted. 0 if all is allowed.
This parameter is ignored if mode != 3 or if the RTCU device is the master.

allow3 : DINT ***(default 0)***
Remote IP address accepted. 0 if all is allowed.
This parameter is ignored if mode != 3 or if the RTCU device is the master.

allow4 : DINT ***(default 0)***
Remote IP address accepted. 0 if all is allowed.
This parameter is ignored if mode != 3 or if the RTCU device is the master.

allow5 : DINT ***(default 0)***
Remote IP address accepted. 0 if all is allowed.
This parameter is ignored if mode != 3 or if the RTCU device is the master.

allow6 : DINT ***(default 0)***
Remote IP address accepted. 0 if all is allowed.
This parameter is ignored if mode != 3 or if the RTCU device is the master.

allow7 : DINT ***(default 0)***
Remote IP address accepted. 0 if all is allowed.
This parameter is ignored if mode != 3 or if the RTCU device is the master.

allow8 : DINT ***(default 0)***
Remote IP address accepted. 0 if all is allowed.
This parameter is ignored if mode != 3 or if the RTCU device is the master.

allow9 : DINT ***(default 0)***
Remote IP address accepted. 0 if all is allowed.
This parameter is ignored if mode != 3 or if the RTCU device is the master.

allow10 : DINT ***(default 0)***
Remote IP address accepted. 0 if all is allowed.
This parameter is ignored if mode != 3 or if the RTCU device is the master.

Returns: INT

ID of the MODBUS connection.

-1 - All MODBUS connections are in use.
-2 - Illegal unit_id selected.
-5 - Illegal serial port.
-6 - Failed to open connection.
-7 - Illegal serial port parameters.
-8 - Illegal mode selected.
-9 - The network interface is not connected.

Declaration:

```
FUNCTION modbusOpenX : INT;
VAR_INPUT
   mode    : SINT;
   unit_id : INT;
   port    : DINT := -1;
   iface   : SINT := 0;
   con_timeout : UINT := 1000;
   baud    : DINT := 115200;
   bit     : SINT := 8;
   parity  : SINT := 1;
   stopbit : SINT := 1;
   rs485   : SINT := -1;
   allow1  : DINT := 0;
   allow2  : DINT := 0;
   allow3  : DINT := 0;
   allow4  : DINT := 0;
   allow5  : DINT := 0;
   allow6  : DINT := 0;
   allow7  : DINT := 0;
   allow8  : DINT := 0;
   allow9  : DINT := 0;
   allow10 : DINT := 0;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   mbNet : INT;
   mbRcv : modbusReceive;
END_VAR;

FUNCTION setINT;
VAR_INPUT
   adr : PTR;
   v   : INT;
END_VAR;
```

```
   memcpy(dst := adr + 1, src := ADDR(v)    , len := 1);
   memcpy(dst := adr    , src := ADDR(v) + 1, len := 1);
END_FUNCTION;

FUNCTION_BLOCK modbusReadDiscreteInputs;
VAR_INPUT
   net_id  : INT;
   unit_id : INT;
   index   : INT;
   length  : INT;
END_VAR;
VAR_OUTPUT
   status : INT;
   size   : INT;
   input  : ARRAY[1..32] OF BOOL;
END_VAR;
VAR
   rc, i  : INT;
   buf    : ARRAY[1..253] OF SINT;
   mask   : SINT;
END_VAR;

   // Check length
   IF length < 1 OR length > 32 THEN
      status := 1;
      RETURN;
   END_IF;
   IF index < 1 THEN
      status := 2;
      RETURN;
   END_IF;

   // Build command
   buf[1] := 16#02;
   setINT(adr := ADDR(buf[2]), v := index - 1);
   setINT(adr := ADDR(buf[4]), v := length);

   // Send command
   rc := modbusSend(net_id := net_id, unit_id := unit_id, frame := ADDR(buf),
size := 5);
   IF rc <> 0 THEN
      status := 3;
      RETURN;
   END_IF;

   // Wait for reply
   IF NOT modbusWaitData(net_id := net_id, timeout := 5000) THEN
      status := 4;
      RETURN;
   END_IF;

   // Read reply
   mbRcv(net_id := net_id, frame := ADDR(buf), maxsize := SIZEOF(buf));
   IF mbRcv.status <> 0 THEN
      status := 5;
      RETURN;
   END_IF;
   IF mbRcv.unit_id <> unit_id THEN
      status := 6;
      RETURN;
   END_IF;
   IF buf[1] <> 16#02 THEN
      status := 7;
      RETURN;
```

```
   END_IF;

   // Parse reply
   mask := 1;
   FOR i := 1 TO length DO
      input[i] := BOOL(buf[3 + ((i - 1) / 8)] AND mask);
      mask := shl8(in := mask, n := 1);
      IF mask = 0 THEN mask := 1; END_IF;
   END_FOR;
   size   := length;
   status := 0;
END_FUNCTION_BLOCK;

PROGRAM ModbusExample;
VAR
   dinRead : modbusReadDiscreteInputs;
   linsec  : DINT;
   i       : INT;
END_VAR;

   DebugMsg(message := "Initializing...");
   mbNet := modbusOpenX(mode := 1, unit_id := 0, port := 2, baud := 19200,
parity := 0);
   DebugFmt(message := "modbus...\1", v1 := mbNet);

BEGIN
   IF clockNow() > linsec THEN
      DebugMsg(message := "------------------------------");
      DebugMsg(message := "Reading inputs:");
      dinRead(net_id := mbNet, unit_id := 9, index := 1, length := 4);
      DebugFmt(message := "-status=\1", v1 := dinRead.status);
      IF dinRead.status = 0 THEN
         FOR i := 1 TO dinRead.size DO
            DebugFmt(message := "-input[\1]=\2", v1 := i, v2 :=
INT(dinRead.input[i]));
         END_FOR;
      END_IF;
      // Set next reading
      linsec := clockNow() + 5;
   END_IF;
END;
END_PROGRAM;
```

4.2.28.13 modbusClose (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with RS485 or LAN
Firmware version: 4.50 / 1.00.00

This function will close a MODBUS connection, previously opened by the modbusOpen or modbusOpenX functions.

Input:

net_id : INT
The ID of the MODBUS connection.

Returns: INT

0 - Success.
-1 - Illegal MODBUS connection ID.

Declaration:

```
FUNCTION modbusClose;
VAR_INPUT
   net_id  : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   switch : BOOL R_EDGE;
END_VAR;

VAR
   mb_net : INT;
   mb_id  : INT := 0; // open as master
END_VAR;

PROGRAM ModbusExample;

   mb_net := modbusOpenX(mode := 1, unit_id := mb_id, port := 2, baud :=
57600, parity := 0);

BEGIN
   IF switch THEN
      // Close MODBUS net
      modbusClose(net_id := mb_net);

      // Open new MODBUS net
      IF mb_id = 0 THEN
         mb_id := 1; // open as slave
      ELSE
         mb_id := 0; // open as master
      END_IF;
      mb_net := modbusOpenX(mode := 1, unit_id := mb_id, port := 2, baud :=
57600, parity := 0);
   END_IF;
END;
END_PROGRAM;
```

4.2.28.14 modbusReceive (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: All devices with RS485 or LAN
Firmware version: 1.50 / 1.00.00

This function will read a package that has been received from a MODBUS connection. While it is possible to use modbusReceive to continuously poll for received packages, it is recommended to only poll after modbusWaitData has detected that a package has been received. This function is non-blocking and will therefore return immediately with the result of the operation.

Note: if the received package is too large for the receive buffer, only the part of the message that can fit the buffer is actually returned.

Input:

net_id : INT
The ID of the MODBUS connection.

frame : PTR
Address of the buffer where the received package is stored.

maxsize : INT
Maximum size of the receive buffer.

Output:

unit_id : INT
The ID of the device on the MODBUS network who sent the package.
Note: if the MODBUS connection is opened in slave mode, this will be the address of the RTCU device.

size : INT
Number of bytes received.

tid : DINT
The transaction ID of the package. This parameter is only used by MODBUS TCP connections.

status : INT
 0 - Success.
-1 - Not a valid MODBUS connection ID.
-3 - No data received.
-7 - Data received is not a valid MODBUS message.

Declaration:

```
FUNCTION_BLOCK modbusReceive;
VAR_INPUT
   net_id  : INT;
   frame   : PTR;
   maxsize : INT;
END_VAR;
VAR_OUTPUT
   unit_id : INT;
   size    : INT;
   status  : INT;
   tid     : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   mbNet : INT;
   mbRcv : modbusReceive;
END_VAR;

FUNCTION setINT;
VAR_INPUT
   adr : PTR;
   v   : INT;
END_VAR;
   memcpy(dst := adr + 1, src := ADDR(v)    , len := 1);
   memcpy(dst := adr    , src := ADDR(v) + 1, len := 1);
END_FUNCTION;

FUNCTION_BLOCK modbusReadDiscreteInputs;
VAR_INPUT
```

```
    net_id  : INT;
    unit_id : INT;
    index   : INT;
    length  : INT;
END_VAR;
VAR_OUTPUT
    status : INT;
    size   : INT;
    input  : ARRAY[1..32] OF BOOL;
END_VAR;
VAR
    rc, i  : INT;
    buf    : ARRAY[1..253] OF SINT;
    mask   : SINT;
END_VAR;

    // Check length
    IF length < 1 OR length > 32 THEN
        status := 1;
        RETURN;
    END_IF;
    IF index < 1 THEN
        status := 2;
        RETURN;
    END_IF;

    // Build command
    buf[1] := 16#02;
    setINT(adr := ADDR(buf[2]), v := index - 1);
    setINT(adr := ADDR(buf[4]), v := length);

    // Send command
    rc := modbusSend(net_id := net_id, unit_id := unit_id, frame := ADDR(buf),
size := 5);
    IF rc <> 0 THEN
        status := 3;
        RETURN;
    END_IF;

    // Wait for reply
    IF NOT modbusWaitData(net_id := net_id, timeout := 5000) THEN
        status := 4;
        RETURN;
    END_IF;

    // Read reply
    mbRcv(net_id := net_id, frame := ADDR(buf), maxsize := SIZEOF(buf));
    IF mbRcv.status <> 0 THEN
        status := 5;
        RETURN;
    END_IF;
    IF mbRcv.unit_id <> unit_id THEN
        status := 6;
        RETURN;
    END_IF;
    IF buf[1] <> 16#02 THEN
        status := 7;
        RETURN;
    END_IF;

    // Parse reply
    mask := 1;
    FOR i := 1 TO length DO
        input[i] := BOOL(buf[3 + ((i - 1) / 8)] AND mask);
```

```
      mask := shl8(in := mask, n := 1);
      IF mask = 0 THEN mask := 1; END_IF;
   END_FOR;
   size   := length;
   status := 0;
END_FUNCTION_BLOCK;

PROGRAM ModbusExample;
VAR
   dinRead : modbusReadDiscreteInputs;
   linsec  : DINT;
   i       : INT;
END_VAR;

   DebugMsg(message := "Initializing...");
   mbNet := modbusOpen(port := 2, baud := 19200, parity := 0, mode := 0);
   DebugFmt(message := "modbus...\1", v1 := mbNet);

BEGIN
   IF clockNow() > linsec THEN
      DebugMsg(message := "------------------------------");
      DebugMsg(message := "Reading inputs:");
      dinRead(net_id := mbNet, unit_id := 9, index := 1, length := 4);
      DebugFmt(message := "-status=\1", v1 := dinRead.status);
      IF dinRead.status = 0 THEN
         FOR i := 1 TO dinRead.size DO
            DebugFmt(message := "-input[\1]=\2", v1 := i, v2 :=
INT(dinRead.input[i]));
         END_FOR;
      END_IF;
      // Set next reading
      linsec := clockNow() + 5;
   END_IF;
END;
END_PROGRAM;
```

4.2.28.15 modbusSend (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with RS485 or LAN
Firmware version: 1.50 / 1.00.00

This function will send a package to a device on a MODBUS connection.

Input:

net_id : INT
The ID of the MODBUS connection.

unit_id : INT
The address of the device on the MODBUS connection to receive the data.
Note: if the MODBUS connection is opened in slave mode, this parameter is ignored.

ip : DINT
The IP address of the receiving device. This parameter is only used by MODBUS TCP connections.
Note: if the MODBUS connection is opened in slave mode, this parameter is ignored.

port : DINT (*default 502*)
The IP port of the receiving device. This parameter is only used by MODBUS TCP connections.
Note: if the MODBUS connection is opened in slave mode, this parameter is ignored.

iface : SINT (*default 2*)
The network interface to use. 1 = Mobile network, 2 = LAN network, etc. (See Network)
This parameter is only used on NX32L devices for MODBUS TCP connections.
Note: if the MODBUS connection is opened in slave mode, this parameter is ignored.

tid : DINT
The transaction ID of the package. This parameter is only used by MODBUS TCP connections.

frame : PTR
Address of the buffer that contains the package to send.

size : INT (1..253)
Number of bytes to send from the buffer.

Returns: INT

0	- Success.
-1	- Illegal MODBUS connection ID.
-2	- The size is too large.
-3	- No data to send.
-4	- MODBUS connection is busy.
-5	- Illegal IP parameters. (MODBUS TCP only)
-6	- TCP communication error. (MODBUS TCP only)

Declaration:

```
FUNCTION modbusSend;
VAR_INPUT
   net_id  : INT;
   unit_id : INT;
   frame   : PTR;
   size    : INT;
   ip      : DINT;
   port    : DINT;
   iface   : SINT;
   tid     : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   mbNet : INT;
   mbRcv : modbusReceive;
END_VAR;

FUNCTION setINT;
VAR_INPUT
   adr : PTR;
   v   : INT;
END_VAR;
   memcpy(dst := adr + 1, src := ADDR(v)    , len := 1);
   memcpy(dst := adr    , src := ADDR(v) + 1, len := 1);
END_FUNCTION;

FUNCTION_BLOCK modbusReadDiscreteInputs;
VAR_INPUT
   net_id  : INT;
   unit_id : INT;
```

```
   index   : INT;
   length  : INT;
END_VAR;
VAR_OUTPUT
   status : INT;
   size   : INT;
   input  : ARRAY[1..32] OF BOOL;
END_VAR;
VAR
   rc, i  : INT;
   buf    : ARRAY[1..253] OF SINT;
   mask   : SINT;
END_VAR;

   // Check length
   IF length < 1 OR length > 32 THEN
      status := 1;
      RETURN;
   END_IF;
   IF index < 1 THEN
      status := 2;
      RETURN;
   END_IF;

   // Build command
   buf[1] := 16#02;
   setINT(adr := ADDR(buf[2]), v := index - 1);
   setINT(adr := ADDR(buf[4]), v := length);

   // Send command
   rc := modbusSend(net_id := net_id, unit_id := unit_id, frame := ADDR(buf),
size := 5);
   IF rc <> 0 THEN
      status := 3;
      RETURN;
   END_IF;

   // Wait for reply
   IF NOT modbusWaitData(net_id := net_id, timeout := 5000) THEN
      status := 4;
      RETURN;
   END_IF;

   // Read reply
   mbRcv(net_id := net_id, frame := ADDR(buf), maxsize := SIZEOF(buf));
   IF mbRcv.status <> 0 THEN
      status := 5;
      RETURN;
   END_IF;
   IF mbRcv.unit_id <> unit_id THEN
      status := 6;
      RETURN;
   END_IF;
   IF buf[1] <> 16#02 THEN
      status := 7;
      RETURN;
   END_IF;

   // Parse reply
   mask := 1;
   FOR i := 1 TO length DO
      input[i] := BOOL(buf[3 + ((i - 1) / 8)] AND mask);
      mask := shl8(in := mask, n := 1);
      IF mask = 0 THEN mask := 1; END_IF;
```

```
   END_FOR;
   size   := length;
   status := 0;
END_FUNCTION_BLOCK;

PROGRAM ModbusExample;
VAR
   dinRead : modbusReadDiscreteInputs;
   linsec  : DINT;
   i       : INT;
END_VAR;

   DebugMsg(message := "Initializing...");
   mbNet := modbusOpen(port := 2, baud := 19200, parity := 0, mode := 0);
   DebugFmt(message := "modbus...\1", v1 := mbNet);

BEGIN
   IF clockNow() > linsec THEN
      DebugMsg(message := "------------------------------");
      DebugMsg(message := "Reading inputs:");
      dinRead(net_id := mbNet, unit_id := 9, index := 1, length := 4);
      DebugFmt(message := "-status=\1", v1 := dinRead.status);
      IF dinRead.status = 0 THEN
         FOR i := 1 TO dinRead.size DO
            DebugFmt(message := "-input[\1]=\2", v1 := i, v2 :=
INT(dinRead.input[i]));
         END_FOR;
      END_IF;
      // Set next reading
      linsec := clockNow() + 5;
   END_IF;
END;
END_PROGRAM;
```

4.2.28.16 modbusWaitData (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with RS485 or LAN
Firmware version: 1.50 / 1.00.00

This function will wait until a data package is received from a MODBUS connection - or for an optional timeout.
This function only indicates that the data is received. To actually read the data, modbusReceive must be used.

Note: waiting for a package by using this function will block the calling thread.

Input:

net_id : INT
The ID of the MODBUS connection.

timeout : INT (-1,0..32000) *(default -1)*
Timeout period in milliseconds to wait.
0 - Disable timeout. The function will return immediately.
-1 - Wait forever. The function will only return when data is received.

Returns: BOOL

TRUE: If data is ready to be read.
FALSE: If no data is present.

Declaration:

```
FUNCTION modbusWaitData : BOOL;
VAR_INPUT
   net_id  : INT;
   timeout : INT := -1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   mbNet : INT;
   mbRcv : modbusReceive;
END_VAR;

FUNCTION setINT;
VAR_INPUT
   adr : PTR;
   v   : INT;
END_VAR;
   memcpy(dst := adr + 1, src := ADDR(v)    , len := 1);
   memcpy(dst := adr    , src := ADDR(v) + 1, len := 1);
END_FUNCTION;

FUNCTION_BLOCK modbusReadDiscreteInputs;
VAR_INPUT
   net_id  : INT;
   unit_id : INT;
   index   : INT;
   length  : INT;
END_VAR;
VAR_OUTPUT
   status : INT;
   size   : INT;
   input  : ARRAY[1..32] OF BOOL;
END_VAR;
VAR
   rc, i  : INT;
   buf    : ARRAY[1..253] OF SINT;
   mask   : SINT;
END_VAR;

   // Check length
   IF length < 1 OR length > 32 THEN
      status := 1;
      RETURN;
   END_IF;
   IF index < 1 THEN
      status := 2;
      RETURN;
   END_IF;

   // Build command
   buf[1] := 16#02;
   setINT(adr := ADDR(buf[2]), v := index - 1);
   setINT(adr := ADDR(buf[4]), v := length);

   // Send command
   rc := modbusSend(net_id := net_id, unit_id := unit_id, frame := ADDR(buf),
size := 5);
```

```
   IF rc <> 0 THEN
      status := 3;
      RETURN;
   END_IF;

   // Wait for reply
   IF NOT modbusWaitData(net_id := net_id, timeout := 5000) THEN
      status := 4;
      RETURN;
   END_IF;

   // Read reply
   mbRcv(net_id := net_id, frame := ADDR(buf), maxsize := SIZEOF(buf));
   IF mbRcv.status <> 0 THEN
      status := 5;
      RETURN;
   END_IF;
   IF mbRcv.unit_id <> unit_id THEN
      status := 6;
      RETURN;
   END_IF;
   IF buf[1] <> 16#02 THEN
      status := 7;
      RETURN;
   END_IF;

   // Parse reply
   mask := 1;
   FOR i := 1 TO length DO
      input[i] := BOOL(buf[3 + ((i - 1) / 8)] AND mask);
      mask := shl8(in := mask, n := 1);
      IF mask = 0 THEN mask := 1; END_IF;
   END_FOR;
   size   := length;
   status := 0;
END_FUNCTION_BLOCK;

PROGRAM ModbusExample;
VAR
   dinRead : modbusReadDiscreteInputs;
   linsec  : DINT;
   i       : INT;
END_VAR;

   DebugMsg(message := "Initializing...");
   mbNet := modbusOpen(port := 2, baud := 19200, parity := 0, mode := 0);
   DebugFmt(message := "modbus...\1", v1 := mbNet);

BEGIN
   IF clockNow() > linsec THEN
      DebugMsg(message := "-----------------------------");
      DebugMsg(message := "Reading inputs:");
      dinRead(net_id := mbNet, unit_id := 9, index := 1, length := 4);
      DebugFmt(message := "-status=\1", v1 := dinRead.status);
      IF dinRead.status = 0 THEN
         FOR i := 1 TO dinRead.size DO
            DebugFmt(message := "-input[\1]=\2", v1 := i, v2 :=
INT(dinRead.input[i]));
         END_FOR;
      END_IF;
      // Set next reading
      linsec := clockNow() + 5;
   END_IF;
```

```
END;
END_PROGRAM;
```

4.2.28.17 modasciiOpen (Function)

```
Architecture:       X32 / NX32 / NX32L
Device support:     All devices with RS485
Firmware version:   4.00 / 1.00.00
```

This function will open a connection to a MODBUS ASCII network.

The device supports both master mode, where the device controls the MODBUS network and polls the slaves, and slave mode, where the device waits for commands from the master.

Both RS232 and RS485 is supported.

Input:

mode : SINT (0/1)
0 - The RTCU device is master on the MODBUS network.
1 - The RTCU device is a slave on the MODBUS network.

unit_id : INT (1..247) *(default 1)*
The address of the RTCU device on the MODBUS network.
Note: this parameter is ignored if mode is set to master.

port : SINT (0..2) *(default 0)*
Selects which serial port to use (see serial port enumeration).

baud : DINT (9600,19200,38400,57600,115200) *(default 115200)*
Selects the desired baud rate.

bit : SINT (7/8) *(default 8)*
Selects the number of bits per character.
Note**:** 7 bit without parity bit only works on LX devices.

parity : SINT (0..2) *(default 1)*
Selects the desired parity, 0 is none, 1 is even, and 2 is odd.

stopbit : SINT (1/2) *(default 1)*
Selects number of stop bits.

Note: Only 1 stop bit is supported on the RS485 ports on NX devices.

rs485 : BOOL *(default ON)*
ON - Use RS485 for communication.
OFF - Use RS232 for communication.

Returns: INT

ID of the MODBUS connection (1..2).
-5 - Illegal serial port.
-6 - All MODBUS connections are in use.
-7 - Illegal serial port parameters.

Declaration:

```
FUNCTION modasciiOpen : INT;
VAR_INPUT
   mode    : SINT;
   unit_id : INT  := 1;
```

```
   port    : SINT := 0;
   baud    : DINT := 115200;
   bit     : SINT := 8;
   parity  : SINT := 1;
   stopbit : SINT := 1;
   rs485   : BOOL := ON;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   mbNet : INT;
   mbRcv : modasciiReceive;
END_VAR;

FUNCTION setINT;
VAR_INPUT
   adr : PTR;
   v   : INT;
END_VAR;
   memcpy(dst := adr + 1, src := ADDR(v)    , len := 1);
   memcpy(dst := adr    , src := ADDR(v) + 1, len := 1);
END_FUNCTION;

FUNCTION_BLOCK modbusReadDiscreteInputs;
VAR_INPUT
   net_id  : INT;
   unit_id : INT;
   index   : INT;
   length  : INT;
END_VAR;
VAR_OUTPUT
   status : INT;
   size   : INT;
   input  : ARRAY[1..32] OF BOOL;
END_VAR;
VAR
   rc, i  : INT;
   buf    : ARRAY[1..253] OF SINT;
   mask   : SINT;
END_VAR;

   // Check length
   IF length < 1 OR length > 32 THEN
      status := 1;
      RETURN;
   END_IF;
   IF index < 1 THEN
      status := 2;
      RETURN;
   END_IF;

   // Build command
   buf[1] := 16#02;
   setINT(adr := ADDR(buf[2]), v := index - 1);
   setINT(adr := ADDR(buf[4]), v := length);

   // Send command
   rc := modasciiSend(net_id := net_id, unit_id := unit_id, frame :=
ADDR(buf), size := 5);
   IF rc <> 0 THEN
      status := 3;
```

```
      RETURN;
   END_IF;

   // Wait for reply
   IF NOT modasciiWaitData(net_id := net_id, timeout := 5000) THEN
      status := 4;
      RETURN;
   END_IF;

   // Read reply
   mbRcv(net_id := net_id, frame := ADDR(buf), maxsize := SIZEOF(buf));
   IF mbRcv.status <> 0 THEN
      status := 5;
      RETURN;
   END_IF;
   IF mbRcv.unit_id <> unit_id THEN
      status := 6;
      RETURN;
   END_IF;
   IF buf[1] <> 16#02 THEN
      status := 7;
      RETURN;
   END_IF;

   // Parse reply
   mask := 1;
   FOR i := 1 TO length DO
      input[i] := BOOL(buf[3 + ((i - 1) / 8)] AND mask);
      mask := shl8(in := mask, n := 1);
      IF mask = 0 THEN mask := 1; END_IF;
   END_FOR;
   size   := length;
   status := 0;
END_FUNCTION_BLOCK;

PROGRAM ModbusExample;
VAR
   dinRead : modbusReadDiscreteInputs;
   linsec  : DINT;
   i       : INT;
END_VAR;

   DebugMsg(message := "Initializing...");
   mbNet := modasciiOpen(port := 2, baud := 19200, parity := 0, mode := 0);
   DebugFmt(message := "modbus...\1", v1 := mbNet);

BEGIN
   IF clockNow() > linsec THEN
      DebugMsg(message := "------------------------------");
      DebugMsg(message := "Reading inputs:");
      dinRead(net_id := mbNet, unit_id := 9, index := 1, length := 4);
      DebugFmt(message := "-status=\1", v1 := dinRead.status);
      IF dinRead.status = 0 THEN
         FOR i := 1 TO dinRead.size DO
            DebugFmt(message := "-input[\1]=\2", v1 := i, v2 :=
INT(dinRead.input[i]));
         END_FOR;
      END_IF;
      // Set next reading
      linsec := clockNow() + 5;
   END_IF;
END;
END_PROGRAM;
```

4.2.28.18 modasciiClose (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with RS485
Firmware version: 4.50 / 1.00.00

This function will close a MODBUS ASCII connection, previously opened by the modasciiOpen function.

Input:

net_id : INT
The ID of the MODBUS connection.

Returns: INT

0 - Success.
-1 - Illegal MODBUS connection ID.

Declaration:

```
FUNCTION modasciiClose;
VAR_INPUT
   net_id  : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   switch : BOOL R_EDGE;
END_VAR;

VAR
   mb_net : INT;
   mode   : SINT;
END_VAR;

PROGRAM ModbusExample;

   mb_net := modasciiOpen(mode := mode, unit_id := 1, port := 2, baud :=
57600, parity := 0);

BEGIN
   IF switch THEN
      // Close MODBUS net
      modasciiClose(net_id := mb_net);

      // Open new MODBUS net
      IF mode = 0 THEN
         mode := 1;
      ELSE
         mode := 0;
      END_IF;
      mb_net := modasciiOpen(mode := mode, unit_id := 1, port := 2, baud :=
57600, parity := 0);
   END_IF;
END;
END_PROGRAM;
```

4.2.28.19 modasciiReceive (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: All devices with RS485
Firmware version: 4.00 / 1.00.00

This function will read a package that has been received from a MODBUS ASCII connection. While it is possible to use modbusReceive to continuously poll for received packages, it is recommended to only poll after modasciiWaitData has detected that a package has been received. This function is non-blocking and will therefore return immediately with the result of the operation.

Note: if the received package is too large for the receive buffer, only the part of the message that can fit the buffer is actually returned.

Input:

net_id : INT
The ID of the MODBUS connection.

frame : PTR
Address of the buffer where the received package is stored.

maxsize : INT
Maximum size of the receive buffer.

Output:

unit_id : INT
The ID of the device on the MODBUS network who sent the package.
Note: if the MODBUS connection is opened in slave mode, this will be the address of the RTCU device.

size : INT
Number of bytes received.

status : INT
 0 - Success.
-1 - Not a valid MODBUS connection ID.
-3 - No data received.

Declaration:

```
FUNCTION_BLOCK modasciiReceive;
VAR_INPUT
   net_id  : INT;
   frame   : PTR;
   maxsize : INT;
END_VAR;
VAR_OUTPUT
   unit_id : INT;
   size    : INT;
   status  : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
```

```
VAR
   mbNet : INT;
   mbRcv : modasciiReceive;
END_VAR;

FUNCTION setINT;
VAR_INPUT
   adr : PTR;
   v   : INT;
END_VAR;
   memcpy(dst := adr + 1, src := ADDR(v)    , len := 1);
   memcpy(dst := adr    , src := ADDR(v) + 1, len := 1);
END_FUNCTION;

FUNCTION_BLOCK modbusReadDiscreteInputs;
VAR_INPUT
   net_id  : INT;
   unit_id : INT;
   index   : INT;
   length  : INT;
END_VAR;
VAR_OUTPUT
   status : INT;
   size   : INT;
   input  : ARRAY[1..32] OF BOOL;
END_VAR;
VAR
   rc, i  : INT;
   buf    : ARRAY[1..253] OF SINT;
   mask   : SINT;
END_VAR;

   // Check length
   IF length < 1 OR length > 32 THEN
      status := 1;
      RETURN;
   END_IF;
   IF index < 1 THEN
      status := 2;
      RETURN;
   END_IF;

   // Build command
   buf[1] := 16#02;
   setINT(adr := ADDR(buf[2]), v := index - 1);
   setINT(adr := ADDR(buf[4]), v := length);

   // Send command
   rc := modasciiSend(net_id := net_id, unit_id := unit_id, frame :=
ADDR(buf), size := 5);
   IF rc <> 0 THEN
      status := 3;
      RETURN;
   END_IF;

   // Wait for reply
   IF NOT modasciiWaitData(net_id := net_id, timeout := 5000) THEN
      status := 4;
      RETURN;
   END_IF;

   // Read reply
   mbRcv(net_id := net_id, frame := ADDR(buf), maxsize := SIZEOF(buf));
   IF mbRcv.status <> 0 THEN
```

```
      status := 5;
      RETURN;
   END_IF;
   IF mbRcv.unit_id <> unit_id THEN
      status := 6;
      RETURN;
   END_IF;
   IF buf[1] <> 16#02 THEN
      status := 7;
      RETURN;
   END_IF;

   // Parse reply
   mask := 1;
   FOR i := 1 TO length DO
      input[i] := BOOL(buf[3 + ((i - 1) / 8)] AND mask);
      mask := shl8(in := mask, n := 1);
      IF mask = 0 THEN mask := 1; END_IF;
   END_FOR;
   size   := length;
   status := 0;
END_FUNCTION_BLOCK;

PROGRAM ModbusExample;
VAR
   dinRead : modbusReadDiscreteInputs;
   linsec  : DINT;
   i       : INT;
END_VAR;

   DebugMsg(message := "Initializing...");
   mbNet := modasciiOpen(port := 2, baud := 19200, parity := 0, mode := 0);
   DebugFmt(message := "modbus...\1", v1 := mbNet);

BEGIN
   IF clockNow() > linsec THEN
      DebugMsg(message := "-----------------------------");
      DebugMsg(message := "Reading inputs:");
      dinRead(net_id := mbNet, unit_id := 9, index := 1, length := 4);
      DebugFmt(message := "-status=\1", v1 := dinRead.status);
      IF dinRead.status = 0 THEN
         FOR i := 1 TO dinRead.size DO
            DebugFmt(message := "-input[\1]=\2", v1 := i, v2 :=
INT(dinRead.input[i]));
         END_FOR;
      END_IF;
      // Set next reading
      linsec := clockNow() + 5;
   END_IF;
END;
END_PROGRAM;
```

4.2.28.20 modasciiSend (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with RS485
Firmware version: 4.00 / 1.00.00

This function will send a package to a device on a MODBUS ASCII connection.

Input:

net_id : INT
The ID of the MODBUS connection.

unit_id : INT
The address of the device on the MODBUS connection to receive the data.
Note: if the MODBUS connection is opened in slave mode, this parameter is ignored.

frame : PTR
Address of the buffer that contains the package to send.

size : INT
Number of bytes to send from the buffer.

Returns: INT

0 - Success.
-1 - Illegal MODBUS connection ID.
-3 - No data to send.

Declaration:

```
FUNCTION modasciiSend;
VAR_INPUT
   net_id  : INT;
   unit_id : INT;
   frame   : PTR;
   size    : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   mbNet : INT;
   mbRcv : modasciiReceive;
END_VAR;

FUNCTION setINT;
VAR_INPUT
   adr : PTR;
   v   : INT;
END_VAR;
   memcpy(dst := adr + 1, src := ADDR(v)    , len := 1);
   memcpy(dst := adr    , src := ADDR(v) + 1, len := 1);
END_FUNCTION;

FUNCTION_BLOCK modbusReadDiscreteInputs;
VAR_INPUT
   net_id  : INT;
   unit_id : INT;
   index   : INT;
   length  : INT;
END_VAR;
VAR_OUTPUT
   status : INT;
   size   : INT;
   input  : ARRAY[1..32] OF BOOL;
END_VAR;
VAR
   rc, i  : INT;
   buf    : ARRAY[1..253] OF SINT;
```

```
   mask   : SINT;
END_VAR;

   // Check length
   IF length < 1 OR length > 32 THEN
      status := 1;
      RETURN;
   END_IF;
   IF index < 1 THEN
      status := 2;
      RETURN;
   END_IF;

   // Build command
   buf[1] := 16#02;
   setINT(adr := ADDR(buf[2]), v := index - 1);
   setINT(adr := ADDR(buf[4]), v := length);

   // Send command
   rc := modasciiSend(net_id := net_id, unit_id := unit_id, frame :=
ADDR(buf), size := 5);
   IF rc <> 0 THEN
      status := 3;
      RETURN;
   END_IF;

   // Wait for reply
   IF NOT modasciiWaitData(net_id := net_id, timeout := 5000) THEN
      status := 4;
      RETURN;
   END_IF;

   // Read reply
   mbRcv(net_id := net_id, frame := ADDR(buf), maxsize := SIZEOF(buf));
   IF mbRcv.status <> 0 THEN
      status := 5;
      RETURN;
   END_IF;
   IF mbRcv.unit_id <> unit_id THEN
      status := 6;
      RETURN;
   END_IF;
   IF buf[1] <> 16#02 THEN
      status := 7;
      RETURN;
   END_IF;

   // Parse reply
   mask := 1;
   FOR i := 1 TO length DO
      input[i] := BOOL(buf[3 + ((i - 1) / 8)] AND mask);
      mask := shl8(in := mask, n := 1);
      IF mask = 0 THEN mask := 1; END_IF;
   END_FOR;
   size   := length;
   status := 0;
END_FUNCTION_BLOCK;

PROGRAM ModbusExample;
VAR
   dinRead : modbusReadDiscreteInputs;
   linsec  : DINT;
   i       : INT;
END_VAR;
```

```
   DebugMsg(message := "Initializing...");
   mbNet := modasciiOpen(port := 2, baud := 19200, parity := 0, mode := 0);
   DebugFmt(message := "modbus...\1", v1 := mbNet);

BEGIN
   IF clockNow() > linsec THEN
      DebugMsg(message := "-----------------------------");
      DebugMsg(message := "Reading inputs:");
      dinRead(net_id := mbNet, unit_id := 9, index := 1, length := 4);
      DebugFmt(message := "-status=\1", v1 := dinRead.status);
      IF dinRead.status = 0 THEN
         FOR i := 1 TO dinRead.size DO
            DebugFmt(message := "-input[\1]=\2", v1 := i, v2 :=
INT(dinRead.input[i]));
         END_FOR;
      END_IF;
      // Set next reading
      linsec := clockNow() + 5;
   END_IF;
END;
END_PROGRAM;
```

4.2.28.21 modasciiWaitData (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with RS485
Firmware version: 4.00 / 1.00.00

This function will wait until a data package is received from a MODBUS ASCII connection - or for an optional timeout.
This function only indicates that the data is received. To actually read the data, modasciiReceive must be used.

Note: waiting for a package by using this function will block the calling thread.

Input:

net_id : INT
The ID of the MODBUS connection.

timeout : INT (-1,0..32000) *(default -1)*
Timeout period in milliseconds to wait. The resolution is 25 ms.
0 - Disable timeout. The function will return immediately.
-1 - Wait forever. The function will only return when data is received.

Returns: BOOL

TRUE: If data is ready to be read.
FALSE: If no data is present.

Declaration:

```
FUNCTION modasciiWaitData : BOOL;
VAR_INPUT
   net_id  : INT;
   timeout : INT := -1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   mbNet : INT;
   mbRcv : modasciiReceive;
END_VAR;

FUNCTION setINT;
VAR_INPUT
   adr : PTR;
   v   : INT;
END_VAR;
   memcpy(dst := adr + 1, src := ADDR(v)    , len := 1);
   memcpy(dst := adr    , src := ADDR(v) + 1, len := 1);
END_FUNCTION;

FUNCTION_BLOCK modbusReadDiscreteInputs;
VAR_INPUT
   net_id  : INT;
   unit_id : INT;
   index   : INT;
   length  : INT;
END_VAR;
VAR_OUTPUT
   status : INT;
   size   : INT;
   input  : ARRAY[1..32] OF BOOL;
END_VAR;
VAR
   rc, i  : INT;
   buf    : ARRAY[1..253] OF SINT;
   mask   : SINT;
END_VAR;

   // Check length
   IF length < 1 OR length > 32 THEN
      status := 1;
      RETURN;
   END_IF;
   IF index < 1 THEN
      status := 2;
      RETURN;
   END_IF;

   // Build command
   buf[1] := 16#02;
   setINT(adr := ADDR(buf[2]), v := index - 1);
   setINT(adr := ADDR(buf[4]), v := length);

   // Send command
   rc := modasciiSend(net_id := net_id, unit_id := unit_id, frame :=
ADDR(buf), size := 5);
   IF rc <> 0 THEN
      status := 3;
      RETURN;
   END_IF;

   // Wait for reply
   IF NOT modasciiWaitData(net_id := net_id, timeout := 5000) THEN
      status := 4;
      RETURN;
   END_IF;
```

```
   // Read reply
   mbRcv(net_id := net_id, frame := ADDR(buf), maxsize := SIZEOF(buf));
   IF mbRcv.status <> 0 THEN
      status := 5;
      RETURN;
   END_IF;
   IF mbRcv.unit_id <> unit_id THEN
      status := 6;
      RETURN;
   END_IF;
   IF buf[1] <> 16#02 THEN
      status := 7;
      RETURN;
   END_IF;

   // Parse reply
   mask := 1;
   FOR i := 1 TO length DO
      input[i] := BOOL(buf[3 + ((i - 1) / 8)] AND mask);
      mask := shl8(in := mask, n := 1);
      IF mask = 0 THEN mask := 1; END_IF;
   END_FOR;
   size   := length;
   status := 0;
END_FUNCTION_BLOCK;

PROGRAM ModbusExample;
VAR
   dinRead : modbusReadDiscreteInputs;
   linsec  : DINT;
   i       : INT;
END_VAR;

   DebugMsg(message := "Initializing...");
   mbNet := modasciiOpen(port := 2, baud := 19200, parity := 0, mode := 0);
   DebugFmt(message := "modbus...\1", v1 := mbNet);

BEGIN
   IF clockNow() > linsec THEN
      DebugMsg(message := "------------------------------");
      DebugMsg(message := "Reading inputs:");
      dinRead(net_id := mbNet, unit_id := 9, index := 1, length := 4);
      DebugFmt(message := "-status=\1", v1 := dinRead.status);
      IF dinRead.status = 0 THEN
         FOR i := 1 TO dinRead.size DO
            DebugFmt(message := "-input[\1]=\2", v1 := i, v2 :=
INT(dinRead.input[i]));
         END_FOR;
      END_IF;
      // Set next reading
      linsec := clockNow() + 5;
   END_IF;
END;
END_PROGRAM;
```

4.2.29 MQTT: Functions for MQTT

4.2.29.1 MQTT: MQTT communication

The MQTT functions implement the Message Queue Telemetry Transport (MQTT) protocol. MQTT is a open message protocol for M2M communications that enables transfer of telemetry-style data in the form of messages.

The MQTT protocol is based on the principle of publishing messages and subscribing to topics, or "pub/sub". Multiple clients connect to a server and subscribe to topics that they are interested in. Clients also connect to the server and publish messages to topics. Many clients may subscribe to the same topics and do with the information as they please. The server and MQTT act as a simple, common interface for everything to connect to.

MQTT has been characterized as the protocol realizing the Internet of Things the same was as HTTP was the enabler for the Internet of People.

Please visit www.mqtt.org detailed information on the MQTT protocol.

• mqttOpen	Opens a connection to a MQTT server.
• mqttClose	Closes a connection to a MQTT server.
• mqttConnected	Queries if a connection to a MQTT server is active.
• mqttConnectionLoss	Queries if a connection to a MQTT server has been interrupted.
• mqttStatus	Get the status of a connection to a MQTT server.
• mqttPublish	Sends a message to a MQTT server.
• mqttPendingCount	Returns the number of messages, waiting to be sent.
• mqttPendingClear	Clears the list of pending messages.
• mqttSubscribe	Subscribe to a topic on a MQSS server.
• mqttUnsubscribe	Unsubscribe to a topic on a MQTT server.
• mqttWaiEvent	Wait for a received message or an optional timeout.
• mqttReceive	Read a received message.
• mqttCredentialsSet	Change the credentials used to connect to a MQTT server.

Unicode
Support for Unicode strings are included in the MQTT API, which means that text (topics, username, etc.) can be in the local language rather than in English. For more information, please refer to mqttOpen.
Unicode strings uses UTF-8 encoding.

Topic
Topics are treated as a hierarchy, using slash (/) as a separator. This allows sensible arrangement of common themes to be created, much in the same way as a file system.
For example, multiple RTCU devices may publish their temperature information on the following topic:

RTCU/222135115/temperature

where 222135115 is the serial number of the RTCU device.

Quality of Service (QoS)
The MQTT protocol supports 3 levels of QoS:

qos	name	description
0	Fire and forget	The message may not be delivered.
1	At least once	The message will be delivered, but may be delivered more than once in some circumstances.
2	Once and only	The message will be delivered exactly once.

Last Will and Testament (LWT)
When the RTCU connects to a server, it may inform the server that it has a will.
This is a message that it wishes the server to send when it disconnects unexpectedly.

Limitations
- Up-to 5 simultaneous MQTT sessions are supported under the NX32L architecture.
- Up to 255 (NX32L) / 10 (X32/NX32) received messages are buffered for removal by the application.
- Maximum of 6 (NX32L) / 3 (X32/NX32) messages inflight from the server.This is the maximum number of QoS 2 messages that are not fully acknowledged.
- Maximum size of message:
  - X32 architecture: 1024 bytes,
  - NX32 architecture: 4096 bytes.
  - NX32L architecture: 16384 bytes.

4.2.29.2 mqttOpen (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 3.12 / 1.00.00

This function will initiate a connection to a MQTT server. When this function returns, the connection may not be finally established. The establishing of the connection must be determined by using the mqttConnected() function.
The function must return a valid handle before any other MQTT functions can be called. The handle will no longer be valid after calling the mqttClose() function, and can no longer be used with MQTT functions.
A valid handle is an integer value in the range 1 and 32767.

Prior to communicating with the server, the network interface connection must be established. For example by using gsmPower and gprsOpen for Mobile network, or netOpen for LAN network.

When the MQTT connection is opened, there is no need for further intervention from the VPL program to keep the connection up. If, for some reason, the MQTT connection terminates, the firmware will automatically reconnect and establish the MQTT connection again.

On NX32L architecture devices up-to 5 simultaneous connections can be established. Other devices only supports a single connection.

Using secure connection
For NX32L devices a secure TLS (TLS v1.2, v1.1 or 1.0) connection can be established assuming that a matching X509 certificate(s) is present.

To use server verification:
1. Ensure the root certificate needed to verify the MQTT server is present.

2. Set 'useTLS := TRUE' and 'port' according to server configuration (standard for secure connections is 8883) when calling mqttOpen.

If client verification is required by server then:
1. Ensure the client certificate needed to verify the device is present.
2. Set 'tls_certificate' and 'tls_password' according to the installed client certificate when calling mqttOpen.

Using Unicode strings

Support for Unicode strings are included in the MQTT API so that text (topics, username, etc.) can be in the local language rather than in English.
The problem with Unicode strings is that only the navigation API and the MQTT API supports Unicode directly.
While it is not possible to enter Unicode text directly into a string variable, it can be done as a string of special characters (see STRING).
This approach requires 3 steps for each character:
1. Find the character at the Unicode homepage (www.unicode.org).
2. Encode the character into UTF-8 (Wikipedia gives a good description of this here: en.wikipedia.org/wiki/Utf-8).
3. Type in the special characters from step 2 (e.g. "CF86" for the Greek character "phi").

Input:

ip : STRING
The IP address of the MQTT server.

port : DINT (*default 1883*)
The port where the MQTT server listens for clients

iface : SINT *(default 0)*
The network interface to use. 0 = Default network, 1 = Mobile network, 2 = LAN network, etc. (See Network)
Note: For backwards compatibility the Mobile network is used as default network.

username : STRING
The user name required to connect to the server.

password : STRING
The password required to connect to the server.

clientid : STRING
The id to use when connecting to the server.

keepalive : INT (30..30000*, default 600*)
Frequency for sending a ping to the server in seconds.

clean : BOOL
If TRUE the server will clear old information about client, if FALSE the old information is kept.

lwt_topic : STRING
The topic where the Last Will and Testament (LWT) is published.
Leave this empty to disable LWT.

lwt_message : STRING
The message that is published with the LWT.
Minimum length is 1 character if lwt_topic is not empty.

lwt_retained : BOOL
If TRUE the LWT will be retained on the broker, if FALSE it is not.

lwt_qos : SINT (0..2)
The QoS used to publish the LWT.

useTLS : BOOL
TRUE: Use TLS to create a secure connection.
FALSE: Use an insecure connection

tls_certificate : STRING
The client certificate if required by the broker.

tls_password : STRING
The password for the client certificate.

Returns: INT

Handle of the MQTT connection.
-1 - Too many MQTT connections.
-2 - Invalid parameter.
-3 - Memory allocation error
-5 - Secure connections are not supported.

Declaration:

```
FUNCTION mqttOpen : INT;
VAR_INPUT
   ip              : STRING;
   port            : DINT;
   iface           : SINT;
   username        : STRING;
   password        : STRING;
   clientid        : STRING;
   keepalive       : INT := 600;
   clean           : BOOL;
   lwt_topic       : STRING;
   lwt_message     : STRING;
   lwt_retained    : BOOL;
   lwt_qos         : SINT;
   useTLS          : BOOL;
   tls_certificate : STRING;
   tls_password    : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   mqtt : INT;
   text : STRING;
END_VAR;

PROGRAM example;

   gsmPower(power := ON);
   gprsOpen();

BEGIN
```

```
   ...
   // Connect to MQTT server
   IF mqtt < 1 AND gprsConnected() THEN
      text := strFormat(format := "RTCU_\4", v4 := boardSerialNumber());
      mqtt := mqttOpen(ip := "test.mosquitto.org", port := 8883, clientid :=
text, useTLS := TRUE);
      IF mqtt < 0 THEN
         DebugFmt(message := "Could not connect to MQTT server (errno=\1)", v1
:= mqtt);
      ELSE
         DebugMsg(message := "Connection opened");
      END_IF;
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.29.3 mqttClose (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 3.12 / 1.00.00

This function will close a connection to a MQTT server. After this function is called the handle is invalid and can no longer be used.
Please also see mqttOpen.

Input:

handle : INT
The handle to the MQTT connection.

Returns: INT

0 - Success,
1 - Invalid handle.
2 - Connecton is busy.

Declaration:

```
FUNCTION mqttClose;
VAR_INPUT
   handle : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   mqtt      : INT;
END_VAR;

PROGRAM example;

   gsmPower(power := ON);
   gprsOpen();

BEGIN
   mqtt := mqttOpen(ip := "test.mosquitto.org", port := 1883, clientid :=
```

```
"Example");
   ...
   // Disconnect from MQTT server
   mqttClose(handle := mqtt);
   ...
END;
END_PROGRAM;
```

4.2.29.4 mqttConnected (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 3.12 / 1.00.00

This returns information about whether a MQTT connection is established.

Input:

handle : INT
The handle to the MQTT connection.

Returns: BOOL

TRUE if MQTT connection is established, FALSE if not.

Declaration:

```
FUNCTION mqttConnected : BOOL;
VAR_INPUT
   handle : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   mqtt : INT;
END_VAR;

PROGRAM example;

BEGIN
   ...
   // Check connection
   IF mqttConnected(handle := mqtt) THEN
      DebugMsg(message := "Connected to MQTT");
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.29.5 mqttConnectionLoss (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 5.12 / 1.64.00

This returns information about whether a MQTT connection has been interrupted.

The mqttConnectionLoss function will return whether an interruption has been detected since last call.

Input:
handle : INT
The handle to the MQTT connection.

Returns: BOOL
TRUE if the connection has been interrupted, FALSE if not.

Declaration:
```
FUNCTION mqttConnectionLoss : BOOL;
VAR_INPUT
   handle : INT;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

VAR
   mqtt : INT;
END_VAR;

PROGRAM example;

BEGIN
   ...
   // Check connection
   IF mqttConnectionLoss(handle := mqtt) THEN
      DebugMsg(message := "Connection to MQTT server has been interrupted");
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.29.6 mqttPublish (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 3.12 / 1.00.00

This function will publish a message to a MQTT server.
The function will not block the MQTT interface while waiting for confirmation when using QoS 1 or 2.

The message will only be sent if connected to the MQTT server.
If the connection is lost before the function returns, the message will be sent when the connection is re-established, unless mqttPendingClear is called.

Input:
handle : INT
The handle to the MQTT connection.

topic : STRING
The topic of the message.

qos : SINT (0..2)
The QoS to publish the message with.

retained : *BOOL*
If TRUE the server should retain a copy of the message, if FALSE the message is not retained. Retained messages will be transmitted to new subscribers of the topic by the server.

data : PTR
Address to the buffer with the contents of the message.

size : INT
The number of bytes to send.
Please note that the maximum size of the message depends on the architecture of the device:
- o X32 architecture: 1024 bytes,
- o NX32 architecture: 4096 bytes.
- o NX32L architecture: 16384 bytes.

Returns: INT

0 - Message was sent.
1 - Invalid handle.
2 - Not connected to the MQTT server.
3 - Invalid parameter.
4 - A publish is already in progress.
5 - Message transmission interrupted due to connection loss. The message will be transmitted again when connection is reestablished.

Declaration:

```
FUNCTION mqttPublish : INT;
VAR_INPUT
   handle   : INT;
   topic    : STRING;
   qos      : SINT;
   retained : BOOL;
   data     : PTR;
   size     : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   mqtt  : INT;
   text  : STRING;
   topic : STRING;
   time  : TON;
   temp  : DINT;
   rc    : INT;
   buf   : ARRAY [1..15] OF SINT;
END_VAR;

PROGRAM example;
   DebugMsg(message := "Initialize GPRS...");
   gsmPower(power := ON);
   gprsOpen();
   WHILE NOT gprsConnected() DO
      DebugMsg(message := "Waiting for cellular connection...");
```

```
      Sleep(delay := 2500);
   END_WHILE;
   DebugMsg(message := "Initialize MQTT...");
   text  := strFormat(format := "RTCU_\4", v4 := boardSerialNumber());
   topic := strFormat(format := "RTCU/\4/temperature", v4 :=
boardSerialNumber());
   mqtt  := mqttOpen(ip := "test.mosquitto.org", port := 1883, clientid :=
text);
   DebugFmt(message := "MQTT handle= \1", v1 := mqtt);

   IF NOT mqttConnected(handle := mqtt) THEN
      rc := mqttStatus(handle := mqtt);
      DebugFmt(message := "Connection failed, Status=\1", v1 := rc);
   END_IF;

   time(trig := OFF, pt := 60000);

BEGIN
   time(trig := ON);
   IF time.q THEN
      temp := boardTemperature();
      text := dintToStr(v := temp / 100) + "." + dintToStr(v := abs(v := temp
MOD 100));
      strToMemory(dst := ADDR(buf), str := text, len := strLen(str := text));
      rc := mqttPublish(
                        handle   := mqtt,
                        topic    := topic,
                        qos      := 1,
                        retained := FALSE,
                        data     := ADDR(buf),
                        size     := strLen(str := text)
                       );
      DebugFmt(message := "Published= \1", v1 := rc);
      time(trig := OFF);
   END_IF;
END;
END_PROGRAM;
```

4.2.29.7 mqttPendingCount (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 4.10 / 1.00.00

This function will return the number of messages that has not been sent yet.
This can be used to verify that all messages have been sent.

Input:

handle : INT
The handle to the MQTT connection.

Returns: SINT

The number of pending messages.
-1 - Invalid handle

Declaration:

```
FUNCTION mqttPendingCount : SINT;
VAR_INPUT
   handle   : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   mqtt  : INT;
END_VAR;

PROGRAM test;

BEGIN
   ...
   // Clear any pending messages
   IF mqttPendingCount(handle:=mqtt) > 0 THEN
      mqttPendingClear(handle:=mqtt);
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.29.8 mqttPendingClear (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 4.10 / 1.00.00

This function will clear the queue of pending messages.

Input:

handle : INT
The handle to the MQTT connection.

Returns: SINT

1 - Invalid MQTT connection.
0 - Pending messages cleared.

Declaration:

```
FUNCTION mqttPendingClear : SINT;
VAR_INPUT
   handle   : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   mqtt  : INT;
END_VAR;

PROGRAM test;

BEGIN
   ...
   // Clear any pending messages
   IF mqttPendingCount(handle:=mqtt) > 0 THEN
      mqttPendingClear(handle:=mqtt);
   END_IF;
   ...
END;
```

```
END_PROGRAM;
```

4.2.29.9 mqttSubscribe (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 3.12 / 1.00.00

This function will register a subscription to a topic on a MQTT server.

The topic can contain wild-cards (+ and #).
The + is used as a wild-card for a single level of hierarchy.
For example, a topic of "a/b/+/c" will match messages with topic "a/b/a/c" and "a/b/b/c", but not "a/b/c" or "a/b/a/c/d"

The # is used as a wild-card for all remaining levels of hierarchy.
For example, a topic of "a/b/#" will match messages with topic "a/b/c" and "a/b/c/d", but not "a/b".

Note:
There is no information stored in the RTCU about the subscriptions made. The information persistence is defined by the implementation behavior defined by the MQTT server.

Input:

handle : INT
The handle to the MQTT connection.

topic : STRING
The topic to subscribe to.

qos : SINT (0..2)
The QoS used by the MQTT server to transfer message to the device.

Returns: INT

0 - Subscription is registered.
1 - Invalid handle.
2 - Not connected to the MQTT server.
3 - Invalid parameter.

Declaration:

```
FUNCTION mqttSubscribe : INT;
VAR_INPUT
   handle : INT;
   topic  : STRING;
   qos    : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   mqtt : INT;
   text : STRING;
   rxd  : mqttReceive;
   rc   : INT;
   msg  : ARRAY [1..50] OF SINT;
END_VAR;
```

```
PROGRAM example;

   DebugMsg(message := "Initialize GPRS...");
   gsmPower(power := ON);
   gprsOpen();
   WHILE NOT gprsConnected() DO
      DebugMsg(message := "Waiting for cellular connection...");
      Sleep(delay := 2500);
   END_WHILE;
   DebugMsg(message := "Initialize MQTT...");
   text := strFormat(format := "RTCU_\4", v4 := boardSerialNumber());
   mqtt := mqttOpen(ip := "test.mosquitto.org", port := 1883, clientid :=
text);
   DebugFmt(message := "MQTT handle= \1", v1 := mqtt);

   IF NOT mqttConnected(handle := mqtt) THEN
      rc := mqttStatus(handle := mqtt);
      DebugFmt(message := "Connection failed, Status=\1", v1 := rc);
   END_IF;

   rc := mqttSubscribe(handle := mqtt, qos := 2, topic := "RTCU/
+/temperature");
   DebugFmt(message := "Subscribed= \1", v1 := rc);
   rxd(data := ADDR(msg), maxsize := SIZEOF(msg));

BEGIN
   rc := mqttWaitEvent(timeout := 5);
   IF rc > 0 THEN
      DebugMsg(message := "Message received");
      rxd();
      DebugMsg(message := "-Topic= " + rxd.topic);
      DebugFmt(message := "-Handle= \1", v1 := rxd.handle);
      DebugFmt(message := "-Size= \1", v1 := rxd.size);
      DebugMsg(message := "-Temperature= " + strFromMemory(src := ADDR(msg),
len := rxd.size));
   END_IF;
END;
END_PROGRAM;
```

4.2.29.10 mqttUnsubscribe (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 3.12 / 1.00.00

This function will remove a subscription from a MQTT server.

Input:

handle : INT
The handle to the MQTT connection.

topic : STRING
The topic of the subscription.

Returns: INT

0 - Subscription is removed.
1 - Invalid handle.

2 - Not connected to the MQTT server.
3 - Invalid parameter.

Declaration:

```
FUNCTION mqttUnsubscribe : INT;
VAR_INPUT
   handle : INT;
   topic  : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   mqtt : INT;
   rc   : INT;
END_VAR;

PROGRAM example;
BEGIN
   ...
   rc := mqttUnsubscribe(handle := mqtt, topic := "RTCU/+/temperature");
   DebugFmt(message := "Unsubscribed= \1", v1 := rc);
   ...
END;
END_PROGRAM;
```

4.2.29.11 mqttWaitEvent (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 3.12 / 1.00.00

This function will block the thread while waiting for a message to be received, with an optional timeout.
Once a message is received, mqttWaitEvent will continue to report this until it has been read using the mqttReceive function block.

Input:

timeout : INT
Timeout period in seconds to wait.
0 - Disable timeout. The function will return immediately.
-1 - Wait forever. The function will only return when a message is received.

Returns: INT

1 - A message is received.
0 - Timeout.
-1 - No MQTT connections.

Declaration:

```
FUNCTION mqttWaitEvent : INT;
VAR_INPUT
   timeout : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   mqtt : INT;
   text : STRING;
   rxd  : mqttReceive;
   rc   : INT;
   msg  : ARRAY [1..50] OF SINT;
END_VAR;

PROGRAM example;
   DebugMsg(message := "Initialize GPRS...");
   gsmPower(power := ON);
   gprsOpen();
   WHILE NOT gprsConnected() DO
      DebugMsg(message := "Waiting for cellular connection...");
      Sleep(delay := 2500);
   END_WHILE;
   DebugMsg(message := "Initialize MQTT...");
   text := strFormat(format := "RTCU_\4", v4 := boardSerialNumber());
   mqtt := mqttOpen(ip := "test.mosquitto.org", port := 1883, clientid :=
text);
   DebugFmt(message := "MQTT handle= \1", v1 := mqtt);

   IF NOT mqttConnected(handle := mqtt) THEN
      rc := mqttStatus(handle := mqtt);
      DebugFmt(message := "Connection failed, Status=\1", v1 := rc);
   END_IF;

   rc := mqttSubscribe(handle := mqtt, qos := 2, topic := "RTCU/
+/temperature");
   DebugFmt(message := "Subscribed= \1", v1 := rc);
   rxd(data := ADDR(msg), maxsize := SIZEOF(msg));

BEGIN
   rc := mqttWaitEvent(timeout := 5);
   IF rc > 0 THEN
      DebugMsg(message := "Message received");
      rxd();
      DebugMsg(message := "-Topic= " + rxd.topic);
      DebugFmt(message := "-Handle= \1", v1 := rxd.handle);
      DebugFmt(message := "-Size= \1", v1 := rxd.size);
      DebugMsg(message := "-Temperature= " + strFromMemory(src := ADDR(msg),
len := rxd.size));
   END_IF;
END;
END_PROGRAM;
```

4.2.29.12 mqttReceive (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 3.12 / 1.00.00

This function will return a received message.

Note that the message is only valid after a "message received" event is raised; the event will be removed when this function block is called.
The RTCU will buffer all valid incoming messages in an internal buffer with room for 10 messages - further messages will be lost.

Input:

data : PTR
Address of the buffer where the data of the message is stored in.

maxsize : INT
The maximum number of bytes that can be stored in the buffer. Any message data after this limit will be lost.
Please note that the firmware is only capable of handling messages with up to a device specific limit, and will truncate the data if a message is received with more data.
The limit depends on the architecture of the device:

- X32 architecture: 1024 bytes,
- NX32 architecture: 4096 bytes.
- NX32L architecture: 16384 bytes.

Output:

ready : BOOL
True if a message has been received.

handle : INT
The handle of the MQTT connection the message is received from.

topic : STRING
The topic of the received message.

size : INT
The actual number of bytes stored in the buffer.

retained : BOOL
If TRUE the message is not a new one, but one that has been retained by the MQTT server, if FALSE the message is a new one.

duplicate : BOOL
If TRUE the message is a duplicate, if FALSE is is not a duplicate.

Declaration:

```
FUNCTION_BLOCK mqttReceive;
VAR_INPUT
   data      : PTR;
   maxsize   : INT;
END_VAR;
VAR_OUTPUT
   ready     : BOOL;
   handle    : INT;
   topic     : STRING;
   size      : INT;
   retained  : BOOL;
   duplicate : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   mqtt : INT;
   text : STRING;
   rxd  : mqttReceive;
   rc   : INT;
```

```
   msg  : ARRAY [1..50] OF SINT;
END_VAR;

PROGRAM example;

   DebugMsg(message := "Initialize GPRS...");
   gsmPower(power := ON);
   gprsOpen();
   WHILE NOT gprsConnected() DO
      DebugMsg(message := "Waiting for cellular connection...");
      Sleep(delay := 2500);
   END_WHILE;
   DebugMsg(message := "Initialize MQTT...");
   text := strFormat(format := "RTCU_\4", v4 := boardSerialNumber());
   mqtt := mqttOpen(ip := "test.mosquitto.org", port := 1883, clientid :=
text);
   DebugFmt(message := "MQTT handle= \1", v1 := mqtt);

   IF NOT mqttConnected(handle := mqtt) THEN
      rc := mqttStatus(handle := mqtt);
      DebugFmt(message := "Connection failed, Status=\1", v1 := rc);
   END_IF;

   rc := mqttSubscribe(handle := mqtt, qos := 2, topic := "RTCU/
+/temperature");
   DebugFmt(message := "Subscribed= \1", v1 := rc);
   rxd(data := ADDR(msg), maxsize := SIZEOF(msg));

BEGIN
   rc := mqttWaitEvent(timeout := 5);
   IF rc > 0 THEN
      DebugMsg(message := "Message received");
      rxd();
      DebugMsg(message := "-Topic= " + rxd.topic);
      DebugFmt(message := "-Handle= \1", v1 := rxd.handle);
      DebugFmt(message := "-Size= \1", v1 := rxd.size);
      DebugMsg(message := "-Temperature= " + strFromMemory(src := ADDR(msg),
len := rxd.size));
   END_IF;
END;
END_PROGRAM;
```

4.2.29.13 mqttStatus (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 3.13 / 1.00.00

This returns the status of a MQTT connection.

Input:

handle : INT
The handle to the MQTT connection.

Returns: **SINT**

0 - Connected to the MQTT server.
1 - Invalid MQTT connection.
2 - Not connected to MQTT server.
3 - MQTT server not found.

4 - No reply from MQTT server
5 - Connection rejected, unacceptable protocol version.
6 - Connection rejected, client ID rejected.
7 - Connection rejected, server unavailable.
8 - Connection rejected, bad user-name or password.
9 - Connection rejected, not authorized.
20 - Secure connection failed.
21 - Secure connection require client certificate.
22 - Certificate verification failed.
23 - Client certificate is incomplete. (Encryption key or client certificate is missing)

Declaration:

```
FUNCTION mqttStatus : SINT;
VAR_INPUT
   handle : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   mqtt : INT;
   text : STRING;
   rc   : SINT;
END_VAR;

PROGRAM example;

   gsmPower(power := ON);
   gprsOpen();

BEGIN
   ...
   // Connect to MQTT server
   IF mqtt < 1 AND gprsConnected() THEN
      text := strFormat(format := "RTCU_\4", v4 := boardSerialNumber());
      mqtt := mqttOpen(ip := "test.mosquitto.org", port := 8883, clientid :=
text, useTLS := TRUE);
      IF NOT mqttConnected(handle := mqtt) THEN
         rc := mqttStatus(handle := mqtt);
         DebugFmt(message := "Status=\1", v1 := rc);
      END_IF;
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.29.14 mqttCredentialsSet (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.70.00

This function will change the credentials used to connect to the MQTT server.

Some MQTT servers require that the credentials used to authenticate the client must be changed every time the client connects.

By calling qmttCredentialsSet, this can be done without having to close the connection and then opening a new connection with the changed credentials.

Input:

handle : INT
The handle to the MQTT connection.

username : STRING
The new user name to use when connecting to the server.
The username will not be changed if empty

password : STRING
The new password to use when connecting to the server.
Setting the password requires that a username is present, either in this function or in mqttOpen.
The password will not be changed if empty

tls_certificate : STRING
The new client certificate to use when connecting to the server.
The tls_certificate will not be changed if empty

tls_password : STRING
The new password for the client certificate.
The tls_password will not be changed if empty

reconnect : BOOL (*Default FALSE*)
TRUE: The connection to the MQTT server will be disconnected and then reconnected immediately.
FALSE: The new credentials will take effect the next time the MQTT server connection is lost.

Returns: INT

0 - Subscription is registered.
1 - Invalid handle.
2 - Invalid parameter.

Declaration:

```
FUNCTION mqttCredentialsSet : INT;
VAR_INPUT
   handle          : INT;
   username        : STRING;
   password        : STRING;
   tls_certificate : STRING;
   tls_password    : STRING;
   reconnect       : BOOL := FALSE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   mqtt : INT;
   text : STRING;
   rxd  : mqttReceive;
   rc   : INT;
   msg  : ARRAY [1..50] OF SINT;
   time : TON;
END_VAR;
```

```
PROGRAM example;

   DebugMsg(message := "Initialize network...");
   netOpen(iface := 2);
   WHILE NOT netConnected() DO
      DebugMsg(message := "Waiting for network connection...");
      Sleep(delay := 2500);
   END_WHILE;
   DebugMsg(message := "Initialize MQTT...");
   text := strFormat(format := "RTCU_\4", v4 := boardSerialNumber());
   mqtt := mqttOpen(
                    ip       := "test.mosquitto.org",
                    port     := 1883,
                    iface    := 2,
                    clientid := text,
                    username := "user",
                    password := "Password 1"
                   );
   DebugFmt(message := "MQTT handle= \1", v1 := mqtt);

   // Update timer
   time(trig := ON, pt := 3600000);

BEGIN
   ...
   time(trig := ON);
   IF time.q THEN
      // Reconnect
      rc := mqttCredentialsSet(handle := mqtt, password := "Password 2",
reconnect := TRUE);
      DebugFmt(message := "mqttCredentialsSet= \1", v1 := rc);
      time(trig := OFF);
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.30 ms: Motion Sensor

4.2.30.1 ms: Motion Sensor

The latest generation of automotive NX32L devices offers a powerful set of inertial sensors that can be used for applications, such as driving behavior analysis, activity monitoring, indoor positioning, virtual ignition, and crash detection.
By combining the inertial sensors with the GNSS sensor enhanced navigation and dead reckoning applications can be implemented.

The full 3-axis accelerometer, gyroscope, and magnetometer are tightly integrated into the RTCU M2M Platform with a uniform motion sensor interface and full support for power management for the most demanding applications.

The motion sensor interface gives the basic tools needed to use the embedded sensors and can thus be used to develop sensor fusion applications.

Sensors

With this interface comes access to simultaneous sets of accelerometer, gyroscope and magnetometer measurements.

Accelerometer:
The accelerometer measures accelerations. This is useful to measure changes in velocity and changes in position (by integrating the signal).
The interface delivers accelerometer data in **units of gravity G** (SI standard: 1.0 m/s2 [meters per second squared]), with a maximum range of [-16.0G : +16.0G]. The accelerometer measurements represents the resulting forces acting on the device. Static forces as the gravity pull can be observed, but it is also possible to get the relative measurements, thus representing the actual changes in velocity of the device in any given direction.

Gyroscope:
The gyroscope measures changes in the orientation.
The interface delivers gyroscope data in **units of degrees per second** (dps), with a maximum range of [-2000.0dps : +2000.0dps]. The gyroscope data represents the rotational changes of the device in any given direction. With no movement of the device, the gyroscope ideally measures zero on all axis. With calibration from knowing the direction of gravity with the accelerometer, it is thus possible to know the orientation of the device in 3-dimensional space (yaw, pitch and roll) with high precision.

Magnetometer:
The magnetometer measures magnetic fields. Because the earth has a significant magnetic field, the magnetometer can be used as a compass. The interface delivers magnetometer data in **units of gauss** (SI standard: 1/10000 Tesla), with a maximum range of [-50.0gauss : +50.0gauss]. The magnetometer data represents the magnetic impact on the device from the surrounding environment. It is thus possible to measure the magnetic north in an area with low magnetic disturbances. Radar pulses with a time frame greater than the sampling frequency will also be detectable.

Architecture of the Motion Sensor Interface

The Motion Sensor interface is based on an asynchronous event-based model, where solicited as well as unsolicited events from the sensors are automatically queued and delivered to the user application. The function msWaitEvent is used to return information about the various events arriving from the sensors. This model eliminates the requirement for the application to continuously poll for messages, and it also encourages use of multithreading with the added benefits that comes with this.

Events that are triggered by the application requesting various information from the sensors are therefore called *solicited* events, or alternatively they can be triggered by the sensors, and are therefore instead called *unsolicited* events. A solicited event is for example, when the application calls msLoggerStop and then msWaitEvent returns with event# 4. An unsolicited event works exactly the same way as a solicited event, except for the fact that msLoggerStop is no longer called by the application.
Please consult msWaitEvent for detailed information about the various events and how they are handled.

The following functions and structures are used to access the Motion Sensor interface:

Interface and setup

The following functions are used to access the interface and set global motion sensor settings.

- msOpen — Opens the interface for use and sets operation to default mode.
- msClose — Closes the interface and clears its wake-up sources.
- msConfig — Set data acquisition rate.

Data acquisition

The following functions and structures are used to start various sensors and get data.

- msAccEnable — Enable the accelerometer.
- msGyrEnable — Enable the gyroscope.
- msMagEnable — Enable the magnetometer.
- msDataSet — Structure to read sensor data with.
- msRead — Read a set of sensor data.

Data logger

The following functions and structures are used to log sensor data to buffers and read the data.

- msLoggerCreate — Create a logger.
- msLoggerDestroy — Remove a logger.
- msLoggerAddAcc — Add accelerometer data acquisition ability to a logger.
- msLoggerAddGyr — Add gyroscope data acquisition ability to a logger.
- msLoggerAddMag — Add magnetometer data acquisiotion ability to a logger.
- msLoggerStart — Start specified logger.
- msLoggerStop — Stop specified logger.
- msLoggerNext — Request which type of sensor data record is the next to read.
- msLoggerRead — Read a sensor data record from the logger buffer. A record from the oldest set is given.
- msLoggerLevel — Tells how full a given logger buffer is in promille.
- msTimestamp — Structure to read a time stamp data record with.
- msAccData — Structure to read an accelerometer data record with.
- msGyrData — Structure to read a gyroscope data record with.
- msMagData — Structure to read a magnetometer data record with.

Event handling

The following functions and structures and used to manage and read sensor and logging related events.

- msEventRegister — Register an event handler, consisting of an event configuration and a possible action.
- msEventUnregister — Unregister an event handler.
- msWaitEvent — Wait for the next event or a timeout.
- msReadEvent — Read event data according to last event.
- msEventLogWatermark — Structure to configure a logger watermark event.
- msEventLogFull — Structure to configure a logger full event.
- msEventShock — Structure to configure shock detection events.
- msEventAcceleration — Structure to configure acceleration detection events.
- msActionStopLogger — Structure to stop a logger on an event.
- msActionStartLogger — Structure to start a logger on an event.
- msActionSwitchLogger — Structure to switch active logger to another logger on an event.
- msReadWatermark — Structure to read returned watermark event data.
- msReadShockEvent — Structure to read returned shock event data.
- msReadAccelEvent — Structure to read returned acceleration event data.

Power management

The following functions are used to add wake-up sources to power management.

- msVibrationSetWakeup — Enable or disable the ability to wake from low power modes with device vibration. See pmSuspend.

Calculation functions

The following functions are used to perform calculations on the sensor data.

- msCompassHeading — Calculates a measure for magnetic north.
- msVectorToAngles — Calculates axis angles relative to acceleration.
- msVectorToForce — Calculates the total force of acceleration.

Limitations:
For the acceleration and shock events to be detected, the accelerometer must be enabled with logger functionality running. See example under msEventRegister.

Only one acceleration event handler configuration and one shock event handler configuration can be active at a time.
Only 8 loggers can be in the system at a time.

4.2.30.2 Interface and setup

4.2.30.2.1 msOpen (Function)

```
Architecture:       NX32L
Device support:     NX-200, LX2
Firmware version:   1.36.00
```

This function must be called to open the Motion Sensor interface, and is mutually exclusive to the accOpen function.

Returns: **INT**

1- Success.
0- This function is not supported.
-2- Generic error.
-12- Failed, as the 3D Accelerometer interface is open. Use accClose to close the interface.
-13- Motion Sensor interface already open.

Declaration:

```
FUNCTION msOpen : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   wk    : DINT;
END_VAR;
// Open Motion Sensor interface
msOpen();
// Enable wake on vibration with high sensitivity
msVibrationSetWakeup(sensitivity := 80, enable := ON);
// Sleep 600 seconds or wake on vibration detection
wk := pmSuspend(time:=600, mode := 0);
...
// Close Motion Sensor interface
msClose();
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.30.2.2 msClose (Function)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.36.00

Closes the Motion Sensor interface, switching off the various sensors and clearing the possible wake-up sources.

Returns: **INT**

1- Success.
0- This function is not supported.
-1- Interface is not open. Call msOpen first.
-2- Generic error.

Declaration:

```
FUNCTION msClose : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   wk    : DINT;
END_VAR;
// Open Motion Sensor interface
msOpen();
```

```
// Enable wake on vibration with high sensitivity
msVibrationSetWakeup(sensitivity := 80, enable := ON);
// Sleep 600 seconds or wake on vibration detection
wk := pmSuspend(time:=600, mode := 0);
...
// Close Motion Sensor interface
msClose();
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.30.2.3 msConfig (Function)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.50.00

Set the data acquisition rate for the motion sensor devices.

Input:

Rate : INT (10, 25, 50, 100) (default 100)
The data acquisition rate in samples per second (Hz).

Returns: INT

1- Success.
0- This function is not supported.
-1- Interface is not open. Call msOpen first.
-2- Generic error.
-3- Mode not supported.

Declaration

```
FUNCTION msConfig : INT;
VAR_INPUT
   Rate : INT := 100;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc    : INT;
END_VAR;
// Open Motion Sensor interface
msOpen();
// Set the sample rate to 50 samples per second
rc := msConfig(Rate:=50);
DebugFmt(message := "msConfig (rc=\1)", v1 := rc);
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.30.3 Data acquisition

4.2.30.3.1 msAccEnable (Function)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.50.00

Configure and enable/disable the accelerometer for data acquisition using msDataSet with msRead or through the data logger (see msLoggerCreate).
The accelerometer data can reflect either absolute or relative mode. In absolute mode the 1g gravity pull on the device will show, otherwise not.
Mixed mode gives absolute data readings, but events are triggered by relative measurements.

Input:

Enable : BOOL (default TRUE)
Enable or disable the accelerometer.

Mode : SINT (1, 2, 3) (default 2)
Mode of operation. 1 = relative mode, 2 = absolute mode, 3 = mixed mode.

Resolution : INT (2, 4, 8, 16) (default 16)
The range of the measurements the device can measure in units of gravity g. F.ex. the value 2 represents the range: [-2g ; +2g].

Returns: INT

1- Success.
0- This function is not supported.
-1- Interface is not open. Call msOpen first.
-2- Generic error.
-3- Invalid configuration, including if the sensor is already enabled/disabled.
-6- The accelerometer is not present.

Declaration

```
FUNCTION msAccEnable : INT;
VAR_INPUT
   Enable     : BOOL := TRUE;
   Mode       : SINT := 2;
   Resolution : INT  := 16;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
  rc : INT;
END_VAR;
// Open interface
rc := msOpen();
DebugFmt(Message:="msOpen (rc=\1)", v1:=rc);
// enable accelerometer
rc := msAccEnable(enable:=TRUE, mode:=2, resolution:=16);
DebugFmt(Message:="accEnable ON (rc=\1)", v1:=rc);
...
BEGIN
   ...
```

```
END;
END_PROGRAM;
```

4.2.30.3.2 msGyrEnable (Function)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.50.00

Configure and enable/disable the gyroscope for data acquisition using msDataSet with msRead or through the data logger (see msLoggerCreate).
The gyroscope measures the angular rate of rotation of the device acting on each axis.
Thus for a constant rotation of the device around an axis, a constant non-zero value should be measured for the given axis.

Input:

Enable : BOOL (default TRUE)
Enable or disable the gyroscope.

Resolution : INT (125, 250, 500, 1000) (default 125)
The range of the measurements the device can measure in units of degrees per second. F.ex. the value 125 represents the range: [-125dps ; +125dps].

Returns: INT

1- Success.
0- This function is not supported.
-1- Interface is not open. Call msOpen first.
-2- Generic error.
-3- Invalid configuration, including if the sensor is already enabled/disabled.
-6- The gyroscope is not present.

Declaration

```
FUNCTION msGyrEnable : INT;
VAR_INPUT
   Enable     : BOOL := TRUE;
   Resolution : INT  := 125;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
  rc : INT;
END_VAR;
// Open interface
rc := msOpen();
DebugFmt(Message:="msOpen (rc=\1)", v1:=rc);
// enable gyroscope
rc := msGyrEnable(enable:=TRUE, resolution:=125);
DebugFmt(Message:="gyrEnable ON (rc=\1)", v1:=rc);
...
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.30.3.3 msMagEnable (Function)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.52.00

Enable/disable the magnetometer for data acquisition using msDataSet with msRead or through the data logger (see msLoggerCreate).
The magnetometer measures the magnetic field impacting the device, and should be able to detect magnetic north/south in an environment without too much magnetic disturbances.

Input:

Enable : BOOL (default TRUE)
Enable or disable the magnetometer.

Returns: INT

1- Success.
0- This function is not supported.
-1- Interface is not open. Call msOpen first.
-2- Generic error.
-3- Invalid configuration, including if the sensor is already enabled/disabled.
-6- The magnetometer is not present.

Declaration

```
FUNCTION msMagEnable : INT;
VAR_INPUT
   Enable     : BOOL := TRUE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
  rc : INT;
END_VAR;
// Open interface
rc := msOpen();
DebugFmt(Message:="msOpen (rc=\1)", v1:=rc);
// enable magnetometer
rc := msMagEnable(enable:=TRUE);
DebugFmt(Message:="magEnable ON (rc=\1)", v1:=rc);
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.30.3.4 msDataSet (Structure)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.50.00

Structure to read a simultaneous set of data from the motion sensor devices. Use with msRead.

Fields:

Time : DINT
The timestamp of the dataset. Represents milliseconds elapsed after the first sensor was turned on, or the logger started.

AccX : FLOAT
The X-axis composant of the received data from the accelerometer.

AccY : FLOAT
The Y-axis composant of the received data from the accelerometer.

AccZ : FLOAT
The Z-axis composant of the received data from the accelerometer.

GyrX : FLOAT
The X-axis composant of the received data from the gyroscope.

GyrY : FLOAT
The Y-axis composant of the received data from the gyroscope.

GyrZ : FLOAT
The Z-axis composant of the received data from the gyroscope.

MagX : FLOAT
The X-axis composant of the received data from the magnetometer.

MagY : FLOAT
The X-axis composant of the received data from the magnetometer.

MagZ : FLOAT
The X-axis composant of the received data from the magnetometer.

Declaration:

```
STRUCT_BLOCK msDataSet;
   Time      : DINT;
   AccX      : FLOAT;
   AccY      : FLOAT;
   AccZ      : FLOAT;
   GyrX      : FLOAT;
   GyrY      : FLOAT;
   GyrZ      : FLOAT;
   MagX      : FLOAT;
   MagY      : FLOAT;
   MagZ      : FLOAT;
END_STRUCT_BLOCK;
```

4.2.30.3.5 msRead (Function)

```
Architecture:       NX32L
Device support:     NX-200, LX2
Firmware version:   1.50.00
```

Acquire a set of simultaneous data from the motion sensor devices.
Sensors that are not enabled will return 'not a number' NaN for their readings.

Input:

Data : msDataSet
The variable to receive the data through. See msDataSet.

Returns: INT

1- Data available.
0- This function is not supported.
-1- Interface is not open. Call msOpen first.
-2- Generic error.
-4- Invalid input type.
-15- No sensors are on. Use some of the functions: msAccEnable, msGyrEnable and msMagEnable.

Declaration

```
FUNCTION msRead : INT;
VAR_INPUT
   Data : ACCESS msDataSet;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
VAR
  msd : msDataSet;
END_VAR;

THREAD_BLOCK Reader
VAR
  rc : INT;
END_VAR;
WHILE TRUE DO
  rc := msRead(data:=msd);
  IF rc = 1 THEN
    DebugMsg(Message:="ACC >>> X: "+floatToStr(V:=(msd.AccX*1000.0))+" mg  Y:
"+
             floatToStr(v:=(msd.AccY*1000.0))+" mg  Z: "+
             floatToStr(v:=(msd.AccZ*1000.0))+" mg");
    DebugMsg(Message:="GYR >>> X: "+floatToStr(v:=(msd.GyrX))+" dps  Y: "+
             floatToStr(v:=(msd.GyrY))+" dps  Z: "+
             floatToStr(v:=(msd.GyrZ))+" dps");
    DebugMsg(Message:="MAG >>> X: "+floatToStr(v:=(msd.MagX))+" gauss  Y: "+
             floatToStr(v:=(msd.MagY))+" gauss  Z: "+
             floatToStr(v:=(msd.MagZ))+" gauss");
    DebugFmt(Message:=" Time : \4", v4:=(msd.time/100));
  END_IF;
  Sleep(Delay:=500); // print a sample twice a second
END_WHILE;
END_THREAD_BLOCK;

PROGRAM example;
VAR
  ms_read : Reader;
  rc : INT;
END_VAR;
// Open interface
rc := msOpen();
DebugFmt(Message:="msOpen (rc=\1)", v1:=rc);
// enable accelerometer
rc := msAccEnable(enable:=TRUE, mode:=2, resolution:=16);
DebugFmt(Message:="accEnable ON (rc=\1)", v1:=rc);
// enable gyroscope
rc := msGyrEnable(enable:=TRUE, resolution:=125);
```

```
DebugFmt(Message:="gyrEnable ON (rc=\1)", v1:=rc);
// enable magnetometer
rc := msMagEnable(enable:=TRUE);
DebugFmt(Message:="magEnable ON (rc=\1)", v1:=rc);
// start reader thread
ms_read();
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.30.4 Data logger

4.2.30.4.1 msLoggerCreate (Function)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.50.00

This function will establish a logger that can be used to log motion sensor data into a user provided buffer.
The detailed configuration of the established logger is done with the functions msLoggerAddAcc, msLoggerAddGyr and msLoggerAddMag.
One can only have a maximum of 8 loggers at a time.

Data from a logger will always deliver a full set as the first set, and then depending on downsample values will deliver partial sets.
With downsampling on the accelerometer at 1, gyroscope 2 and magnetomter 3, the data received will look like:

Gyroscope sets	Acceleromter sets	Magnetometer sets	Time
GYR0	ACC0	MAG0	T0
	ACC1		T1
GYR1	ACC2		T2
	ACC3	MAG1	T3
GYR2	ACC4		T4
	ACC5		T5
GYR3	ACC6	MAG2	T6
...	...	...	...

When switching loggers, the first set will be a full set again.

In case there is no sensor sets for a given time, the timestamp will not be included, thus there can not be two successive timestamps.

Input:

Logger : SYSHANDLE
A user provided handle to uniquely identify the logger. See SYSHANDLE page. This field is mandatory.

Buffer : PTR
A pointer to the user provided buffer.

Size : DINT
Size in bytes of the user provided buffer. The size of the buffer must be at least 400 bytes for a cyclical buffer and otherwise at least 4000 bytes.

StopOnFull : BOOL (default FALSE)
If true, the buffer will stop collecting data when there is no more space for further data records in the buffer. Otherwise the buffer will function as a ring buffer.

Returns: INT

1- Success.
0- This function is not supported.
-1- Interface is not open. Call msOpen first.
-2- Generic error.
-3- Invalid configuration.
-7- The maximum amount of loggers are present.
-8- The logger was already established.

Declaration

```
FUNCTION msLoggerCreate : INT;
VAR_INPUT
   Logger       : ACCESS SYSHANDLE;
   Buffer       : MANDATORY PTR;
   Size         : MANDATORY DINT;
   StopOnFull   : BOOL := FALSE;
END_VAR;
```

Example:

```
// For a more complete example, see example under msReadEvent
INCLUDE rtcu.inc

PROGRAM test;
VAR
   logger       : SYSHANDLE;
   buf          : ARRAY[1..10000] OF DINT;
   rc           : INT;
END_VAR;
rc := msOpen();
DebugFmt(message:="msOpen (rc=\1)", v1:=rc);
...
rc := msLoggerCreate(logger:=logger, buffer:=ADDR(buf), size:=SIZEOF(buf),
stoponfull:=TRUE);
DebugFmt(message:="msLoggerCreate (rc=\1) buffer_size=\4, stoponfull=TRUE",
v1:=rc, v4:=SIZEOF(buf));
...
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.30.4.2 msLoggerDestroy (Function)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.50.00

Remove specified logger from available loggers.

Input:

Logger : SYSHANDLE
A user provided handle to uniquely identify the logger. See SYSHANDLE page. This field is mandatory.
The handle will be invalidated after a successful call to this function.

Returns: INT

1- Success.
0- This function is not supported.
-1- Interface is not open. Call msOpen first.
-2- Generic error.
-5- The logger is not present.
-9- The specified logger is running. Call msLoggerStop first.

Declaration

```
FUNCTION msLoggerDestroy : INT;
VAR_INPUT
   Logger       : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   logger       : SYSHANDLE;
END_VAR;

PROGRAM test;
VAR
   rc           : INT;
   buf1         : ARRAY[1..10000] OF DINT;
END_VAR;
rc := msOpen();
DebugFmt(message:="msOpen (rc=\1)", v1:=rc);
...
rc := msLoggerCreate(logger:=logger, buffer:=ADDR(buf1), size:=SIZEOF(buf1),
stoponfull:=TRUE);
DebugFmt(message:="msLoggerCreate (rc=\1) buffer_size=\4, stoponfull=TRUE",
v1:=rc, v4:=SIZEOF(buf1));
...
rc := msLoggerDestroy(logger:=logger);
...
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.30.4.3 msLoggerAddAcc (Function)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.50.00

Add logging of accelerometer data to logger. Each set of accelerometer data in full, consists of 2 bytes per axis (X, Y and Z) stored in the buffer.

Input:

Logger : SYSHANDLE
A user provided handle to uniquely identify the logger. See SYSHANDLE page. This field is mandatory.

Downsample : UINT (default 1)
Only collect a accelerometer data set after this amount of measurements, thus saving buffer space.
Values 0 and 1 represent no downsampling.

LowRes : BOOL (default FALSE)
If true, only store the most significant byte of each axis in the buffer.

Returns: INT

1- Success.
0- This function is not supported.
-1- Interface is not open. Call msOpen first.
-2- Generic error.
-5- The logger is not present.
-6- The accelerometer is not present.
-9- The specified logger is running. Call msLoggerStop first.

Declaration

```
FUNCTION msLoggerAddAcc : INT;
VAR_INPUT
   Logger     : MANDATORY SYSHANDLE;
   Downsample : UINT := 1;
   LowRes     : BOOL := FALSE;
END_VAR;
```

Example:

```
// For a more complete example, see example under msReadEvent
INCLUDE rtcu.inc

VAR
   logger       : ARRAY[1..2] OF SYSHANDLE;
END_VAR;

PROGRAM test;
VAR
   rc           : INT;
   buf1         : ARRAY[1..10000] OF DINT;
END_VAR;
rc := msOpen();
DebugFmt(message:="msOpen (rc=\1)", v1:=rc);
...
rc := msLoggerCreate(logger:=logger[1], buffer:=ADDR(buf1),
size:=SIZEOF(buf1), stoponfull:=TRUE);
DebugFmt(message:="msLoggerCreate (rc=\1) buffer_size=\4, stoponfull=TRUE",
v1:=rc, v4:=SIZEOF(buf1));
...
rc := msLoggerAddAcc(logger:=logger[1], downsample:=0, lowres:=FALSE);
DebugFmt(message:="msLoggerAddAcc (rc=\1) downsample=0, lowres=FALSE",
v1:=rc);
...
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.30.4.4 msLoggerAddGyr (Function)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.50.00

Add logging of gyroscope data to logger. Each set of gyroscope data in full, consists of 2 bytes per axis (X, Y and Z) stored in the buffer.

Input:

Logger : SYSHANDLE
A user provided handle to uniquely identify the logger. See SYSHANDLE page. This field is mandatory.

Downsample : UINT (default 1)
Only collect a gyroscope data set after this amount of measurements, thus saving buffer space. Values 0 and 1 represent no downsampling.

LowRes : BOOL (default FALSE)
If true, only store the most significant byte of each axis in the buffer.

Returns: INT

1- Success.
0- This function is not supported.
-1- Interface is not open. Call msOpen first.
-2- Generic error.
-5- The logger is not present.
-6- The gyroscope is not present.
-9- The specified logger is running. Call msLoggerStop first.

Declaration

```
FUNCTION msLoggerAddGyr : INT;
VAR_INPUT
   Logger     : MANDATORY SYSHANDLE;
   Downsample : UINT := 1;
   LowRes     : BOOL := FALSE;
END_VAR;
```

Example:

```
// For a more complete example, see example under msReadEvent
INCLUDE rtcu.inc

VAR
   logger       : ARRAY[1..2] OF SYSHANDLE;
END_VAR;

PROGRAM test;
VAR
   rc           : INT;
   buf1         : ARRAY[1..10000] OF DINT;
END_VAR;
rc := msOpen();
DebugFmt(message:="msOpen (rc=\1)", v1:=rc);
...
rc := msLoggerCreate(logger:=logger[1], buffer:=ADDR(buf1),
size:=SIZEOF(buf1), stoponfull:=TRUE);
DebugFmt(message:="msLoggerCreate (rc=\1) buffer_size=\4, stoponfull=TRUE",
v1:=rc, v4:=SIZEOF(buf1));
...
rc := msLoggerAddGyr(logger:=logger[1], downsample:=10, lowres:=FALSE);
DebugFmt(message:="msLoggerAddGyr (rc=\1) downsample=10, lowres=FALSE",
v1:=rc);
...
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.30.4.5 msLoggerAddMag (Function)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.52.00

Add logging of magnetometer data to logger. Each set of magnetometer data in full, consists of 2 bytes per axis (X, Y and Z) stored in the buffer.

Input:

Logger : SYSHANDLE
A user provided handle to uniquely identify the logger. See SYSHANDLE page. This field is mandatory.

Downsample : UINT (default 1)
Only collect a magnetometer data set after this amount of measurements, thus saving buffer space.
Values 0 and 1 represent no downsampling.

LowRes : BOOL (default FALSE)
If true, only store the most significant byte of each axis in the buffer.

Returns: INT

1- Success.
0- This function is not supported.
-1- Interface is not open. Call msOpen first.
-2- Generic error.
-5- The logger is not present.
-6- The magnetometer is not present.
-9- The specified logger is running. Call msLoggerStop first.

Declaration

```
FUNCTION msLoggerAddMag : INT;
VAR_INPUT
   Logger     : MANDATORY SYSHANDLE;
   Downsample : UINT := 1;
   LowRes     : BOOL := FALSE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
VAR
   logger       : ARRAY[1..2] OF SYSHANDLE;
END_VAR;

PROGRAM test;
VAR
   rc           : INT;
   buf1         : ARRAY[1..10000] OF DINT;
END_VAR;
rc := msOpen();
DebugFmt(message:="msOpen (rc=\1)", v1:=rc);
...
rc := msLoggerCreate(logger:=logger[1], buffer:=ADDR(buf1),
size:=SIZEOF(buf1), stoponfull:=TRUE);
DebugFmt(message:="msLoggerCreate (rc=\1) buffer_size=\4, stoponfull=TRUE",
v1:=rc, v4:=SIZEOF(buf1));
```

```
...
rc := msLoggerAddMag(logger:=logger[1], downsample:=0, lowres:=FALSE);
DebugFmt(message:="msLoggerAddMag (rc=\1) downsample=0, lowres=FALSE",
v1:=rc);
...
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.30.4.6 msLoggerStart (Function)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.50.00

Start collecting motion sensor data to the buffer of the specified logger.

Before calling this function, it is necessary to be sure to enable all the specified sensors that have been added to the logger with the functions msAccEnable, msGyrEnable and msMagEnable.

Starting or restarting (after it has been stopped) the logger function, resets the timestamp to zero.

Input:

Logger : SYSHANDLE
A user provided handle to uniquely identify the logger. See SYSHANDLE page. This field is mandatory.

Reset : BOOL (default TRUE)
Start with a clean buffer.

Returns: INT

1- Success.
0- This function is not supported.
-1- Interface is not open. Call msOpen first.
-2- Generic error.
-5- The logger is not present.
-9- The specified logger is running. Call msLoggerStop first.
-15- The needed sensors are not enabled.

Declaration

```
FUNCTION msLoggerStart : INT;
VAR_INPUT
   Logger      : MANDATORY SYSHANDLE;
   Reset       : BOOL := TRUE;
END_VAR;
```

Example:

```
// For a more complete example, see example under msReadEvent
INCLUDE rtcu.inc

VAR
   logger        : ARRAY[1..2] OF SYSHANDLE;
END_VAR;

PROGRAM test;
VAR
   rc            : INT;
   buf1          : ARRAY[1..10000] OF DINT;
```

```
END_VAR;
rc := msOpen();
DebugFmt(message:="msOpen (rc=\1)", v1:=rc);
...
rc := msLoggerCreate(logger:=logger[1], buffer:=ADDR(buf1),
size:=SIZEOF(buf1), stoponfull:=TRUE);
DebugFmt(message:="msLoggerCreate (rc=\1) buffer_size=\4, stoponfull=TRUE",
v1:=rc, v4:=SIZEOF(buf1));
...
rc := msLoggerAddAcc(logger:=logger[1], downsample:=0, lowres:=FALSE);
DebugFmt(message:="msLoggerAddAcc (rc=\1) downsample=0, lowres=FALSE",
v1:=rc);
...
rc := msLoggerStart(logger:=logger[1], reset:=TRUE);
DebugFmt(message:="msLoggerStart (rc=\1) reset=TRUE", v1:=rc);
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.30.4.7 msLoggerStop (Function)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.50.00

Stop collection of data with the specified logger.

Input:

Logger : SYSHANDLE
A user provided handle to uniquely identify the logger. See SYSHANDLE page. This field is mandatory.

SwitchTo : SYSHANDLE
Handle to another logger to switch to. Disabled if not specified.

Reset : BOOL (Default TRUE)
Reset the new logger when switched in.

Returns: INT

1- Success.
0- This function is not supported.
-1- Interface is not open. Call msOpen first.
-2- Generic error.
-3- Invalid switch logger handle.
-5- The logger to stop is not present.
-10- The specified logger is not running. Call msLoggerStart first.
-17- Switch-logger sensor setup is invalid.

Declaration

```
FUNCTION msLoggerStop : INT;
VAR_INPUT
   Logger     : MANDATORY SYSHANDLE;
   SwitchTo   : SYSHANDLE;
   Reset      : BOOL := TRUE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   logger       : ARRAY[1..2] OF SYSHANDLE;
END_VAR;

PROGRAM test;
VAR
   rc           : INT;
   buf1         : ARRAY[1..10000] OF DINT;
   buf2         : ARRAY[1..10000] OF DINT;
END_VAR;
rc := msOpen();
DebugFmt(message:="msOpen (rc=\1)", v1:=rc);
...
rc := msLoggerCreate(logger:=logger[1], buffer:=ADDR(buf1),
size:=SIZEOF(buf1), stoponfull:=TRUE);
DebugFmt(message:="msLoggerCreate (rc=\1) buffer_size=\4, stoponfull=TRUE",
v1:=rc, v4:=SIZEOF(buf1));
rc := msLoggerCreate(logger:=logger[2], buffer:=ADDR(buf2),
size:=SIZEOF(buf2), stoponfull:=TRUE);
DebugFmt(message:="msLoggerCreate (rc=\1) buffer_size=\4, stoponfull=TRUE",
v1:=rc, v4:=SIZEOF(buf1));
...
rc := msLoggerAddAcc(logger:=logger[1], downsample:=0, lowres:=FALSE);
DebugFmt(message:="msLoggerAddAcc (rc=\1) downsample=0, lowres=FALSE",
v1:=rc);
...
rc := msLoggerStart(logger:=logger[1], reset:=TRUE);
DebugFmt(message:="msLoggerStart (rc=\1) reset=TRUE", v1:=rc);
...
rc := msLoggerStop(logger:=logger[1], switchto:= logger[2], reset:=TRUE);
...
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.30.4.8 msLoggerNext (Function)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.50.00

Request which type of record is the next to be read from the logger buffer. Subsequently read the specified record with msLoggerRead.
Record types are:

Define	Value	Description
_MS_LOGGER_NODATA	0	No more data to be read from current buffer
_MS_LOGGER_TIMESTAMP	1	Timestamp of the dataset. Represents seconds elapsed after the first sensor was turned on.
_MS_LOGGER_ACCDATA	2	Accelerometer data set.
_MS_LOGGER_GYRDATA	3	Gyroscope data set.
_MS_LOGGER_MAGDATA	4	Magnetometer data set.

The logger will place sensor records in the following order:
Gyroscope, Accelerometer, Magnetometer, Timestamp.

Thus a timestamp always belongs to the preceding sensor measurements.

Input:

Logger : SYSHANDLE
A user provided handle to uniquely identify the logger. See SYSHANDLE page. This field is mandatory.

Returns: INT

>0- Record ID.
0- No new data.
-1- Interface is not open. Call msOpen first.
-2- Generic error.
-5- The logger is not present.
-9- The specified logger is running. Call msLoggerStop first.

Declaration

```
FUNCTION msLoggerNext : INT;
VAR_INPUT
   Logger     : MANDATORY SYSHANDLE;
END_VAR;
```

Example:

```
// See example under msLoggerRead
```

4.2.30.4.9 msLoggerRead (Function)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.50.00

Read a data set from the specified inactive logger buffer.

Input:

Logger : SYSHANDLE
A user provided handle to uniquely identify the logger. See SYSHANDLE page. This field is mandatory.

Data : PTR
A pointer to a structure to read the sensor data into. The structure type must be according to the data being read. See msLoggerNext for how to get the structure type.

Returns: INT

1- Success.
0- This function is not supported.
-1- Interface is not open. Call msOpen first.
-2- Generic error.
-4- Illegal input data type.
-5- The logger is not present.
-9- The specified logger is running. Call msLoggerStop first.
-16- Logger buffer error.

Declaration

```
FUNCTION msLoggerRead : INT;
VAR_INPUT
   Logger      : MANDATORY SYSHANDLE;
   Data        : PTR;
END_VAR;
```

Example:

```
VAR
  acc           : msAccData;
  gyr           : msGyrData;
  timestamp     : msTimestamp;
END_VAR;

FUNCTION readLogger;
VAR_INPUT
   _logger : SYSHANDLE;
END_VAR;
VAR
   next          : INT;
   rc            : INT;
END_VAR;
REPEAT
   next := msLoggerNext(logger:=_logger);
   CASE next OF
      _MS_LOGGER_NODATA:
         DebugMsg(message:="No more data in buffer!");
      _MS_LOGGER_TIMESTAMP:
         rc := msLoggerRead(logger:=_logger, data:=ADDR(timestamp));
         DebugFmt(message:="Timestamp: \4", v4:=timestamp.time);
         DebugMsg(message:="--------------------------------");
      _MS_LOGGER_ACCDATA:
         rc := msLoggerRead(logger:=_logger, data:=ADDR(acc));
         DebugMsg(message:="  ACC >>> X: "+floatToStr(v:=(acc.x*1000.0))+" mg
 Y: "+
                     floatToStr(v:=(acc.y*1000.0))+" mg  Z: "+
                     floatToStr(v:=(acc.z*1000.0))+" mg");
      _MS_LOGGER_GYRDATA:
         rc := msLoggerRead(logger:=_logger, data:=ADDR(gyr));
         DebugMsg(message:="  GYR >>> X: "+floatToStr(v:=(gyr.x))+" dps  Y: "+
                     floatToStr(v:=(gyr.y))+" dps  Z: "+
                     floatToStr(v:=(gyr.z))+" dps");
   ELSE
      DebugFmt(message:="Invalid record ID \1", v1:=next);
   END_CASE;
   IF rc <> 1 THEN
      DebugFmt(message:="msLoggerRead (rc=\1)", v1:=rc);
   END_IF;
UNTIL (next = _MS_LOGGER_NODATA);
END_REPEAT;
END_FUNCTION;
```

4.2.30.4.10 msLoggerLevel (Function)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.50.00

Tell how full the specified logger buffer is in promille.

Input:

Logger : SYSHANDLE
A user provided handle to uniquely identify the logger. See SYSHANDLE page. This field is mandatory.

Returns: INT

0-- Buffer full level. 0 means buffer is empty. 1000 means buffer is full.
10
00
-1- Interface is not open. Call msOpen first.
-2- Generic error.
-5- The logger is not present.

Declaration

```
FUNCTION msLoggerLevel : INT;
VAR_INPUT
   Logger     : MANDATORY SYSHANDLE;
END_VAR;
```

Example:

```
// For a more complete example, see example under msReadEvent
...
logger_lvl := msLoggerLevel(logger:=logger[1]);
DebugFmt(message:="Logger #1 level is at \1", v1:=logger_lvl);
...
```

4.2.30.4.11 msTimestamp (Structure)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.50.00

Structure to read a timestamp from a logger buffer. See msLoggerNext and msLoggerRead.

Fields:

Time : DINT
The timestamp of the dataset. Represents milliseconds elapsed after the logger was started.

Declaration:

```
STRUCT_BLOCK msTimestamp;
   Time      : DINT;
END_STRUCT_BLOCK;
```

4.2.30.4.12 msAccData (Structure)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.50.00

Structure to read a set of accelerometer sensor data from a logger buffer. See msLoggerNext and msLoggerRead.

Fields:

X : FLOAT
The X-axis composant of the received data from the accelerometer.

Y : FLOAT
The Y-axis composant of the received data from the accelerometer.

Z : FLOAT
The Z-axis composant of the received data from the accelerometer.

Declaration:

```
STRUCT_BLOCK msAccData;
   X          : FLOAT;
   Y          : FLOAT;
   Z          : FLOAT;
END_STRUCT_BLOCK;
```

4.2.30.4.13 msGyrData (Structure)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.50.00

Structure to read a set of gyroscope sensor data from a logger buffer. See msLoggerNext and msLoggerRead.

Fields:

X : FLOAT
The X-axis composant of the received data from the gyroscope.

Y : FLOAT
The Y-axis composant of the received data from the gyroscope.

Z : FLOAT
The Z-axis composant of the received data from the gyroscope.

Declaration:

```
STRUCT_BLOCK msGyrData;
   X          : FLOAT;
   Y          : FLOAT;
   Z          : FLOAT;
END_STRUCT_BLOCK;
```

4.2.30.4.14 msMagData (Structure)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.50.00

Structure to read a set of magnetometer sensor data from a logger buffer. See msLoggerNext and msLoggerRead.

Fields:

X : FLOAT
The X-axis composant of the received data from the magnetometer.

Y : FLOAT
The X-axis composant of the received data from the magnetometer.

Z : FLOAT
The X-axis composant of the received data from the magnetometer.

Declaration:

```
STRUCT_BLOCK msMagData;
   X          : FLOAT;
   Y          : FLOAT;
   Z          : FLOAT;
END_STRUCT_BLOCK;
```

4.2.30.5 Event handling

4.2.30.5.1 msEventRegister (Function)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.50.00

Create an event handler consisting of an event configuration and a possible action to follow.

Shock and acceleration events are treated separately, and will trigger as long as they are registered and the logger is running.
For performance reasons, there can only be one active configuration for each. See limitations.
It is always the last configuration registered that is used.
If the last configuration is unregistered, the detection will be disabled and a new must be registered for the detection to continue.
In case an action is connected to a shock or acceleration event, it will only be done once.

When any of the other events has been triggered, it will automatically be deactivated and unregistered.

Input:

Event : PTR
A mandatory pointer to a structure containing the event configuration.
See msEventLogWatermark, msEventLogFull, msEventShock and msEventAcceleration.

Action : PTR
An optional pointer to a structure containing the possible action configuration.
See msActionStopLogger, msActionStartLogger and msActionSwitchLogger.

Returns: DINT

>0- The ID of the event handler.
0- This function is not supported.
-1- Interface is not open. Call msOpen first.
-2- Generic error.
-3- Invalid configuration.
-4- Invalid input type.
-5- The logger is not present.
-17- Switch-logger sensor setup is invalid.

Declaration

```
FUNCTION msEventRegister : DINT;
VAR_INPUT
   Event        : MANDATORY PTR;
   Action       : PTR;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   logger      : SYSHANDLE;
   buf         : ARRAY[1..2000] OF DINT;
   timestamp   : msTimestamp;
   acc         : msAccData;
   gyr         : msGyrData;
   running     : BOOL;
END_VAR;

THREAD_BLOCK msHandleEvents;
VAR
   rc        : INT;
   eventID   : DINT;
   watermark : msReadWatermark;
   shock     : msReadShockEvent;
   accel     : msReadAccelEvent;
END_VAR;
DebugMsg(message:="Event Handler started!");
running := TRUE;
WHILE running DO
   rc := msWaitEvent();
   IF rc < 0 THEN
      DebugFmt(message:="msWaitEvent error (rc=\1)", v1:=rc);
   ELSE
      CASE rc OF
         _MS_EVENT_TIMEOUT:
            DebugMsg(message:="msWaitEvent - Timeout!");
         _MS_EVENT_LOGGER_LEVEL:
            msReadEvent(event:=eventID, data:=ADDR(watermark));
            DebugFmt(message:="Watermark event received with id: \4",
v4:=eventID);
            DebugFmt(message:="Level of buffer was \1", v1:=watermark.level);
         _MS_EVENT_LOGGER_FULL:
            msReadEvent(event:=eventID);
            DebugFmt(message:="Logger full event received with id: \4",
v4:=eventID);
         _MS_EVENT_LOGGER_STOPPED:
            msReadEvent(event:=eventID);
            DebugFmt(message:="Logger stopped event received with id: \4",
v4:=eventID);
         _MS_EVENT_LOGGER_STARTED:
            msReadEvent(event:=eventID);
            DebugFmt(message:="Logger started event received with id: \4",
v4:=eventID);
         _MS_EVENT_LOGGER_SWITCHED:
            msReadEvent(event:=eventID);
            DebugFmt(message:="Logger switched event received with id: \4",
v4:=eventID);
         _MS_EVENT_SHOCK:
            msReadEvent(event:=eventID, data:=ADDR(shock));
            DebugFmt(message:="Shock event received with id: \4",
v4:=eventID);
            DebugFmt(message:="Timestamp was \4", v4:=shock.time);
```

```
            IF shock.x THEN
               DebugMsg(message:="Shock event on X");
            END_IF;
            IF shock.y THEN
               DebugMsg(message:="Shock event on y");
            END_IF;
            IF shock.z THEN
               DebugMsg(message:="Shock event on Z");
            END_IF;
         _MS_EVENT_ACCELERATION:
            msReadEvent(event:=eventID, data:=ADDR(accel));
            DebugFmt(message:="Acceleration event received with id: \4",
v4:=eventID);
            DebugFmt(message:="Acc : X "+floatToStr(v:=accel.accx)+
                              ", Y "+floatToStr(v:=accel.accy)+
                              ", Z "+floatToStr(v:=accel.accz));
            IF accel.xabove THEN
               DebugMsg(message:="X was above threshold");
            END_IF;
            IF accel.yabove THEN
               DebugMsg(message:="Y was above threshold");
            END_IF;
            IF accel.zabove THEN
               DebugMsg(message:="Z was above threshold");
            END_IF;
            IF accel.xbelow THEN
               DebugMsg(message:="X was below threshold");
            END_IF;
            IF accel.ybelow THEN
               DebugMsg(message:="Y was below threshold");
            END_IF;
            IF accel.zbelow THEN
               DebugMsg(message:="Z was below threshold");
            END_IF;
      ELSE
         DebugFmt(message:="Unknown event received: \1", v1:=rc);
      END_CASE;
   END_IF;
END_WHILE;
END_THREAD_BLOCK;


PROGRAM test;
VAR
   ms_events  : msHandleEvents;
   rc         : INT;
   logger_lvl : INT;
   accelcfg   : msEventAcceleration;
   accelid    : DINT;
   shockcfg   : msEventShock;
   shockid    : DINT;
END_VAR;
rc := msOpen();
DebugFmt(message:="msOpen (rc=\1)", v1:=rc);
ms_events();
rc := msAccEnable(enable:=TRUE, mode:=3);
DebugFmt(message:="accEnable ON (rc=\1)", v1:=rc);
rc := msGyrEnable(enable:=TRUE);
DebugFmt(message:="gyrEnable ON (rc=\1)", v1:=rc);

rc := msLoggerCreate(logger:=logger, buffer:=ADDR(buf), size:=SIZEOF(buf),
stoponfull:=FALSE);
DebugFmt(message:="msLoggerCreate (rc=\1) buffer_size=\4, stoponfull=FALSE",
v1:=rc, v4:=SIZEOF(buf));
```

```
rc := msLoggerAddAcc(logger:=logger, downsample:=100, lowres:=TRUE);
DebugFmt(message:="msLoggerAddAcc (rc=\1) downsample=100, lowres=TRUE",
v1:=rc);
rc := msLoggerAddGyr(logger:=logger, downsample:=100, lowres:=TRUE);
DebugFmt(message:="msLoggerAddGyr (rc=\1) downsample=100, lowres=TRUE",
v1:=rc);

shockid := msEventRegister(event:=ADDR(shockcfg));
IF shockid > 0 THEN
   DebugFmt(message:="Success, shock event id is \4", v4:=shockid);
ELSE
   DebugFmt(message:="msEventRegister  (rc=\4)", v4:=shockid);
END_IF;

accelcfg.threshold := 0.1;
accelid := msEventRegister(event:=ADDR(accelcfg));
IF accelid > 0 THEN
   DebugFmt(message:="Success, acceleration event id is \4", v4:=accelid);
ELSE
   DebugFmt(message:="msEventRegister  (rc=\4)", v4:=accelid);
END_IF;

rc := msLoggerStart(logger:=logger, reset:=TRUE);
DebugFmt(Message:="msLoggerStart (rc=\1) reset=TRUE", v1:=rc);
BEGIN
   Sleep(delay:=10000);
END;
END_PROGRAM;
```

4.2.30.5.2 msEventUnregister (Function)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.50.00

Remove specified event handler.

After an event has been triggered it is automatically unregistered. The exception to this is shock and acceleration events.
Unregistering the last registered shock or acceleration event disables shock or acceleration detection, and there must be registered a new to continue detection.

Input:

Event : DINT
The event ID given from msEventRegister of the event handler to unregister.

Returns: INT

1- Success.
0- This function is not supported.
-1- Interface is not open. Call msOpen first.
-2- Generic error.
-11- No event handler with given ID.

Declaration

```
FUNCTION msEventUnregister : INT;
VAR_INPUT
   Event     : DINT;
END_VAR;
```

Example:

```
...
rc := msEventUnregister(event:=shockid);
DebugFmt(message:="msEventUnregister shock (rc=1)", v1:=rc);
...
```

4.2.30.5.3 msWaitEvent (Function)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.50.00

Wait until an event has occurred and return the given type of event or return from a timeout.

The event types possible:

Define	Value	Description
_MS_EVENT_TIMEOUT	1	Timeout reached.
_MS_EVENT_LOGGER_LEVEL	2	Logger reached watermark level.
_MS_EVENT_LOGGER_FULL	3	Logger is full.
_MS_EVENT_LOGGER_STOPPED	4	Logger was stopped.
_MS_EVENT_LOGGER_STARTED	5	Logger was started.
_MS_EVENT_LOGGER_SWITCHED	6	A switch of loggers occurred.
_MS_EVENT_SHOCK	7	A shock event occurred.
_MS_EVENT_ACCELERATION	8	An acceleration event occurred.

Input:

Timeout : DINT (default -1)
The timeout in seconds before returning if no event was detected in the given time. -1 to disable the timeout.

Returns: INT

>0- Event type that occurred. If there is already events on the event queue, the oldest event type is returned.
0- This function is not supported.
-1- Interface is not open. Call msOpen first.
-2- Generic error.
-3- Invalid configuration.
-19- Event handling error.
-21- Event queue overflow.

Declaration

```
FUNCTION msWaitEvent : INT;
VAR_INPUT
   Timeout   : DINT := -1;
END_VAR;
```

Example:

```
// For a more complete example, see example under msReadEvent
```

```
...
rc := msWaitEvent();
IF rc < 0 THEN
   DebugFmt(message:="msWaitEvent error (rc=\1)", v1:=rc);
END_IF;
...
```

4.2.30.5.4 msReadEvent (Function)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.50.00

Read the returned data from an event. This function also clears the event and should always be called after msWaitEvent, even when there is no data to read.

Input:

Data : PTR
A pointer to a variable to read the received data into, or omitted, depending on the type of event given.
Possible variable types include msReadWatermark, msReadShockEvent, msReadAccelEvent,

Output:

Event : DINT
The ID of the event received. Event ID 0 represents solicited events, e.g. events triggered by function calls and not registered events.

Returns: INT

1- Success.
0- This function is not supported.
-1- Interface is not open. Call msOpen first.
-2- Generic error.
-4- Invalid input type.
-18- No event to read.
-19- Event handling error.
-20- Invalid event data received.

Declaration

```
FUNCTION msReadEvent : INT;
VAR_INPUT
   Data       : PTR;
   Event      : ACCESS DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   logger       : ARRAY[1..2] OF SYSHANDLE;
   eventid      : ARRAY[1..4] OF DINT;
   accelid      : DINT;
   shockid      : DINT;
   timestamp    : msTimestamp;
   acc          : msAccData;
   gyr          : msGyrData;
```

```
   running      : BOOL;
END_VAR;

FUNCTION readLogger;
VAR_INPUT
   _logger : SYSHANDLE;
END_VAR;
VAR
   next         : INT;
   rc           : INT;
END_VAR;
   REPEAT
      next := msLoggerNext(logger:=_logger);
      CASE next OF
         _MS_LOGGER_NODATA:
            DebugMsg(message:="No more data in buffer!");
         _MS_LOGGER_TIMESTAMP:
            rc := msLoggerRead(logger:=_logger, data:=ADDR(timestamp));
            DebugFmt(message:="Timestamp: \4", v4:=timestamp.time);
            DebugMsg(message:="--------------------------------");
         _MS_LOGGER_ACCDATA:
            rc := msLoggerRead(logger:=_logger, data:=ADDR(acc));
            DebugMsg(message:="  ACC >>> X: "+floatToStr(v:=(acc.x*1000.0))+"
mg  Y: "+
                           floatToStr(v:=(acc.y*1000.0))+" mg  Z: "+
                           floatToStr(v:=(acc.z*1000.0))+" mg");
         _MS_LOGGER_GYRDATA:
            rc := msLoggerRead(logger:=_logger, data:=ADDR(gyr));
            DebugMsg(message:="  GYR >>> X: "+floatToStr(v:=(gyr.x))+" dps  Y:
"+
                           floatToStr(v:=(gyr.y))+" dps  Z: "+
                           floatToStr(v:=(gyr.z))+" dps");
      ELSE
         DebugFmt(message:="Invalid record ID \1", v1:=next);
      END_CASE;
      IF rc <> 1 THEN
         DebugFmt(message:="msLoggerRead (rc=\1)", v1:=rc);
      END_IF;
   UNTIL (next = _MS_LOGGER_NODATA);
   END_REPEAT;
END_FUNCTION;

THREAD_BLOCK msHandleEvents;
VAR
   rc           : INT;
   evid         : DINT;
   watermark    : msReadWatermark;
   shock        : msReadShockEvent;
   accel        : msReadAccelEvent;
   logger_lvl   : INT;
END_VAR;
DebugMsg(message:="Event Handler started!");
running := TRUE;
WHILE running DO
   rc := msWaitEvent();
   IF rc < 0 THEN
      DebugFmt(message:="msWaitEvent error (rc=\1)", v1:=rc);
   ELSE
      CASE rc OF
         _MS_EVENT_TIMEOUT:
            DebugMsg(message:="msWaitEvent - Timeout!");
         _MS_EVENT_LOGGER_LEVEL:
            msReadEvent(event:=evid, data:=ADDR(watermark));
            DebugFmt(message:="Watermark event received with id: \4",
```

```
v4:=evid);
            DebugFmt(message:="Level of buffer was \1", v1:=watermark.level);
         _MS_EVENT_LOGGER_FULL:
            msReadEvent(event:=evid);
            DebugFmt(message:="Logger full event received with id: \4",
v4:=evid);
         _MS_EVENT_LOGGER_STOPPED:
            msReadEvent(event:=evid);
            DebugFmt(message:="Logger stopped event received with id: \4",
v4:=evid);
         _MS_EVENT_LOGGER_STARTED:
            msReadEvent(event:=evid);
            DebugFmt(message:="Logger started event received with id: \4",
v4:=evid);
         _MS_EVENT_LOGGER_SWITCHED:
            msReadEvent(event:=evid);
            DebugFmt(message:="Logger switched event received with id: \4",
v4:=evid);
         _MS_EVENT_SHOCK:
            msReadEvent(event:=evid, data:=ADDR(shock));
            DebugFmt(message:="Shock event received with id: \4", v4:=evid);
            DebugFmt(message:="Timestamp was \4", v4:=shock.time);
            IF shock.x THEN
               DebugMsg(message:="Shock event on X");
            END_IF;
            IF shock.y THEN
               DebugMsg(message:="Shock event on y");
            END_IF;
            IF shock.z THEN
               DebugMsg(message:="Shock event on Z");
            END_IF;
         _MS_EVENT_ACCELERATION:
            msReadEvent(event:=evid, data:=ADDR(accel));
            DebugFmt(message:="Acceleration event received with id: \4",
v4:=evid);
            DebugFmt(message:="Acc : X "+floatToStr(v:=accel.accx)+
                              ", Y "+floatToStr(v:=accel.accy)+
                              ", Z "+floatToStr(v:=accel.accz));
            IF accel.xabove THEN
               DebugMsg(message:="X was above threshold");
            END_IF;
            IF accel.yabove THEN
               DebugMsg(message:="Y was above threshold");
            END_IF;
            IF accel.zabove THEN
               DebugMsg(message:="Z was above threshold");
            END_IF;
            IF accel.xbelow THEN
               DebugMsg(message:="X was below threshold");
            END_IF;
            IF accel.ybelow THEN
               DebugMsg(message:="Y was below threshold");
            END_IF;
            IF accel.zbelow THEN
               DebugMsg(message:="Z was below threshold");
            END_IF;
      ELSE
         DebugFmt(message:="Unknown event received: \1", v1:=rc);
      END_CASE;
      IF evid = shockid THEN
         logger_lvl := msLoggerLevel(logger:=logger[1]);
         DebugFmt(message:="Logger #1 level is at \1", v1:=logger_lvl);
         logger_lvl := msLoggerLevel(logger:=logger[2]);
         DebugFmt(message:="Logger #2 level is at \1", v1:=logger_lvl);
```

```
      END_IF;
      IF evid = eventid[4] THEN
         logger_lvl := msLoggerLevel(logger:=logger[2]);
         DebugFmt(message:="Logger #2 level is at \1", v1:=logger_lvl);
         rc := msLoggerStop(logger:=logger[2]);
         DebugFmt(message:="msLoggerStop (rc=\1)", v1:=rc);
         readLogger(_logger:=logger[1]);
         readLogger(_logger:=logger[2]);
         evid := 0;
         running := FALSE;
      END_IF;
   END_IF;
END_WHILE;
END_THREAD_BLOCK;

PROGRAM test;
VAR
   rc           : INT;
   ms_events    : msHandleEvents;
   accelcfg     : msEventAcceleration;
   shockcfg     : msEventShock;
   watermark    : msEventLogWatermark;
   switchLogger : msActionSwitchLogger;
   buf1         : ARRAY[1..10000] OF DINT;
   buf2         : ARRAY[1..10000] OF DINT;
END_VAR;
rc := msOpen();
DebugFmt(message:="msOpen (rc=\1)", v1:=rc);

ms_events();

rc := msAccEnable(enable:=TRUE, mode:=3);
DebugFmt(message:="accEnable ON (rc=\1)", v1:=rc);
rc := msGyrEnable(enable:=TRUE);
DebugFmt(message:="gyrEnable ON (rc=\1)", v1:=rc);

rc := msLoggerCreate(logger:=logger[1], buffer:=ADDR(buf1),
size:=SIZEOF(buf1), stoponfull:=TRUE );
DebugFmt(message:="msLoggerCreate (rc=\1) buffer_size=\4, stoponfull=TRUE",
v1:=rc, v4:=SIZEOF(buf1));
rc := msLoggerAddAcc(logger:=logger[1], downsample:=0, lowres:=FALSE);
DebugFmt(message:="msLoggerAddAcc (rc=\1) downsample=0, lowres=FALSE",
v1:=rc);
rc := msLoggerAddGyr(logger:=logger[1], downsample:=10, lowres:=FALSE);
DebugFmt(message:="msLoggerAddGyr (rc=\1) downsample=10, lowres=FALSE",
v1:=rc);
rc := msLoggerCreate(logger:=logger[2], buffer:=ADDR(buf2),
size:=SIZEOF(buf2), stoponfull:=TRUE );
DebugFmt(message:="msLoggerCreate (rc=\1) buffer_size=\4, stoponfull=TRUE",
v1:=rc, v4:=SIZEOF(buf2));
rc := msLoggerAddAcc(logger:=logger[2], downsample:=0, lowres:=FALSE);
DebugFmt(message:="msLoggerAddAcc (rc=\1) downsample=0, lowres=TRUE", v1:=rc);
rc := msLoggerAddGyr(logger:=logger[2], downsample:=0, lowres:=FALSE);
DebugFmt(message:="msLoggerAddGyr (rc=\1) downsample=0, lowres=FALSE",
v1:=rc);

watermark.logger := logger[1];
watermark.level := 400;
eventid[1] := msEventRegister(event:=ADDR(watermark));
IF eventid[1] > 0 THEN
   DebugFmt(message:="Success, watermark event id is \4", v4:=eventid[1]);
ELSE
   DebugFmt(message:="msEventRegister  (rc=\4)", v4:=eventid[1]);
END_IF;
```

```
watermark.level := 800;
switchLogger.src := logger[1];
switchLogger.dst := logger[2];
eventid[2] := msEventRegister(event:=ADDR(watermark),
action:=ADDR(switchLogger));
IF eventid[2] > 0 THEN
   DebugFmt(message:="Success, watermark event id is \4", v4:=eventid[2]);
ELSE
   DebugFmt(message:="msEventRegister  (rc=\4)", v4:=eventid[2]);
END_IF;

watermark.logger := logger[2];
watermark.level := 400;
eventid[3] := msEventRegister(event:=ADDR(watermark));
IF eventid[3] > 0 THEN
   DebugFmt(message:="Success, watermark event id is \4", v4:=eventid[3]);
ELSE
   DebugFmt(message:="msEventRegister  (rc=\4)", v4:=eventid[3]);
END_IF;
watermark.level := 800;
eventid[4] := msEventRegister(event:=ADDR(watermark));
IF eventid[4] > 0 THEN
   DebugFmt(message:="Success, watermark event id is \4", v4:=eventid[4]);
ELSE
   DebugFmt(message:="msEventRegister  (rc=\4)", v4:=eventid[4]);
END_IF;

shockcfg.threshold := 0.1;
shockid := msEventRegister(event:=ADDR(shockcfg));
IF shockid > 0 THEN
   DebugFmt(message:="Success, shock event id is \4", v4:=shockid);
ELSE
   DebugFmt(message:="msEventRegister  (rc=\4)", v4:=shockid);
END_IF;

accelcfg.threshold := 0.2;
accelid := msEventRegister(event:=ADDR(accelcfg));
IF accelid > 0 THEN
   DebugFmt(message:="Success, acceleration event id is \4", v4:=accelid);
ELSE
   DebugFmt(message:="msEventRegister  (rc=\4)", v4:=accelid);
END_IF;

rc := msLoggerStart(logger:=logger[1], reset:=TRUE);
DebugFmt(message:="msLoggerStart (rc=\1) reset=TRUE", v1:=rc);
BEGIN
   WHILE running DO
      Sleep(delay:=1000);
   END_WHILE;
   rc := msEventUnregister(event:=shockid);
   DebugFmt(message:="msEventUnregister shock (rc=1)", v1:=rc);
   rc := msEventUnregister(event:=accelid);
   DebugFmt(message:="msEventUnregister accel (rc=1)", v1:=rc);
   rc := msClose();
   IF rc = 1 THEN
      DebugMsg(message:="Done");
   ELSE
      DebugFmt(message:="msClose (rc=\1)", v1:=rc);
   END_IF;
   EXIT;
END;
END_PROGRAM;
```

4.2.30.5.5 msEventLogWatermark (Structure)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.50.00

Structure to configure a watermark event. Used with msEventRegister,

Fields:

Logger : SYSHANDLE
A handle to a valid logger.

Level : UINT (default 800)
The level in promille of how full the buffer should be to trigger the event.

Declaration:

```
STRUCT_BLOCK msEventLogWatermark;
   Logger      : SYSHANDLE;
   Level       : UINT := 800;
END_STRUCT_BLOCK;
```

4.2.30.5.6 msEventLogFull (Structure)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.50.00

Structure to configure a log full event, triggered when the specified logger is full. Used with msEventRegister,

Fields:

Logger : SYSHANDLE
A handle to a valid logger.

Declaration:

```
STRUCT_BLOCK msEventLogFull;
   Logger      : SYSHANDLE;
END_STRUCT_BLOCK;
```

4.2.30.5.7 msEventShock (Structure)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.50.00

Structure to configure detection of shock events. Used with msEventRegister,

A shock is defined as the measured acceleration exceeding a threshold and returning below the threshold before the duration limit expires.
A double shock is defined as two shocks detected after each other where the second shock must start within a time window after the first has ended.

If a shock (or double shock) is detected, a shock event is raised. Use msWaitEvent and msReadEvent to receive the events.

Note: The threshold should be below the full scale resolution set with msAccEnable.

Fields:

Threshold : FLOAT [0.0 .. 16.0] (default 0.3)
The acceleration which each axis is compared against to detect the start and end of a shock (in mg).

Duration : DINT [1..5120] (default 100)
The time in milliseconds before which the end of a shock must occur.

Latency : DINT [0..10240] (default 100)
Number of milliseconds where shocks are ignored.

Window : DINT [0..10240] (default 0)
For double shock detection. Number of milliseconds before the second shock must occur.
Set to zero (0) for single shock detection.

X : BOOL (default TRUE)
Detects shocks on x-axis.

Y : BOOL (default TRUE)
Detects shocks on y-axis.

Z : BOOL (default TRUE)
Detects shocks on z-axis.

Once : BOOL (default FALSE)
If TRUE, only one event will occur based on this configuration.

Declaration:

```
STRUCT_BLOCK msEventShock;
   Threshold   : FLOAT := 0.3;
   Duration    : DINT  := 100;
   Latency     : DINT  := 100;
   Window      : DINT  := 0;
   X           : BOOL  := TRUE;
   Y           : BOOL  := TRUE;
   Z           : BOOL  := TRUE;
   Once        : BOOL  := FALSE;
END_STRUCT_BLOCK;
```

4.2.30.5.8 msEventAcceleration (Structure)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.50.00

Structure to configure detection of acceleration events. Used with msEventRegister,

If the acceleration is above (or below) the threshold for the selected duration, an acceleration event is raised.
The accelerometer will continue to raise the event after each duration interval until the acceleration moves below (or above) the threshold again.

Use msWaitEvent and msReadEvent to receive the events.

Note: The threshold should be below the full scale resolution set with msAccEnable.

Fields:

Threshold : FLOAT [0.0 .. 16.0] (default 0.3)
The acceleration which each axis is compared against (in g).

Duration : DINT [1..5120] (default 100)
Number of milliseconds the axis must be high/low before the axis condition is true.

Combined : BOOL (default FALSE)
Whether all axis conditions should be true or just a single one before the event is raised.

XAbove : BOOL (default TRUE)
Fire event when x-axis is above threshold for the duration.

XBelow : BOOL (default FALSE)
Fire event when x-axis is below threshold for the duration.

YAbove : BOOL (default TRUE)
Fire event when y-axis is above threshold for the duration.

YBelow : BOOL (default FALSE)
Fire event when y-axis is below threshold for the duration.

ZAbove : BOOL (default TRUE)
Fire event when z-axis is above threshold for the duration.

ZBelow : BOOL (default FALSE)
Fire event when z-axis is below threshold for the duration.

Declaration:

```
STRUCT_BLOCK msEventAcceleration;
   Threshold   : FLOAT := 0.3;
   Duration    : DINT  := 100;
   Combined    : BOOL  := FALSE;
   XAbove      : BOOL  := TRUE;
   XBelow      : BOOL  := FALSE;
   YAbove      : BOOL  := TRUE;
   YBelow      : BOOL  := FALSE;
   ZAbove      : BOOL  := TRUE;
   ZBelow      : BOOL  := FALSE;
END_STRUCT_BLOCK;
```

4.2.30.5.9 msActionStopLogger (Structure)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.50.00

Structure to add the action of stopping a logger to an event. See msEventRegister.

After the configured event is raised, the specified logger is stopped.
(In case the specified logger is not active, nothing will happen.)

Fields:

Logger : SYSHANDLE
A handle to a valid logger.

Declaration:

```
STRUCT_BLOCK msActionStopLogger;
   Logger      : SYSHANDLE;
END_STRUCT_BLOCK;
```

4.2.30.5.10 msActionStartLogger (Structure)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.50.00

Structure to add the action of starting a logger to an event. See msEventRegister.

After the configured event is raised, the specified logger is started.
(In case a logger is already active, nothing will happen.)

Fields:

Logger : SYSHANDLE
A handle to a valid logger.

Reset : BOOL (default TRUE)
Reset the logger. The logger will be in the same state as when it was newly created.

Declaration:

```
STRUCT_BLOCK msActionStartLogger;
   Logger      : SYSHANDLE;
   Reset       : BOOL := TRUE;
END_STRUCT_BLOCK;
```

4.2.30.5.11 msActionSwitchLogger (Structure)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.50.00

Structure to add the action of switching loggers to an event. See msEventRegister.
After the configured event is raised, the specified loggers are switched.

This action ensures that no sensor measurements are lost in transition, as the combined actions of stopping a logger and starting another logger in succession will result in loss of sensor measurements.

Note: The newly started logger and the stopped logger must use the same set of sensors. Other settings, such as f.ex. downsampling do not need to be kept.

Fields:

Src : SYSHANDLE
A handle to a valid logger. The logger to stop.

Dst : SYSHANDLE
A handle to a valid logger. The logger to start.

Reset : BOOL (default TRUE)
Reset the logger started. This logger will be in the same state as when it was newly created.

Declaration:

```
STRUCT_BLOCK msActionSwitchLogger;
   Src          : SYSHANDLE;
   Dst          : SYSHANDLE;
   Reset        : BOOL := TRUE;
END_STRUCT_BLOCK;
```

4.2.30.5.12 msReadWatermark (Structure)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.50.00

Structure to get the returned information from a watermark event. Used with msReadEvent.

Fields:

Level : UINT
The level in promille of how full the buffer was when the event was triggered.
This can deviate quite some from the configured watermark, especially if the buffer is small.

Declaration:

```
STRUCT_BLOCK msReadWatermark;
   Level        : UINT;
END_STRUCT_BLOCK;
```

4.2.30.5.13 msReadShockEvent (Structure)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.50.00

Structure to get the returned information from a shock event. Used with msReadEvent.

Fields:

Time : DINT
The timestamp given when the event was triggered.

X : BOOL
True if shock was detected on x-axis.

X : BOOL
True if shock was detected on y-axis.

X : BOOL
True if shock was detected on z-axis.

Declaration:

```
STRUCT_BLOCK msReadShockEvent;
   Time         : DINT;
   X            : BOOL;
   Y            : BOOL;
   Z            : BOOL;
END_STRUCT_BLOCK;
```

4.2.30.5.14 msReadAccelEvent (Structure)

Architecture: NX32L
Device support: NX-200, LX2
Firmware version: 1.50.00

Structure to get the returned information from an acceleration event. Used with msReadEvent.

Fields:

Time : DINT
The timestamp given when the event was triggered.

AccX : FLOAT
Last x-axis reading when the event was triggered (mg).

AccY : FLOAT
Last y-axis reading when the event was triggered (mg).

AccZ : FLOAT
Last z-axis reading when the event was triggered (mg).

XAbove : BOOL
Was x-axis above threshold for the duration.

Xbelow : BOOL
Was x-axis below threshold for the duration.

YAbove : BOOL
Was y-axis above threshold for the duration.

Ybelow : BOOL
Was y-axis below threshold for the duration.

ZAbove : BOOL
Was z-axis above threshold for the duration.

Zbelow : BOOL
Was z-axis below threshold for the duration.

Declaration:

```
STRUCT_BLOCK msReadAccelEvent;
   Time        : DINT;
   AccX        : FLOAT;
   AccY        : FLOAT;
   AccZ        : FLOAT;
   XAbove      : BOOL;
   XBelow      : BOOL;
   YAbove      : BOOL;
   YBelow      : BOOL;
   ZAbove      : BOOL;
   ZBelow      : BOOL;
END_STRUCT_BLOCK;
```

4.2.30.6 Power management

4.2.30.6.1 msVibrationSetWakeup (Function)

```
Architecture:       NX32L
Device support:     NX-200, LX2
Firmware version:   1.36.00
```

Configure the system to wake from suspend on detection of vibration. See pmSuspend.

Note:
The Motion Sensor interface must be opened with msOpen before calling this function.
To wake on vibration without calling msOpen, see accVibrationSetWakeup.

All of the motion sensors must be disabled to be able to wake on vibration, see msAccEnable, msAccEnable and msMagEnable.

pmSuspend return codes for this wake-up source:

Error	Type	Value	Description
False	4	1	Vibration has woken up the system.
True	4	-10	Sensitivity is 0.
True	4	-11	Accelerometer is open, preventing vibration from working. Call accClose to close it.
True	4	-12	MS is open and accelerometer, gyroscope or magnetometer is active.

Input:

Enable : BOOL
Controls if enabling or disabling wake-up on vibration.

Sensitivity : SINT (-1..100, default -1)
The sensitivity level of the vibration detection. 0 is no vibration detection, 100 is most sensitive.
-1 is to use the sensitivity used in normal operation.
See pmSetVibrationSensivity

To use vibration as a wake-up source, the sensitivity must not be 0.

Returns: INT

1- Success.
0- This function is not supported.
-1- Interface is not open. Call msOpen first.
-2- Generic error.
-3- Invalid input configuration.
-9- Logger is active.
-14- Accelerometer, gyroscope or magnetometer is active.

Declaration

```
FUNCTION msVibrationSetWakeup : INT;
VAR_INPUT
   Enable      : BOOL;
   Sensitivity : SINT := -1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
// These are the local variables of the program block
VAR
   wk    : DINT;
END_VAR;
// The next code will only be executed once after the program starts
   ...
   // Open Motion Sensor interface
   msOpen();
   ...
   // Enable wake on vibration with high sensitivity
   msVibrationSetWakeup(sensitivity := 80, enable := ON);
   ...
   // Sleep 600 seconds or wake on vibration detection
   wk := pmSuspend(time:=600, mode := 0);
```

```
    ...
    // Close Motion Sensor interface
    msClose();
    ...
BEGIN
// Code from this point until END will be executed repeatedly
END;
END_PROGRAM;
```

4.2.30.7 Calculation

4.2.30.7.1 msCompassHeading (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.52.00

This function takes the measured magnetometer readings in the plane perpendicular to the axis passing through the center of the earth and the RTCU device and gives a measure for magnetic north, also called Compass Heading.
Returned value is in degrees.

The measured compass heading follows the rules:

- If the result is greater than 337.25° or less than 22.5° – North
- If the result is between 22.5° and 67.5° – North-East
- If the result is between 67.5° and 112.5° – East
- If the result is between 112.5° and 157.5° – South-East
- If the result is between 157.5° and 202.5° – South
- If the result is between 202.5° and 247.5° – South-West
- If the result is between 247.5° and 292.5° – West
- If the result is between 292.5° and 337.25° – North-West

Input:

MagX : FLOAT
The measured magnetometer reading on the X-axis.

MagY : FLOAT
The measured magnetometer reading on the Y-axis.

Returns: FLOAT

The measure for the compass heading in the range [0.0° - 360.0°[

Declaration:

```
FUNCTION msCompassHeading : FLOAT;
VAR_INPUT
    MagX : FLOAT;
    MagY : FLOAT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
    val     : msDataSet;
    heading : FLOAT;
END_VAR;
```

```
msOpen();
msMagEnable(enable := TRUE);

BEGIN
  msRead(data := val);
  heading := msCompassHeading(MagX := val.MagX, MagY := val.MagY);
  DebugMsg(message:="Heading is "+floatToStr(v:=heading));
  ...
END;
END_PROGRAM;
```

4.2.30.7.2 msVectorToAngles (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.50.00

This function will take the applied values and calculate the angles of the axis relative to acceleration forces.

When mode is ABSOLUTE (see msAccEnable) and not moving, the angles returned will represent the tilt of each axis relative to the earth surface.

Input:

X : FLOAT
The acceleration force measured on the X-axis.

Y : FLOAT
The acceleration force measured on the Y-axis.

Z : FLOAT
The acceleration force measured on the Z-axis.

Output:

Roll : FLOAT
Angle of the X-axis relative to acceleration forces in 0.01 degrees.

Pitch : FLOAT
Angle of the Y-axis relative to acceleration forces in 0.01 degrees.

Yaw : FLOAT
Angle of the Z-axis relative to acceleration forces in 0.01 degrees.

Returns: INT

1 - Success.
0 - This function is not supported.

Declaration:

```
FUNCTION msVectorToAngles : INT;
VAR_INPUT
   X       : FLOAT;
   Y       : FLOAT;
   Z       : FLOAT;
   Roll    : ACCESS FLOAT;
   Pitch   : ACCESS FLOAT;
   Yaw     : ACCESS FLOAT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   val   : msDataSet;
   roll  : FLOAT;
   pitch : FLOAT;
   yaw   : FLOAT;
END_VAR;

msOpen();
msAccEnable(enable := TRUE, mode := 2);

BEGIN
  msRead(data := val);
  msVectorToAngles(x := val.AccX, y := val.AccY, z := val.AccZ, roll := roll,
pitch := pitch, yaw := yaw);
  ...
END;
END_PROGRAM;
```

4.2.30.7.3 msVectorToForce (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.50.00

This function will take the applied values and calculate the total force.
Returned values are in G.

Input:

X : FLOAT
The acceleration force measured on the X-axis.

Y : FLOAT
The acceleration force measured on the Y-axis.

Z : FLOAT
The acceleration force measured on the Z-axis.

Returns: FLOAT

The total acceleration force.

Declaration:

```
FUNCTION msVectorToForce : FLOAT;
VAR_INPUT
   X    : FLOAT;
   Y    : FLOAT;
   Z    : FLOAT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   val   : msDataSet;
   force : FLOAT;
```

```
END_VAR;

msOpen();
msAccEnable(enable := TRUE);

BEGIN
  msRead(data := val);
  force := msVectorToForce(x := val.AccX, y := val.AccY, z := val.AccZ);
  ...
END;
END_PROGRAM;
```

4.2.31 nav: Navigation

4.2.31.1 nav: Navigation

The Navigation API supports the implementation of advanced fleet management applications that integrate navigation, two-way messaging, driver identification, and Job dispatch capabilities.
The Job dispatch functionality includes advanced features such as support for routes with multiple stops, route optimization, and periodic Estimated Time of Arrival (ETA) notifications.

To use the Navigation interface, it is necessary to connect a supported Navigation device from Garmin or a PNM device from Logic IO.
The connection between the Garmin/PNM Navigation device is established through a serial port with a special interface cable available from Logic IO.

To enable a quick and easy overview of what functionality is supported by the navigation devices, the Navigation API is divided into levels.
Each level of the API is a group of functionality that is added to the functionality of the previous level(s). For example, a device with level 1 API support can only use the basic navigation functionality while a device with level 2 API support can use both the basic navigation functionality and the more advanced functionality that the level 2 API offers.
For each function in the navigation, their API level is documented.

For the latest list of devices that support the Garmin Navigation API, please visit this link:
https://www8.garmin.com/solutions/mobile-resource-management/supported-devices/
Please also contact Logic IO for further details.

All devices supports level 2 of the navigation API, which includes the following functionality:

- User authentication.
- 3 simultaneous users.
- Waypoints.
- Message threading.
- Deleting messages.
- Message formatting using HTML tags.

NX32 and NX32L execution architecture devices also have support for the following levels:

API Level 8:
- Speed Limit Alerts

API Level 9:
- Remote Reboot

API Level 12:
- Custom Forms

API Level 13:
- Avoidance areas

API Level 14:
- Routes

API Level 16:
- Support for 57600 baud.

API Level 17:
- Alert Pop-up
- Sensor Display

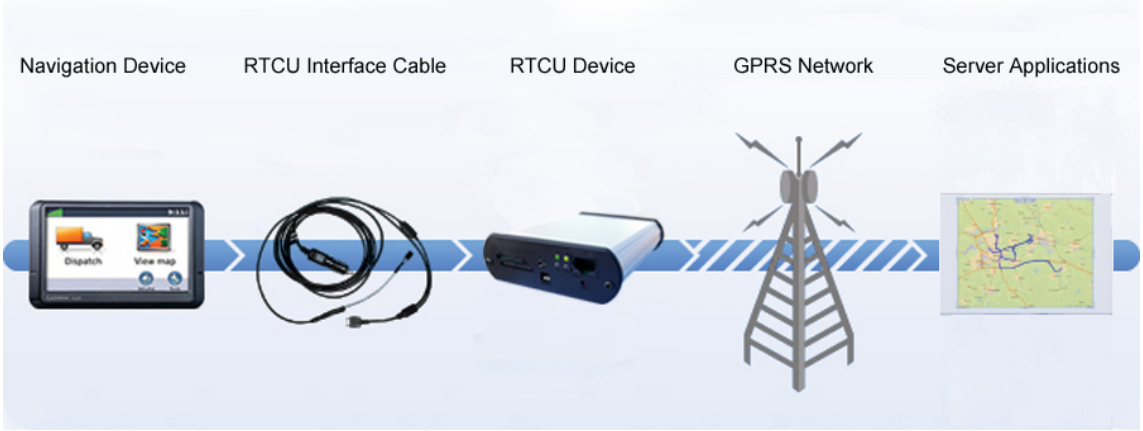
API Level 21:
- Show custom forms.

API Level 23:
- ETA mode

The specific API level supported depends on the specific Garmin device connected and only the latest devices supports all levels.

It is possible to determine the functionality available on a device by calling the navGetAPILevel function which returns the API level supported by the connected device.
Using a functionality not supported by the connected device will fail with an error code specifying that the feature is unsupported.

The illustration below shows the system architecture with the various components constituting a complete Fleet Management Solution:



For easy use and access, the Navigation API has been grouped into 12 logical categories:

- Navigation Management.
- User Identification.
- Text Messaging.
- Route Management.
- Estimated Time of Arrival.
- Garmin Point of Interest.
- Waypoint management.
- Sensor display.
- Avoidance areas.
- Custom forms
- Speed Limit Alerts
- Conversion.

Navigation Management

This group is used for managing the navigation interface.

navOpen	Opens the navigation interface.
navClose	Closes the navigation interface.
navPresent	Queries whether a navigation device is present.
navVersion	Gets the version number of the software on the navigation device.

- navGetAPILevel — Get the API level that a connected navigation device supports.
- navDeviceSerial — Gets the serial number of a connected navigation device.
- navWaitEvent — Waits for an event from the navigation interface.
- navSetUIText — Customizes the text of the user interface in the navigation device.
- navDeleteData — Deletes data from the navigation device.
- navFix — Gets the current GPS position from the connected navigation device.
- navRemoteReboot — Reboots the navigation device.
- navSafeMode — Controls safe mode, where some menus are disabled while driving.

User Identification

This group is used for maintaining the user ID and status with the user being the person operating the navigation device.

- navUserIDAuthenticate — Authenticates a received user ID.
- navUserIDReceive — Reads a received user ID.
- navUserIDReceiveX — Reads a received user ID with extra information.
- navUserIDRequest — Requests the current user ID.
- navUserIDSet — Sets the user ID in the navigation device.
- navUserStatusDefine — Defines a user status item.
- navUserStatusDelete — Deletes a user status item.
- navUserStatusReceive — Reads a received user status.
- navUserStatusRcvX — Reads a received user status with extra information.
- navUserStatusRequest — Requests the current user status.
- navUserStatusSet — Sets the user status in the navigation device.

Text messaging

This group is used for sending and receiving simple text messages to/from the navigation device. Quick messages are predefined text messages in the navigation device so the user does not have to repeatedly type in common text messages.
User-defined replies allow the user to reply with something other than yes/no without sending a text message.

- navMessageSend — Sends a text message to the navigation device.
- navMessageDelete — Deletes a text message in the navigation device.
- navMessageStatusReceive — Reads a received text message status.
- navMessageStatusRequest — Requests the status of a text message.
- navMessageReceive — Reads a received text message.
- navMessageReceiveX — Reads a received text message with extra information.
- navMessageQuickDefine — Defines a quick message in the navigation device.
- navMessageQuickDelete — Deletes a quick message in the navigation device.
- navMessageReplyDefine — Adds a user-defined reply in the navigation device.
- navMessageReplyDelete — Removes a user-defined reply from the navigation device.
- navMessageReplyReceive — Reads a received text message reply.
- navPopupAlert — Shows an alert message on the navigation device.

Route Management

This group is used for maintaining the destinations and the order of the destinations in the route.

- navStopSet — Adds a destination to the route.
- navStopReceive — Reads a received destination status.
- navStopRequest — Requests the status of a destination.
- navStopSort — Sorts destinations by shortest path.
- navStopIndexSet — Sets the position of a destination in the route.
- navStopSetActive — Activates a destination.
- navStopSetDone — Marks a destination as completed.
- navStopDelete — Deletes a destination from the route.
- navAutoArrival — Sets the criteria for the auto-arrival detection.
- navStopRouteCreate — Begins the construction of a new route.
- navStopRouteAbort — Cancels the construction of a route.
- navStopRouteAddPoint — Adds a stop or a via point to a route.
- navStopSetRoute — Sends the route to the navigation device.
- navStopSetRouteFile — Sends a route from a file to the navigation device.

Estimated Time of Arrival (ETA)

This group is used for handling the ETA updates.

- navETAReceive — Reads a received ETA update.
- navETARequest — Requests an ETA update.
- navETAAutoSet — Enables or disables ETA events at regular intervals.
- navETASetMode — Configure which ETA events to report.

File transfers

This group is used for handling file transfers.

- navGPITransfer — Starts to transfer a GPI file.
- navGPIProgressReceive — Reads a transfer progress report.
- navGPIRequestID — Requests the ID of the GPI file.
- navGPIReceiveID — Reads a received GPI file ID.

Waypoint management

This group is used for maintaining waypoints with waypoints being user-defined points of interest.

- navWaypointSet — Sets a waypoint in the navigation device.
- navWaypointDelete — Deletes a waypoint from the navigation device.
- navWaypointDeleteByCat — Deletes all waypoints that are part of the categories.

Sensor display

This group is used for displaying sensor data to the user. The sensor data can come from any source and does not necessarily have to be from an actual sensor.

- navSensorConfig — Create or modify a sensor.
- navSensorUpdate — Set the status of a sensor.
- navSensorDelete — Delete a sensor.

Avoidance areas

This group is used for configuring areas to avoid while routing.

- navAvoidEnable — Enable or disable the support for avoidance areas.
- navAvoidAreaCreate — Create or modify an avoidance area.
- navAvoidAreaDelete — Delete an avoidance area.
- navAvoidAreaEnable — Enable or disable an avoidance area.

Custom forms

This group is used for creating custom forms on the navigation device. These forms can then be filled out by the user on the navigation device and then be sent back to the RTCU. For more details, see Custom forms.

Speed Limit alerts

This group is used for configuring and reading Speed Limit Alerts, which are events that happens when the vehicle exceeds the speed limit.

- navSpeedLimitAlertSetup — Configures the Speed Limit Alert support.
- navSpeedLimitAlert — Reads a Speed Limit Alert.

Conversion

This group is used for converting data formats between the ones that are used in the navigation interface and the ones that are used in VPL.

- navPositionToSemicircles — Converts position format to semicircles.
- navSemicirclesToPosition — Converts position format from semicircles.

Architecture of the Navigation API

The Navigation API is based on an asynchronous event-based model where solicited as well as unsolicited events from the Navigation device are automatically queued and delivered to the user application. The function navWaitEvent is used to return information about the various events that are arriving from the Navigation device. This model eliminates the requirement for the application to continuously poll for messages, and it also encourages the use of multithreading with the added benefits that come with this.
Events that are triggered by the application requesting various pieces of information from the Navigation device are therefore called *solicited* events. They can also be triggered by the Navigation itself, and if such is the case, they are instead called *unsolicited* events. A solicited event is for example when the application calls navStopRequest. navWaitEvent will then return with event# 5, and the application must use navStopReceive to read the actual stop status. An unsolicited event works exactly the same way as a solicited event, except that navStopRequest is not called by the application.
Please consult navWaitEvent for detailed information about the various events and how they are handled.

Using Unicode strings with the Navigation API

Support for Unicode strings are included in the navigation API, which means that text (messages, user IDs, etc.) can be in the local language rather than in English. For more information, please refer to navOpen.

The navigation interface has the following limitations:

- The number of user statuses is limited to 16.
- The number of quick messages is limited to 120.
- The number of user-defined replies is limited to 200.
- The number of destinations is limited to 20.

4.2.31.2 Navigation Management

4.2.31.2.1 navOpen (Function)

Architecture:	**X32 / NX32 / NX32L**
Device support:	**MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5**
Firmware version:	**1.40 / 1.00.00**
Nav. API level:	**1**

This function will open the navigation interface. Calling this function is required before the navigation interface can be used.
When calling this function, it is possible to specify the serial port (0/1) used for communication with the Navigation device.
Support for Navigation must be enabled in the RTCU device. Otherwise an error code will be returned when navOpen is called.
By using boardGetProfileX, it can be checked whether Navigation is supported.
Also see navClose for closing the navigation interface after having finished the operations.

When using the NMP, please also see nmpPower.

Using Unicode strings

Support for Unicode strings are included in the Navigation API so that text (messages, user IDs, etc.) can be in the local language rather than in English.
The problem with Unicode strings is that no other API in the RTCU supports Unicode directly. This means that any function that takes a STRING parameter, outside of the navigation functions, cannot use a string received from the navigation device - except strFromMemory and strToMemory.
This limits handling of text from the navigation device in the RTCU to being sent to a server. Sending (and receiving) Unicode must be done as binary data - for example by using gsmSendPDU or gwSendPacket.
While it is not possible to enter Unicode text directly into a string variable, it can be done as a string of special characters (see STRING).
This approach requires 3 steps for each character:
1. Find the character at the Unicode homepage (www.unicode.org).
2. Encode the character into UTF-8 (Wikipedia gives a good description of this here: en.wikipedia.org/wiki/Utf-8).
3. Type in the special characters from step 2 (e.g. "CF86" for the Greek character "phi").

Unicode is controlled by the "Unicode" parameter specified with the call to navOpen.

Input:

port : SINT (0/1, *Default: 1*)
The serial port where the navigation device is connected (see serial port enumeration).

unicode : BOOL (*Default: OFF*)
ON: Enables Unicode strings.
OFF: Disable Unicode strings.

baud : DINT (0,9600,19200,38400,57600, *Default: 9600*)
The baud rate to communicate with. A value of 0 will be changed to 9600 baud.
Garmin devices that support API Level 16 can use both 9600 and 57600 baud, otherwise only 9600 baud is supported. If 57600 baud has been used on a Garmin device, it must be disconnected from power to restore the baud rate to 9600, e.g by removing it from the mount.

Returns: INT

0 - Success.
-1 - Navigation interface is already open.
-6 - Error opening serial port.
-11 - Navigation interface not enabled in device.

Declaration:

```
FUNCTION navOpen : INT;
VAR_INPUT
   port    : SINT := 1;
   unicode : BOOL := OFF;
   baud    : DINT := 9600;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;
VAR
   rc : INT;
END_VAR;

   DebugMsg(message := "Initializing navigation...");
   rc := navOpen();
   IF rc <> 0 THEN
      DebugFmt(message := "Error: navOpen=\1", v1 := rc);
   END_IF;

   ...

   // Close the navigation interface
   navClose();
   ...
END_PROGRAM;
```

4.2.31.2.2 navClose (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 1.40 / 1.00.00
Nav. API level: 1

This function will close the navigation interface. After the navigation interface is closed, it cannot be used until opened again.

Please also see navOpen.

Input:
None.

Returns: **INT**
0 - Success.
-1 - Navigation interface is not opened.

Declaration:
```
FUNCTION navClose : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;
VAR
   rc : INT;
END_VAR;

   DebugMsg(message := "Initializing navigation...");
   rc := navOpen();
   IF rc <> 0 THEN
      DebugFmt(message := "Error: navOpen=\1", v1 := rc);
   END_IF;

   ...

   // Close the navigation interface
   navClose();
   ...
END_PROGRAM;
```

4.2.31.2.3 navPresent (Function)

Architecture:	**X32 / NX32 / NX32L**
Device support:	**MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5**
Firmware version:	**1.40 / 1.00.00**
Nav. API level:	**1**

This function will determine if a navigation device is present.

Input:
None.

Returns: **INT**
TRUE: If navigation device is present.
FALSE: If navigation device is not present.

Declaration:
```
FUNCTION navPresent : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   // Check if navigation device is present
   IF navPresent() THEN
      DebugMsg(message := "Navigation device (" + navDeviceSerial() + ")
present");
      ...
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.31.2.4 navVersion (Function)

Architecture:	**X32 / NX32 / NX32L**
Device support:	**MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5**
Firmware version:	**2.62 / 1.00.00**
Nav. API level:	**1**

This function will return the version number of the software in the navigation device.

Input:

None.

Returns: INT

Version number of the firmware in the navigation device scaled by 100. Version 3.00 will be returned as 300.

0 - Navigation device not present.
-1 - Navigation interface is not open.

Declaration:

```
FUNCTION navVersion : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Print version
   DebugFmt(message := "Version: \1.\2" , v1:=( navVersion()/100),
v2:=(navVersion() MOD 100));
   ...
END;
END_PROGRAM;
```

4.2.31.2.5 navGetAPILevel (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 2.20 / 1.00.00
Nav. API level: 1

This function will determine the maximum navigation API level that is supported by the navigation device.

Note: Some devices that report support for level 8 and above may not support every level below. Check the return code for the specific functions to see if the feature is supported.

Input:

None.

Returns: INT

The navigation API level supported by navigation device.

0 - Navigation device not present.
-1 - Navigation interface is not open.

Declaration:

```
FUNCTION navGetAPILevel : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;
VAR
   rc : INT;
END_VAR;

   DebugMsg(message := "Initializing navigation...");
   navOpen();
   ...

BEGIN
   ...
   rc := navGetAPILevel();
   IF rc = 2 THEN
      // Waypoints are only awailable in navigation API level 2
      navWaypointSet(ID := 5, symbol := 18, latitude := 666377598, longitude
:= 117525954, categories := 16#1000, name := "Logic IO", comment := "Main
office");
   END_IF;
   ...
END_PROGRAM;
```

4.2.31.2.6 navDeviceSerial (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 1.50 / 1.00.00
Nav. API level: 1

This function will return the serial number of the connected navigation device. The serial number is unique for each navigation device.

Input:

None.

Returns: STRING

The serial number of the connected navigation device.

Declaration:

```
FUNCTION navDeviceSerial : STRING;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   // Check if navigation device is present
   IF navPresent() THEN
      DebugMsg(message := "Navigation device (" + navDeviceSerial() + ")
present");
      ...
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.31.2.7 navWaitEvent (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 1.40 / 1.00.00
Nav. API level: 1

This function will wait until an event is raised with an optional timeout.

An event is a notification that something has happened in the navigation interface.
There are 2 types of events:

1. Events triggered by the application by a request for information (*solicited event*).
2. Events triggered by the Navigation device (*unsolicited event*).

The events are queued until appropriate action is taken as described in this table:

Event#	Event	Solicited	Action
1	A reply for a text message is received.	No.	Call navMessageReplyReceive.
2	A text message is received.	No.	Call navMessageReceive.
3	The status for a text message is received.	Yes.	Call navMessageStatusReceive.
4	An ETA update is received.	Yes.	Call navETAReceive.
5	The status af a destination is received.	Yes.	Call navStopReceive.
6	A user ID is received.	Yes.	Call navUserIDReceive.
7	The status of a user is received.	Yes.	Call navUserStatusReceive.
8	The ID of the GPI is received.	Yes.	Call navGPIReceiveID.
9	The progress of the file transfer is updated.	No.	Call navGPIProgressReceive.
10	The progress of the NMP update is updated.	No.	Call nmpUpdateProgressReceive.
11	A button has been pressed on the NMP.	No.	Call nmpButtonPressedReceive.
12	A button has been clicked in an NMP dialog.	No.	Call nmpDialogClickReceive.
13	A hardware button has been pressed on the NMP.	No.	Call nmpHardwareButtonPressedReceive.
14	A submitted custom form has been received.	No.	Call navFormReceive and navFormReceiveDone.
15	A Speed Limit Alert is received.	No.	Call navSpeedLimitAlert.
129	A request to refresh the quick messages.	No.	*Optional*: Refresh quick-messages
130	A request to refresh the message replies.	No.	*Optional*: Refresh message replies.
131	A request to refresh the user status list.	No.	*Optional*: Refresh user status.
132	Connection to navigation device established.	No.	
133	Connection to navigation device lost.	No.	

navWaitEvent notifies the application of the oldest queued event.
This means that if the appropriate action is not taken, navWaitEvent will continue to report the same event.

Note: waiting for an event by using this function will block the calling thread.

Input:

timeout : INT (-1,0..32000) ***(default -1)***
Timeout period in milliseconds to wait.

- 0 - Disable timeout. The function will return immediately.
- -1 - Wait forever. The function will only return when data is received.

Returns: INT

- -1 - Navigation interface is not open.
- 0 - Timeout.
- 1 - A message reply is received.
- 2 - A message is received.
- 3 - A message status is received.
- 4 - An ETA is received.
- 5 - A stop status is received.
- 6 - A user ID is received.
- 7 - A user status is received.

8 - The ID of the GPI is received.
9 - The GPI transfer progress is updated.
10 - The NMP update progress is updated.
11 - A button has been pressed on the NMP.
12 - A button has been clicked in an NMP dialog.
13 - A hardware button has been pressed on the NMP.
129 - A request to refresh the quick message list is received.
130 - A request to refresh the message reply list is received.
131 - A request to refresh the user status list is received.
132 - Connection to a navigation device is established.
133 - Connection to the navigation device lost.

Declaration:

```
FUNCTION navWaitEvent : INT;
VAR_INPUT
   timeout : INT := -1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

THREAD_BLOCK navMonitor;
VAR
   event : INT := 0;
END_VAR;

WHILE event <> -1 DO
   event := navWaitEvent(timeout := -1);
   CASE event OF
      1: ReadMessageReply();
      2: ReadMessage();
      3: ReadMessageStatus();
      4: ReadETA();
      5: ReadStopStatus();
      6: ReadUserID();
      7: ReadUserStatus();
      129: DebugMsg(message := "Request for refresh of quick messages");
      130: DebugMsg(message := "Request for refresh of message replies");
      131: DebugMsg(message := "Request for refresh of user status");
      132: DebugMsg(message := "Navigation device present");
      133: DebugMsg(message := "Navigation device not present");
   ELSE
      DebugFmt(message := "navWaitEvent - event=\1", v1 := event);
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;
```

4.2.31.2.8 navSetUIText (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 1.40 / 1.00.00
Nav. API level: 1

This function is used to customize the text of certain Fleet Management user interface elements.

Input:

ID : INT

The ID of the user interface element.

0	- The "Dispatch" menu.
100 - 105	- The text fields above the map. Only available on NMP v3.10 and above, see the NMP User Guide for more details.

text : STRING

The new text of the menu element. Max. 49 characters.

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-4	- The navigation device rejected the command.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navSetUIText : INT;
VAR_INPUT
   ID   : INT;
   text : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Check if navigation device is present
   IF navPresent() THEN
      navSetUIText(id := 0, text := "Logic IO");
      ...
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.31.2.9 navDeleteData (Function)

Architecture:	**X32 / NX32 / NX32L**
Device support:	**MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5**
Firmware version:	**1.40 / 1.00.00**
Nav. API level:	**1**

This function will delete a group of data from the navigation device.

Input:

type : DINT

The type of data to delete from the navigation device.

0	- Deletes all destinations.
1	- Deletes all messages.

2 - Deletes the active route.
3 - Deletes all quick messages.
4 - Deletes all message replies.
5 - Deletes GPI file.
6 - Deletes user ID and status.
8 - Deletes all waypoints.
10 - Deletes all custom forms.
11 - Deletes all custom avoidance areas.
12 - Deletes all sensor displays.

Returns: INT

0 - Success.
-1 - Navigation interface is not open.
-2 - Error communicating with navigation device.
-12 - Navigation interface is busy.

Declaration:

```
FUNCTION navDeleteData : INT;
VAR_INPUT
   type : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Delete data from the navigation device
   navDeleteData(type := 1);
   ...
END;
END_PROGRAM;
```

4.2.31.2.10 navFix (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 2.60 / 1.00.00
Nav. API level: 1

The navFix function block is used to query the current position and time from the connected navigation device.
The navFix works similar to the gpsFix function block except for the fact that the information comes from the connected navigation device.

This function is especially suited for the AX9 series that does not have an on-board GPS receiver to use for positioning.
All values returned by navFix() are in the WGS-84 datum.

Note: this function is not supported on the NMP.

Note regarding decimal minutes:
The value returned in Decimal minutes, both *latdecmin* and *londecmin*, are assumed to be 4 digits long.

This means that for the position 55 Deg, 51.3 Min North is returned as *latdeg*=55 *latmin*=51 and *latdecmin*=3000, and the position 55 Deg, 52.0076 North is returned as *latdeg*=55 *latmin*=52 *latdecmin*=76.

Input:

None.

Output:

mode : SINT;
0 = No info available.
1 = Fix not available.
2 = 2D position fix.
3 = 3D position fix.
4 = 3D position fix with SBAS correction.

linsec : DINT
Linsec for timestamps received from the GPS receiver.
This is the same as the separate year, month, day, minute, and second but expressed as a linsec.
Also see clockLinsecToTime().

year : INT
Year (absolute).

month : SINT
Month (1..12).

day : SINT
Date (1..31).

hour : SINT
Hour (0..23).

minute : SINT
Minute (0..59).

second : SINT
Second (0..59).

latitude : DINT
Latitude expressed in a DINT.
Negative is South (ddmm.mmmm (multiplied by 10000)).

latsouth : BOOL
Direction of latitude (TRUE is South, FALSE is North).

latdeg : SINT
Latitude Degress (0..90).

latmin : SINT
Latitude Minutes (0..59).

latdecmin : INT
Latitude Decimal Minutes (0..9999).
This value is assumed to be 4 digits with the leading zeros removed (see note regarding decimal minutes above).

longitude : DINT
Longitude expressed in a DINT.
Negative is West (dddmm.mmmm (multiplied by 10000)).

lonwest : BOOL
Direction of longitude (TRUE is West, FALSE is East).

londeg : INT
Longitude Degress (0..180).

lonmin : SINT
Longitude Minutes (0..59).

londecmin : INT
Longitude Decimal Minutes (0..9999).
This value is assumed to be 4 digits with the leading zeros removed (see note regarding decimal minutes above).

height : INT
Height in meters over Mean Sea Level.

Declaration:

```
FUNCTION_BLOCK navFix;
VAR_OUTPUT
   mode         : SINT;     | 0=No info available, 1=Fix not available, 2=2D
position fix, 3=3D position fix, 4=3D pos. with SBAS
   linsec       : DINT;     | Linsec of time-stamp
   year         : INT;      | year (absolute, like 2004)
   month        : SINT;     | month (1..12)
   day          : SINT;     | date (1..31)
   hour         : SINT;     | hour (0..23)
   minute       : SINT;     | minute (0..59)
   second       : SINT;     | second (0..59)
   latitude     : DINT;     | Negative is South (ddmm.mmmm (multiplied by
10000))
   latsouth     : BOOL;     | True = South, False = North
   latdeg       : SINT;     | Degrees (0..90)
   latmin       : SINT;     | minutes (0..59)
   latdecmin    : INT;      | decimal minutes (0..9999)
   longitude    : DINT;     | Negative is West (dddmm.mmmm (multiplied by
10000))
   lonwest      : BOOL;     | True = West, False = East
   londeg       : INT;      | degrees (0..180)
   lonmin       : SINT;     | minutes (0..59)
   londecmin    : INT;      | decimal minutes (0..9999)
   height       : INT;      | Height in meters over Mean Sea Level.
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

FUNCTION ShowDecmin : STRING;
VAR_INPUT
   decmin : INT;
END_VAR;
   ShowDecmin := intToStr(v := decmin);
   WHILE strLen(str := ShowDecmin) < 4 DO
      ShowDecmin := strConcat(str1 := "0", str2 := ShowDecmin);
   END_WHILE;
```

```
END_FUNCTION;

PROGRAM test;
VAR
   pos   : navFix;
END_VAR;

navOpen();

BEGIN
   pos();
   IF pos.mode > 1 THEN
      DebugFmt(message:="Mode=\1", v1:=pos.mode);
      DebugFmt(message:="   Linsec=\4", v4:=pos.linsec);
      DebugFmt(message:="   Time=\1:\2:\3", v1:=pos.hour, v2:=pos.minute,
v3:=pos.second);
      DebugFmt(message:="   Date=\1:\2:\3", v1:=pos.year, v2:=pos.month,
v3:=pos.day);
      DebugFmt(message:="   Lat: latitude=\4", v4:=pos.latitude);
      IF pos.latsouth THEN DebugMsg(message:="   Lat: South"); ELSE
DebugMsg(message:="   Lat: North"); END_IF;
      str := strConcat(str1 := "   Lat: Deg=\1 Min=\2 Dec=", str2 :=
ShowDecmin(decmin := pos.latdecmin));
      DebugFmt(message:=str, v1:=pos.latdeg, v2:=pos.latmin);
      DebugFmt(message:="   Lon: longitude=\4", v4:=pos.longitude);
      IF pos.lonwest THEN DebugMsg(message:="   Lon: West"); ELSE
DebugMsg(message:="   Lon: East"); END_IF;
      str := strConcat(str1 := "   Lon: Deg=\1 Min=\2 Dec=", str2 :=
ShowDecmin(decmin := pos.londecmin));
      DebugFmt(message:=str, v1:=pos.londeg, v2:=pos.lonmin);
      DebugFmt(message:="   Height=\1", v1:=pos.height);
   END_IF;
END;
END_PROGRAM;
```

4.2.31.2.11 navRemoteReboot (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 9

This function will make the navigation unit reboot.
This should only be performed if it no longer responds to the user, as some user data on the navigation device could potentially be lost.

Input:
None.

Returns: **INT**

0	- Success.
-1	- Navigation interface is not opened.
-2	- Error communicating with navigation device.
-11	- Remote reboot is not supported.

Declaration:

```
FUNCTION navRemoteReboot : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Reboot the navigation unit
   navRemoteReboot();
   ...
END;
END_PROGRAM;
```

4.2.31.2.12 navSafeMode (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 6

This function is used to enable Safe Mode and configure the threshold speed in the navigation unit.
While this is enabled, the user configurable Safe Mode is removed.

If Safe Mode is enabled, the following restrictions are applied when the speed exceeds the threshold:

- The driver will be restricted from going to 'Dispatch' and 'Tools' menus .
- If the driver is browsing a page descending from the 'Dispatch' or 'Tools' menus, the driver will be taken to the main map page.
- The driver will not be able to read new stops or non-immediate text messages

Input:

speed : INT (0..2236, *default: 2236*)
The speed threshold in mm/s. The maximum value of 2236 mm/s is around 8km/hr.
Set this value to 0 to restore the user configurable safe mode.

Returns: INT

0	- Success.
-1	- Navigation interface is not opened.
-2	- Error communicating with navigation device.
-4	- The navigation device rejected the command.
-8	- Speed is outside the valid range.
-11	- Configuring safe mode is not supported.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navSafeMode : INT;
VAR_INPUT
   speed    : INT := 2236;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
```

```
PROGRAM NavigationExample;

BEGIN
   ...
   // Enable safe mode
   navSafeMode();
   ...
END;
END_PROGRAM;
```

4.2.31.3 User Identification

4.2.31.3.1 navUserIDAuthenticate (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 2.20 / 1.00.00
Nav. API level: 2

This function will authenticate the user ID that has been received from the navigation device. This function must be called when the status of navUserIDReceiveX returns 2 (user ID is ready, authentication is required).

Input:

accept : BOOL
Set to TRUE to accept the user, set to FALSE to reject the user.

Returns: **INT**

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-3	- No user ID is waiting for authentication.
-11	- This is not supported by navigation device.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navUserIDAuthenticate : INT;
VAR_INPUT
   accept : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   userID : navUserIDReceiveX;
END_VAR;

FUNCTION ReadUserID;
   userID();
   IF userID.status > 0 THEN
      DebugMsg(message := "*** user ID received ***");
```

```
      DebugFmt(message := "-time=\4", v4 := userID.time);
      DebugFmt(message := "-idx= \1", v1 := userID.index);
      DebugMsg(message := "-user=" + userID.user);
      DebugMsg(message := "-pass=" + userID.password);
   END_IF;
   IF userID.status = 2 THEN
      navUserIDAuthenticate(accept := TRUE);
   END_IF;
END_FUNCTION;

THREAD_BLOCK navMonitor;
VAR
   event : INT := 0;
END_VAR;

WHILE event <> -1 DO
   event := navWaitEvent(timeout := -1);
   CASE event OF
      ...
      6: ReadUserID();
      ...
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;
```

4.2.31.3.2 navUserIDReceive (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 1.40 / 1.00.00
Nav. API level: 1

---

navUserIDReceive returns the user ID and timestamp.

Note that the user ID is only valid after a "User ID" event is raised; the event will be removed when this function block is called.
A "User ID" event is raised when the user ID is changed in the navigation device or is requested with navUserIDRequest.

navUserIDReceive automatically authenticates a user ID when this is required.
For manual authentication, use navUserIDReceiveX and navUserIDAuthenticate.

Input:

None.

Output:

time : DINT
The timestamp when the specified user ID took effect.

user : STRING
The user ID.
If no user ID has been set, the string will be empty.

ready : BOOL
TRUE: If data is ready.
FALSE: If data is not ready.

Declaration:

```
FUNCTION_BLOCK navUserIDReceive;
VAR_OUTPUT
   time  : DINT;
   user  : STRING;
   ready : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   userID : navUserIDReceive;
END_VAR;

FUNCTION ReadUserID;
   userID();
   IF userID.ready THEN
      DebugMsg(message := "*** user ID received ***");
      DebugFmt(message := "-time=\4", v4 := userID.time);
      DebugMsg(message := "-user=" + userID.user);
   END_IF;
END_FUNCTION;

THREAD_BLOCK navMonitor;
VAR
   event : INT := 0;
END_VAR;

WHILE event <> -1 DO
   event := navWaitEvent(timeout := -1);
   CASE event OF
      ...
      6: ReadUserID();
      ...
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;
```

4.2.31.3.3 navUserIDReceiveX (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 2.20 / 1.00.00
Nav. API level: 2

navUserIDReceiveX is an extended version of navUserIDReceive and returns the information of the user ID change.

Note that the user ID is only valid after a "User ID" event is raised; the event will be removed when this function block is called.
A "User ID" event is raised when the user ID is changed in the navigation device or is requested with navUserIDRequest.

If the status indicates that the user ID requires authentication, then the navUserIDAuthenticate function must be called.
Failure to do this will result in the "User ID" event being raised in regular intervals by the navigation device.

Input:
None.

Output:
time : DINT
The timestamp when the specified user ID took effect.

index : SINT
The index of the user.

user : STRING
The ID of the user.
If no user ID has been set, the string will be empty.

password : STRING
The password of the user.

status : SINT
0- User ID is not ready.
1- User ID is ready, authentication is not required.
2- User ID is ready, authentication is required.

Declaration:
```
FUNCTION_BLOCK navUserIDReceiveX;
VAR_OUTPUT
   time     : DINT;
   index    : SINT;
   user     : STRING;
   password : STRING;
   status   : BOOL;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

VAR
   userID : navUserIDReceiveX;
END_VAR;

FUNCTION ReadUserID;
   userID();
   IF userID.status > 0 THEN
      DebugMsg(message := "*** user ID received ***");
      DebugFmt(message := "-time=\4", v4 := userID.time);
      DebugFmt(message := "-idx= \1", v1 := userID.index);
      DebugMsg(message := "-user=" + userID.user);
      DebugMsg(message := "-pass=" + userID.password);
   END_IF;
   IF userID.status = 2 THEN
      navUserIDAuthenticate(accept := TRUE);
   END_IF;
END_FUNCTION;

THREAD_BLOCK navMonitor;
VAR
   event : INT := 0;
END_VAR;
```

```
WHILE event <> -1 DO
   event := navWaitEvent(timeout := -1);
   CASE event OF
      ...
      6: ReadUserID();
      ...
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;
```

4.2.31.3.4 navUserIDRequest (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 1.40 / 1.00.00
Nav. API level: 1

This function will request the user ID from the navigation device.
A "User ID" event is raised when the user ID is received. See also navWaitEvent for event # 6.

Input:

index : SINT (1..3, *Default: 1*)
The index of the user requested.

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-8	- Illegal user index.
-11	- This is not supported by navigation device.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navUserIDRequest : INT;
VAR_INPUT
   index : SINT := 1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Request the current user ID
   navUserIDRequest(index := 2);
   ...
END;
END_PROGRAM;
```

4.2.31.3.5 navUserIDSet (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 1.40 / 1.00.00
Nav. API level: 1

This function will set the user ID of the current user.

Input:

index : SINT (1..3, *Default: 1*)
The index of the user.

user : STRING
The new ID of the user (maximum 49 characters).

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-8	- Illegal user index or illegal user ID.
-11	- This is not supported by navigation device.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navUserIDSet : INT;
VAR_INPUT
   index : SINT := 1;
   user  : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Set user ID in the navigation device
   navUserIDSet(index := 1, user := "Driver 55");
   ...
END;
END_PROGRAM;
```

4.2.31.3.6 navUserStatusDefine (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 1.40 / 1.00.00
Nav. API level: 1

This function will define the textual description corresponding to a particular user status.

If there is already a status item with the specified id, the status text is updated; otherwise the status text is added to the list.
The user status list may contain up to 16 items.

Input:

id : INT
Unique ID to identify this user status.

text : STRING
The textual description for this status. Max. 49 characters.

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-4	- The navigation device rejected the action.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navUserStatusDefine : INT;
VAR_INPUT
   id   : INT;
   text : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   navUserStatusDefine(id := 30, text := "On the way to work");
   navUserStatusDefine(id := 31, text := "At work");
   navUserStatusDefine(id := 32, text := "On the way home");
   navUserStatusDefine(id := 33, text := "At the pizza store");
   navUserStatusDefine(id := 34, text := "At home");
   ...
END;
END_PROGRAM;
```

4.2.31.3.7 navUserStatusDelete (Function)

Architecture:	**X32 / NX32 / NX32L**
Device support:	**MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5**
Firmware version:	**1.40 / 1.00.00**
Nav. API level:	**1**

This function will delete (remove) a textual description corresponding to a particular user status. The user's current status cannot be removed.

Input:

id : INT
The ID used to identify the user status in navUserStatusDefine.

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-4	- The navigation device rejected the action.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navUserStatusDelete : INT;
VAR_INPUT
   id : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Delete a user status item
   navUserStatusDelete(id := 33);
   ...
END;
END_PROGRAM;
```

4.2.31.3.8 navUserStatusReceive (Functionblock)

Architecture:	**X32 / NX32 / NX32L**
Device support:	**MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5**
Firmware version:	**1.40 / 1.00.00**
Nav. API level:	**1**

---

navUserStatusReceive returns the user status and timestamp.

If the navigation device support API level 2, additional information can be received with navUserStatusRcvX.

Note that the user status is only valid after a "User Status" event is raised; the event will be removed when this function block is called.
A "User Status" event is raised when the user status is changed in the navigation device or is requested with navUserStatusRequest.

Input:

None.

Output:

time : DINT
The timestamp when the specified user status took effect.

status : INT
The ID used to identify the user status in navUserStatusDefine.
-1 if no status is set.

ready : BOOL
TRUE: If data is ready.
FALSE: If data is not ready.

Declaration:

```
FUNCTION_BLOCK navUserStatusReceive;
VAR_OUTPUT
   time   : DINT;
   status : INT;
   ready  : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   userStatus : navUserStatusReceive;
END_VAR;

FUNCTION ReadUserStatus;
   userStatus();
   IF userStatus.ready THEN
      DebugMsg(message := "*** user status received ***");
      DebugFmt(message := "-time=\4", v4 := userStatus.time);
      DebugFmt(message := "-status=\1", v1 := userStatus.status);
   END_IF;
END_FUNCTION;

THREAD_BLOCK navMonitor;
VAR
   event : INT := 0;
END_VAR;

WHILE event <> -1 DO
   event := navWaitEvent(timeout := -1);
   CASE event OF
      ...
      7: ReadUserStatus();
      ...
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;
```

4.2.31.3.9 navUserStatusRcvX (Functionblock)

Architecture:	**X32 / NX32 / NX32L**
Device support:	**MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5**
Firmware version:	**2.20 / 1.00.00**
Nav. API level:	**2**

navUserStatusRcvX is an extended version of navUserStatusReceive and returns the user status and timestamp.

If the navigation device does not support API level 2, the function will work as navUserStatusReceive, and extended data will not be accessible.

Note that the user status is only valid after a "User Status" event is raised; the event will be removed when this function block is called.

A "User Status" event is raised when the user status is changed in the navigation device or is requested with navUserStatusRequest.

Input:

None.

Output:

time : DINT
The timestamp when the specified user status took effect.

index : INT
The index of the user.

status : INT
The ID used to identify the user status in navUserStatusDefine.
-1 if no status is set.

ready : BOOL
TRUE: If user status event is read.
FALSE: If user status event is not read.

Declaration:

```
FUNCTION_BLOCK navUserStatusRcvX;
VAR_OUTPUT
   time   : DINT;
   index  : SINT;
   status : INT;
   ready  : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   userStatus : navUserStatusRcvX;
END_VAR;

FUNCTION ReadUserStatus;
   userStatus();
   IF userStatus.ready THEN
      DebugFmt(message := "*** status for user \1 received ***", v1 :=
userStatus.index);
      DebugFmt(message := "-time=\4", v4 := userStatus.time);
      DebugFmt(message := "-status=\1", v1 := userStatus.status);
   END_IF;
END_FUNCTION;

THREAD_BLOCK navMonitor;
VAR
   event : INT := 0;
END_VAR;

WHILE event <> -1 DO
   event := navWaitEvent(timeout := -1);
   CASE event OF
      ...
      7: ReadUserStatus();
      ...
```

```
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;
```

4.2.31.3.10 navUserStatusRequest (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 1.40 / 1.00.00
Nav. API level: 1

This function will request the user status from the navigation device.
A "User Status" event is raised when the user status is received. See also navWaitEvent for event # 7.

Input:

index : SINT (1..3, *Default: 1*)
The index of the user requested.

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-8	- Illegal user index.
-11	- This is not supported by navigation device.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navUserStatusRequest : INT;
VAR_INPUT
   index : SINT := 1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Request the current user status
   navUserStatusRequest(index := 2);
   ...
END;
END_PROGRAM;
```

4.2.31.3.11 navUserStatusSet (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 1.40 / 1.00.00
Nav. API level: 1

This function will set the user status of the current user.

Input:

index : SINT (1..3, *Default: 1*)
The index of the user.

id : INT
The ID used to identify the user status in navUserStatusDefine.

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-8	- Illegal user index.
-11	- This is not supported by navigation device.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navUserStatusSet : INT;
VAR_INPUT
   index : SINT := 1;
   id    : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Set the user status in the navigation device
   navUserStatusSet(index := 1, id := 34);
   ...
END;
END_PROGRAM;
```

4.2.31.4 Text messaging

4.2.31.4.1 navMessageSend (Function)

Architecture:	**X32 / NX32 / NX32L**
Device support:	**MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5**
Firmware version:	**1.40 / 1.00.00**
Nav. API level:	**1**

This function will send a simple text message to the navigation device.
Depending on the type of the message, it is either displayed immediately (pop-up message) or a small notification is presented.

With the introduction of navigation API level 2, support for simple HTML tags for text formatting is included.
The supported tags are:

, , <h1>, <h2>, <b>, <i>, <u>, <strike>, <ol>, <ul>, <li>.

Input:

id : INT
Unique ID to identify this message.

text : STRING
The text message to send.
If using firmware 5.0 / 1.30.00 or newer and if the navigation device supports navigation API level 11, the text can be up to 1999 characters for some message types.
Otherwise the text is limited to 199 characters.

type : SINT (0..4, *default: 0*)
The type of message to send.

- 0 - Normal message.
- 1 - Pop-up message.
- 2 - Message with acknowledgment (reply ID zero). The text is limited to 199 characters.
- 3 - Message with "yes" (reply ID 1) or "no" (reply ID 2). The text is limited to 199 characters.
- 4 - Message with user-defined reply.

count : SINT (1..10)
The number of user-defined replies used.
This parameter is ignored if type is not 4.

reply1 : INT
The ID of the reply to use for reply option 1.
This parameter is ignored if type is not 4.

reply2 : INT
The ID of the reply to use for reply option 2.
This parameter is ignored if type is not 4.

reply3 : INT
The ID of the reply to use for reply option 3.
This parameter is ignored if type is not 4.

reply4 : INT
The ID of the reply to use for reply option 4.
This parameter is ignored if type is not 4.

reply5 : INT
The ID of the reply to use for reply option 5.
This parameter is ignored if type is not 4.

reply6 : INT
The ID of the reply to use for reply option 6.
This parameter is ignored if type is not 4.

reply7 : INT
The ID of the reply to use for reply option 7.
This parameter is ignored if type is not 4.

reply8 : INT
The ID of the reply to use for reply option 8.
This parameter is ignored if type is not 4.

reply9 : INT
The ID of the reply to use for reply option 9.

This parameter is ignored if type is not 4.

reply10 : INT
The ID of the reply to use for reply option 10.
This parameter is ignored if type is not 4.

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-4	- The navigation device rejected the command.
-7	- Invalid or duplicate IDs (reply and/or message).
-8	- Invalid number of replies.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navMessageSend : INT;
VAR_INPUT
   ID      : INT;
   text    : STRING;
   type    : SINT := 0;
   count   : SINT;
   reply1  : INT;
   reply2  : INT;
   reply3  : INT;
   reply4  : INT;
   reply5  : INT;
   reply6  : INT;
   reply7  : INT;
   reply8  : INT;
   reply9  : INT;
   reply10 : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Send a text message to the navigation device
   navMessageSend(id := 55, text := "Remember to pick up lunch on the way",
type := 1);
   ...
   // Ask a question on the navigation device
   navMessageSend(id := 56, text := "Did you pick up lunch?", type := 4, count
:= 3, reply1 := 20, reply2 := 21, reply3 := 22);
   ...
END;
END_PROGRAM;
```

4.2.31.4.2 navMessageDelete (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 2.20 / 1.00.00
Nav. API level: 2

This function will delete a text message from the navigation device.

Input:

id : INT
Unique ID of the message.

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-4	- The navigation device rejected the command.
-11	- This is not supported by navigation device.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navMessageDelete : INT;
VAR_INPUT
   ID : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Send a text message to the navigation device
   navMessageSend(id := 55, text := "Remember to pick up lunch on the way",
type := 1);
   ...
   // Delete the text message
   navMessageDelete(id := 56);
   ...
END;
END_PROGRAM;
```

4.2.31.4.3 navMessageStatusReceive (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 1.40 / 1.00.00
Nav. API level: 1

---

navMessageStatusReceive returns the status of a text message.

Note that the message status is only valid after a "message status" event is raised; the event will be removed when this function block is called.
A "message status" event is raised when the status of a text message changes or is requested with navMessageStatusRequest.

Input:

None.

Output:

ID : INT
The ID used to identify the text message in navMessageSend.

status : SINT
The status of the text message.
1 - Unread.
2 - Read.
3 - Deleted/not found.

ready : BOOL
TRUE: If data is ready.
FALSE: If data is not ready.

Declaration:

```
FUNCTION_BLOCK navMessageStatusReceive;
VAR_OUTPUT
   ID     : INT;
   status : SINT;
   ready  : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   msgSts : navMessageStatusReceive;
END_VAR;

FUNCTION ReadMessageStatus;
   msgSts();
   IF msgSts.ready THEN
      DebugMsg(message := "*** message status received ***");
      DebugFmt(message := "-msgid=\1", v1 := msgSts.ID);
      CASE msgSts.status OF
         1: DebugMsg(message := "-status=Unread");
         2: DebugMsg(message := "-status=Read");
         3: DebugMsg(message := "-status=Deleted/Not found");
      ELSE
         DebugFmt(message := "-status=\1", v1 := msgSts.status);
      END_CASE;
   END_IF;
END_FUNCTION;

THREAD_BLOCK navMonitor;
VAR
   event : INT := 0;
END_VAR;

WHILE event <> -1 DO
```

```
   event := navWaitEvent(timeout := -1);
   CASE event OF
      ...
      3: ReadMessageStatus();
      ...
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;
```

4.2.31.4.4 navMessageStatusRequest (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 1.40 / 1.00.00
Nav. API level: 1

This function will request the status for a text message.
A "message status" event is raised when the text message status is received. See also navWaitEvent for event # 3.

Input:

id : INT
The ID used to identify the text message in navMessageSend.

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navMessageStatusRequest : INT;
VAR_INPUT
   ID : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Request the status of a text message
   navMessageStatusRequest(id := 55);
   ...
END;
END_PROGRAM;
```

4.2.31.4.5 navMessageReceive (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 1.40 / 1.00.00
Nav. API level: 1

---

navMessageReceive returns a text message received from the navigation device.

If the navigation device supports API level 2, additional information can be received with navMessageReceiveX.

Note that the text message is only valid after a "message received" event is raised; the event will be removed when this function block is called.
A "message received" event is raised when the user writes a text message or sends a quick message.

Input:

None.

Output:

time : DINT
The timestamp when the text message was sent from the navigation device.

message : STRING
The received text message.

ready : BOOL
TRUE: If data is ready.
FALSE: If data is not ready.

Declaration:

```
FUNCTION_BLOCK navMessageReceive;
VAR_OUTPUT
   time    : DINT;
   message : STRING;
   ready   : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   msgRcv : navMessageReceive;
END_VAR;

FUNCTION ReadMessage;
   msgRcv();
   IF msgRcv.ready THEN
      DebugMsg(message := "*** message received ***");
      DebugFmt(message := "-time=\4", v4 := msgRcv.time);
      DebugMsg(message := "-text=" + msgRcv.message);
   END_IF;
END_FUNCTION;
```

```
THREAD_BLOCK navMonitor;
VAR
   event : INT := 0;
END_VAR;

WHILE event <> -1 DO
   event := navWaitEvent(timeout := -1);
   CASE event OF
      ...
      2: ReadMessage();
      ...
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;
```

4.2.31.4.6 navMessageReceiveX (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 2.20 / 1.00.00
Nav. API level: 2

---

navMessageReceiveX is an extended version of navMessageReceive and returns a text message received from the navigation device.

If the navigation device does not support API level 2, the function will work as navMessageReceive, and extended data will not be accessible.

Note that the text message is only valid after a "message received" event is raised; the event will be removed when this function block is called.
A "message received" event is raised when the user writes a text message or sends a quick message.

Input:

None.

Output:

time : DINT
The timestamp when the text message was sent from the navigation device.

id : INT
This message is a reply to the text message with this unique ID.
The ID will be zero on navigation devices that do not support message threading, or if it is not a reply.

message : STRING
The received text message.

ready : BOOL
TRUE: If data is ready.
FALSE: If data is not ready.

Declaration:

```
FUNCTION_BLOCK navMessageReceiveX;
VAR_OUTPUT
   id      : INT;
```

```
   time    : DINT;
   message : STRING;
   ready   : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   msgRcv : navMessageReceiveX;
END_VAR;

FUNCTION ReadMessage;
   msgRcv();
   IF msgRcv.ready THEN
      DebugMsg(message := "*** message received ***");
      DebugFmt(message := "-time=\4", v4 := msgRcv.time);
      DebugFmt(message := "-last=\1", v1 := msgRcv.id);
      DebugMsg(message := "-text=" + msgRcv.message);
   END_IF;
END_FUNCTION;

THREAD_BLOCK navMonitor;
VAR
   event : INT := 0;
END_VAR;

WHILE event <> -1 DO
   event := navWaitEvent(timeout := -1);
   CASE event OF
      ...
      2: ReadMessage();
      ...
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;
```

4.2.31.4.7 navMessageQuickDefine (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 1.40 / 1.00.00
Nav. API level: 1

This function will define a quick message.
If a quick message with the specified ID already exists, it will be replaced with the message text in this packet; otherwise the quick message will be added.

Quick messages are a list of predefined text messages that can be sent from the navigation device.
When sending a quick message, the user has the option to modify the text before sending it.
Up to 120 quick messages can be stored.

Input:

id : INT
Unique ID to identify the quick message.

text : STRING
The text that is sent to the RTCU device when the quick message is selected (max. 49 characters).

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-4	- The navigation device rejected the command.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navMessageQuickDefine : INT;
VAR_INPUT
   ID   : INT;
   text : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   navMessageQuickDefine(id := 10, text := "99 bottles of beer on the
wall...");
   navMessageQuickDefine(id := 11, text := "Do you want the usual pizza?");
   ...
END;
END_PROGRAM;
```

4.2.31.4.8 navMessageQuickDelete (Function)

Architecture:	**X32 / NX32 / NX32L**
Device support:	**MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5**
Firmware version:	**1.40 / 1.00.00**
Nav. API level:	**1**

This function will delete a quick message.

Input:

id : INT
The ID used to identify the quick message in navMessageQuickDefine.

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with Navigation device.
-4	- The navigation device rejected the command.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navMessageQuickDelete : INT;
VAR_INPUT
```

```
   ID : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Delete a quick message
   navMessageQuickDelete(id := 11);
   ...
END;
END_PROGRAM;
```

4.2.31.4.9 navMessageReplyDefine (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 1.40 / 1.00.00
Nav. API level: 1

This function will define a reply for a text message.
If a reply with the same ID already exists, the text of the reply is updated.
Up to 200 message replies can be defined.

Input:

id : INT
Unique ID to identify this reply.

text : STRING
The text that is displayed for this reply. (Max. 49 characters)

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-4	- The navigation device rejected the command.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navMessageReplyDefine : INT;
VAR_INPUT
   ID   : INT;
   text : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
```

```
   navMessageReplyDefine(id := 20, text := "On the way");
   navMessageReplyDefine(id := 21, text := "Are you kidding?");
   navMessageReplyDefine(id := 22, text := "I dont feel so good");
   ...
END;
END_PROGRAM;
```

4.2.31.4.10 navMessageReplyDelete (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 1.40 / 1.00.00
Nav. API level: 1

This function will remove a text message reply.
When a reply is removed, it is also removed from the reply list of any text messages that is not replied to.
If all allowed replies are removed from a message, the message is treated as a normal message.

Input:

id : INT
The ID used to identify the reply in navMessageReplyDefine.

Returns: INT

0	- Success.
-1	- Navigation interface is not opened.
-2	- Error communicating with navigation device.
-4	- The navigation device rejected the command.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navMessageReplyDelete : INT;
VAR_INPUT
   ID : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Delete a message reply
   navMessageReplyDelete(id := 23);
   ...
END;
END_PROGRAM;
```

4.2.31.4.11 navMessageReplyReceive (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 1.40 / 1.00.00
Nav. API level: 1

---

navMessageReplyReceive returns the reply and timestamp for a text message.
Note that the message reply is only valid after a "message reply" event is raised; the event will be removed when this function block is called.
A "message reply" event is raised when the user replies to a text message.

Input:

None.

Output:

id : INT
The ID used to identify the text message in navMessageSend.

time : DINT
The timestamp for the reply.

reply : INT
The ID of the selected reply.

ready : BOOL
TRUE: If data is ready.
FALSE: If data is not ready.

Declaration:

```
FUNCTION_BLOCK navMessageReplyReceive;
VAR_OUTPUT
   ID    : INT;
   time  : DINT;
   reply : INT;
   ready : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   msgAck : navMessageReplyReceive;
END_VAR;

FUNCTION ReadMessageReply;
   msgAck();
   IF msgAck.ready THEN
      DebugMsg(message := "*** message reply received ***");
      DebugFmt(message := "-msgid=\1", v1 := msgAck.ID);
      DebugFmt(message := "-time=\4", v4 := msgAck.time);
      CASE msgAck.reply OF
         1: DebugMsg(message := "-reply=[OK]");
         2: DebugMsg(message := "-reply=[YES]");
         3: DebugMsg(message := "-reply=[NO]");
```

```
      ELSE
         DebugFmt(message := "-reply=\1", v1 := msgAck.reply);
      END_CASE;
   END_IF;
END_FUNCTION;

THREAD_BLOCK navMonitor;
VAR
   event : INT := 0;
END_VAR;

WHILE event <> -1 DO
   event := navWaitEvent(timeout := -1);
   CASE event OF
      1: ReadMessageReply();
      ...
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;
```

4.2.31.4.12 navPopupAlert (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 17

This function will show an alert on the display of the navigation unit.
It will disappear after the configurable timeout or when the user presses anywhere on the display.

Input:

id : INT
Unique ID to identify this alert.

icon : INT (0..22, *default: 0*)
The icon to use for the alert:

- 0 - No icon needed
- 1 - Driver Behavior
- 2 - Tire Pressure
- 3 - Temperature
- 4 - Door Ajar
- 5 - Vehicle Maintenance
- 6 - OBD-II Generic Sensor
- 7 - Generic Sensor 1
- 8 - Generic Sensor 2
- 9 - Generic Sensor 3
- 10 - No signal
- 11 - Day Hours counter
- 12 - Weekly Hours counter
- 13 - Rest Hours counter
- 14 - Break Hours counter
- 15 - Dispatcher Tasks
- 16 - Weight
- 17 - Information
- 18 - Fuel
- 19 - Available (EU symbol)
- 20 - Driving (EU symbol)

21 - Rest (EU symbol)
22 - Working (EU symbol)

timeout : SINT (0..15, *default: 0*)
1-15 - The number of seconds to show the alert for.
0 - Show the alert for two hours

severity : SINT (0..2, *default: 0*)
The severity of the alert:
0 - Normal
1 - Medium
2 - High

play_sound : BOOL (*default FALSE*)
Set to true to play a sound when the alert is received.

text : STRING
The text for the alert. Max. 109 characters.

Returns: INT

0 - Success.
-1 - Navigation interface is not open.
-2 - Error communicating with navigation device.
-4 - The navigation device rejected the command.
-8 - Invalid parameter.
-11 - Popup alerts are not supported.
-12 - Navigation interface is busy.

Declaration:

```
FUNCTION navPopupAlert : INT;
VAR_INPUT
   ID         : DINT;
   icon       : INT := 0;
   timeout    : SINT := 0;
   severity   : SINT := 0;
   play_sound : BOOL := FALSE;
   text       : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Show alert
   navPopupAlert(ID := 5, text := "Overheating", severity := 2, icon := 3,
play_sound := TRUE);
   ...
END;
END_PROGRAM;
```

4.2.31.5 Route Management

4.2.31.5.1 navStopSet (Function)

```
Architecture:       X32 / NX32 / NX32L
Device support:     MX2, AX9, CX1 pro/flex/warp, SX1, MX2
                    turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-
                    900, LX2, LX4, LX5
Firmware version:   1.40 / 1.00.00
Nav. API level:     1
```

This function will set a destination in the navigation device.
When a destination is set, a small notification is presented.

The position of the destination must be given in the semicircle format.
Up to 20 destinations can be defined.

A semicircle is a unit of location-based measurement on an arc.
An arc of 180 degrees is made up of many semicircle units; 2^31 semicircles to be exact.
Semicircles that correspond to North latitudes and East longitudes are indicated with positive values;
semicircles that correspond to South latitudes and West longitudes are indicated with negative values.

The following formulas show how to convert between degrees and semicircles:
degrees = semicircles * (180 / 2^31)
semicircles = degrees * (2^31 / 180)

Note:
On some older models/firmware versions, and if a destination with the same ID already exists, the position and text will be updated. If the active destination is updated, the navigation device will recalculate the route to the new location.
Updating a destination does not change the status.

Input:

ID : INT
Unique ID to identify the destination when status is received.
Note: the ID of -1 is reserved and will result in an error.

latitude : DINT
The latitude of the position.

longitude : DINT
The longitude of the position.

text : STRING
A text comment to identify the destination in the route. Max. 199 characters.

Returns: **INT**

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-7	- The selected ID is illegal.
-9	- A stop with this ID is already set.
-10	- The stop list is full.

-12 - Navigation interface is busy.

Declaration:

```
FUNCTION navStopSet : INT;
VAR_INPUT
   ID        : INT;
   latitude  : DINT;
   longitude : DINT;
   text      : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Add a destination to the route
   navStopSet(id := 61, latitude := 666377598, longitude := 117525954, text :=
"Logic IO - main office");
   ...
END;
END_PROGRAM;
```

4.2.31.5.2 navStopReceive (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 1.40 / 1.00.00
Nav. API level: 1

navStopReceive returns the status of a destination.
Note that the destination status is only valid after a "destination status" event is raised; the event will be removed when this function block is called.
A "destination status" event is raised when the status of a destination changes or is requested with navStopRequest.

Input:

None.

Output:

ID : INT
The ID used to identify the destination in navStopSet.

status : INT
The status of the destination.

1 - Active. The user is enroute to the destination.
2 - Done. The user has arrived at the destination.
3 - Unread inactive. The destination is received but the user has not read the destination comment yet.
4 - Read inactive. The user has read the destination comment.
5 - Deleted/not found. The destination has been deleted or could not be found.

index : INT
The index of the destination in the route.

ready : BOOL
TRUE: If data is ready.
FALSE: If data is not ready.

Declaration:

```
FUNCTION_BLOCK navStopReceive;
VAR_OUTPUT
   ID     : INT;
   status : INT;
   index  : INT;
   ready  : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   stopRcv : navStopReceive;
END_VAR;

FUNCTION ReadStopStatus;
   stopRcv();
   IF stopRcv.ready THEN
      DebugMsg(message := "*** stop status received ***");
      DebugFmt(message := "-stop=\1", v1 := stopRcv.ID);
      DebugFmt(message := "-index=\1", v1 := stopRcv.index);
      CASE stopRcv.status OF
         1: DebugMsg(message := "-state=Active");
         2: DebugMsg(message := "-state=Done");
         3: DebugMsg(message := "-state=Unread - inactive");
         4: DebugMsg(message := "-state=Read - inactive");
         5: DebugMsg(message := "-state=Deleted/Not found");
      ELSE
         DebugFmt(message := "-state=\1", v1 := stopRcv.status);
      END_CASE;
   END_IF;
END_FUNCTION;

THREAD_BLOCK navMonitor;
VAR
   event : INT := 0;
END_VAR;

WHILE event <> -1 DO
   event := navWaitEvent(timeout := -1);
   CASE event OF
      ...
      5: ReadStopStatus();
      ...
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;
```

4.2.31.5.3 navStopRequest (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 1.40 / 1.00.00
Nav. API level: 1

This function will request the status of a destination.
A "destination status" event is raised when the destination status is received. See also navWaitEvent for event # 5.

Input:

ID : INT
The ID used to identify the destination in navStopSet.

Returns: INT

0 - Success.
-1 - Navigation interface is not open.
-2 - Error communicating with navigation device.
-12 - Navigation interface is busy.

Declaration:

```
FUNCTION navStopRequest : INT;
VAR_INPUT
   ID : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Request the status of a destination
   navStopRequest(id := 61);
   ...
END;
END_PROGRAM;
```

4.2.31.5.4 navStopSort (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 1.40 / 1.00.00
Nav. API level: 1

This function will sort all the destinations in the route in such a way that they can be visited in order in the shortest distance possible if starting from the user's current position.

Input:

None.

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navStopSort : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Sort destinations for shortest path
   navStopSort();
   ...
END;
END_PROGRAM;
```

4.2.31.5.5 navStopIndexSet (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 1.40 / 1.00.00
Nav. API level: 1

This function will move the destination to another position in the route.

Input:

ID : INT
The ID used to identify the destination in navStopSet.

index : INT
The index of the new position in the route.

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navStopIndexSet : INT;
VAR_INPUT
   ID    : INT;
   index : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Set the position of a destination in the route
   navStopIndexSet(id := 61, index := 0);
   ...
END;
END_PROGRAM;
```

4.2.31.5.6 navStopSetActive (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 1.40 / 1.00.00
Nav. API level: 1

This function will activate the destination. The currently active destination, if any, is set to inactive. Note that if the navigation device cannot find a street at the destination, the destination is not activated, although the former active destination is still set to inactive.

Input:

ID : INT
The ID used to identify the destination in navStopSet.

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navStopSetActive : INT;
VAR_INPUT
   ID : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Activate a destination
   navStopSetActive(id := 61);
   ...
END;
END_PROGRAM;
```

4.2.31.5.7 navStopSetDone (Function)

```
Architecture:       X32 / NX32 / NX32L
Device support:     MX2, AX9, CX1 pro/flex/warp, SX1, MX2
                    turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-
                    900, LX2, LX4, LX5
Firmware version:   1.40 / 1.00.00
Nav. API level:     1
```

This function will change the status of the destination to "Done".
The next destination in the route, if any, is not activated.

Input:

ID : INT
The ID used to identify the destination in navStopSet.

Returns: INT

```
   0    - Success.
  -1    - Navigation interface is not open.
  -2    - Error communicating with navigation device.
 -12    - Navigation interface is busy.
```

Declaration:

```
FUNCTION navStopSetDone : INT;
VAR_INPUT
   ID : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Mark a destination as completed
   navStopSetDone(id := 61);
   ...
END;
END_PROGRAM;
```

4.2.31.5.8 navStopDelete (Function)

```
Architecture:       X32 / NX32 / NX32L
Device support:     MX2, AX9, CX1 pro/flex/warp, SX1, MX2
                    turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-
                    900, LX2, LX4, LX5
Firmware version:   1.40 / 1.00.00
Nav. API level:     1
```

This function will delete a destination. After this function is called, the destination ID is invalid.

Input:

ID : INT
The ID used to identify the destination in navStopSet.

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navStopDelete : INT;
VAR_INPUT
   ID : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Delete a destination from the route
   navStopDelete(id := 61);
   ...
END;
END_PROGRAM;
```

4.2.31.5.9 navStopRouteCreate (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 14

This function will begin the construction of a route.
Routes are special stops that contain a number of individual stop points and optionally via points between the stop points.
When the route has been constructed, call navStopSetRoute to send it to the navigation device.
If the route is not to be used anyway, call navStopRouteAbort to close the route.

Input:

None.

Returns: INT

0	- Success.
-11	- Routes are not supported.
-12	- A route or a custom form is already being constructed. Please abort it and try again.

Declaration:

```
FUNCTION navStopRouteCreate : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;
VAR
```

```
   rc : INT;
END_VAR;
BEGIN
   ...
   // Begin the construction of a route
   rc := navStopRouteCreate();
   ...
END;
END_PROGRAM;
```

4.2.31.5.10 navStopRouteAbort (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 14

This function will abort the construction of a route.

Input:
None.

Returns: INT
0 - Success.
-1 - The route was not open.

Declaration:
```
FUNCTION navStopRouteAbort : INT;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Abort construction of a route.
   navStopRouteAbort();
   ...
END;
END_PROGRAM;
```

4.2.31.5.11 navStopRouteAddPoint (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 14

This function will add a stop or a via point to the route that is currently being constructed.
Via points are used for shaping the route while stop points are used to define intermediate destinations.

To avoid problems due to differences between the map on the navigation device and the map used for creating the route, the following guide lines should be used when adding points:

- Via points are recommended to be placed at least 50 meters apart when shaping a route travelling the same road segment.
- Avoid placing via points on or near intersections. Placement of the point at least 25 meters from any other road is advised.
- Stop points are recommend to be placed at least 160 meters away from the next immediate turn required.
- Use of multiple via points along the same guidance path may not be necessary. Via points can be used for purpose of identifying a new guidance maneuver (e.g., turning left) where a via point can be placed just before or after a turn point. Guidelines should continue to be followed where a 25 meter minimum distance is recommended between the placed via point and the intersection.
- Avoid placing via and stop points on or within 25 meters of an on-ramp or exit-ramp. For intentional exits, it should be sufficient to place the via point on the next road maneuver required.
- Avoid placing via and stop points on or within 25 meters of an interchange, or a bridge crossing one or more roads.
- Avoid placing via points or stop points in a tunnel.

Input:

stop : BOOL (*default : TRUE*)
Set to false to create a via point that has no text.

text : STRING
A text to describe the stop point. not used for via points. Max 39 characters.

latitude : DINT
The latitude of the position.

longitude : DINT
The longitude of the position.

Returns: INT

0	- Success.
-1	- The route is not being constructed. Call navStopRouteCreate first.
-8	- The text is invalid.
-10	- There is not space for this point. It may be necessary to split the route up into smaller routes.
-13	- The route must begin with a stop.

Declaration:

```
FUNCTION navStopRouteAddPoint : INT;
VAR_INPUT
   stop      : BOOL := TRUE;
   text      : STRING;
   latitude  : DINT;
   longitude : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Add a stop point
```

```
   navStopRouteAddPoint(stop := TRUE, latitude := 666367158, longitude :=
117518775, text := "Start");
   ...
END;
END_PROGRAM;
```

4.2.31.5.12 navStopSetRoute (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 14

This function will upload the constructed route to the navigation device.
The upload progress will be reported with GPI progress events that can be read with navGPIProgressReceive.
When the upload is completed, navGPIProgressReceive will report any errors with the form.
See navStopSetRouteFile for a list of the additional error codes.

Input:

ID : INT
Unique ID to identify the route when a stop status is received.
Note: the ID of -1 is reserved and will result in an error.

text : STRING
A text comment to identify the route. Max. 199 characters.

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-7	- Invalid stop id.
-8	- The text is invalid.
-10	- The route is too large to upload. Call navStopRouteAbort and try again. It may be necessary to split the route into smaller routes.
-11	- Routes are not supported.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navStopSetRoute : INT;
VAR_INPUT
   id       : INT;
   text     : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Add a route
   navStopSetRoute(id := 1, text := "Route 1");
   ...
```

```
END;
END_PROGRAM;
```

4.2.31.5.13 navStopSetRouteFile (Function)

```
Architecture:       NX32 / NX32L
Device support:     MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400,
                    NX-900, LX2, LX4, LX5
Firmware version:   4.70 / 1.30.00
Nav. API level:     14
```

This function will start to transfer a route file to the navigation device.
A "GPI progress" event is raised when there are changes to the progress and/or status of the file transfer. See also navWaitEvent for event # 9.
The route file is a binary file that follows the format specified in "5.1.6.2.1 Path Specific Stop (PSS) file data format" in the "Garmin Fleet Management Interface Control Specification", which is part of the Garmin Fleet Management SDK, available at http://developer.garmin.com/fleet-management/overview/.
The file can optionally be compressed with the GZIP format.

When the upload is completed, navGPIProgressReceive will report any errors with the route.
In addition to the common errors, the following errors can happen:

257	- Unexpected signature
258	- Unexpected version.
259	- Zero length Stop text name.
261	- Exceeded text maximum length.
262	- Invalid Unique ID.
263	- Invalid Latitude value.
264	- Invalid Longitude value.
265	- Invalid point type found.
266	- First point type is not a Destination.
267	- Last point type is not a Destination.
268	- Exceeded maximum number of shaping points.
269	- Exceeded maximum number of Destination points.
270	- Exceeded maximum total number of points.
271	- Less than 2 destination points found.

Note: To create a route on the RTCU device, use the navStopRouteCreate and navStopSetRoute functions instead.

Input:

filename : STRING
The name and path of the file to transfer.

ID : STRING
A small text message to identify the file.
Must be between 1 and 16 characters.

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-3	- File not found.
-4	- The navigation device rejected the command.
-8	- Invalid ID.
-11	- Routes are not supported.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navStopSetRouteFile : INT;
VAR_INPUT
   filename : STRING;
   id       : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Start transfer of a route
   navStopSetRouteFile(filename := "route01.pss", id := "Route 1");
   ...
END;
END_PROGRAM;
```

4.2.31.6 Estimated Time of Arrival (ETA)

4.2.31.6.1 navAutoArrival (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 1.40 / 1.00.00
Nav. API level: 1

This function will change the auto-arrival criteria.
The auto-arrival feature is used to automatically detect that the user has arrived at a destination and then to prompt the user if they would like to mark the destination as done and start navigating to the next destination on the route.
Note that if the navigation device cannot find a street at the destination, it is not activated and the navigation device activates the next destination on the route. (Until a destination is OK or there are no more destinations on the route).

Input:

time : DINT (*default: 30*)
The number of seconds the navigation device should be stopped close to the destination before the auto-arrival feature is activated.

distance : DINT (*default: 100*)
How close to the destination, in meters, the navigation device has to be before the auto-arrival feature is activated.

Returns: INT

0	- Success.
-1	- Navigation interface is not opened.
-2	- Error communicating with navigation device.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navAutoArrival : INT;
VAR_INPUT
   time     : DINT := 30;
   distance : DINT := 100;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Set the criteria for the auto-arrival detection
   navAutoArrival(time := 20, distance := 75);
   ...
END;
END_PROGRAM;
```

4.2.31.6.2 navETAReceive (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 1.40 / 1.00.00
Nav. API level: 1

This function block returns information about the Estimated Time of Arrival.
Note that the ETA is only valid after an "ETA" event is raised; the event will be removed when this function block is called.
An "ETA" event is raised: when the navigation device calculates a route to the active destination, at regular intervals (when enabled with navETAAutoSet), or when requested with navETARequest.

Input:

None.

Output:

time : DINT
The estimated time of arrival for destination.

distance : DINT
The distance in meters from the current position to the active destination.
-1 if no destination is active.

latitude : DINT
The latitude of the active destination.

longitude : DINT
The longitude of the active destination.

ID : INT
The unique ID of the active destination.
-1 if stop is not set with navStopSet.

ready : BOOL
TRUE: If data is ready.
FALSE: If data is not ready.

Declaration:

```
FUNCTION_BLOCK navETAReceive;
VAR_OUTPUT
   time      : DINT;
   distance  : DINT;
   latitude  : DINT;
   longitude : DINT;
   ID        : INT;
   ready     : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   etaRcv : navETAReceive;
END_VAR;

FUNCTION ReadETA;
   etaRcv();
   IF etaRcv.ready THEN
      DebugMsg(message := "*** ETA received ***");
      DebugFmt(message := "-time=\4", v4 := etaRcv.time);
      DebugFmt(message := "-distance=\4", v4 := etaRcv.distance);
      DebugFmt(message := "-stop=\1", v1 := etaRcv.id);
      DebugFmt(message := "-latitude=\4", v4 := etaRcv.latitude);
      DebugFmt(message := "-longitude=\4", v4 := etaRcv.longitude);
   END_IF;
END_FUNCTION;

THREAD_BLOCK navMonitor;
VAR
   event : INT := 0;
END_VAR;

WHILE event <> -1 DO
   event := navWaitEvent(timeout := -1);
   CASE event OF
      ...
      4: ReadETA();
      ...
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;
```

4.2.31.6.3 navETARequest (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 1.40 / 1.00.00
Nav. API level: 1

This function will request an ETA from the navigation device.
An "ETA" event is raised when the ETA is received. See also navWaitEvent for event # 4.

Input:
None.

Returns: INT
0 - Success.
-1 - Navigation interface is not opened.
-2 - Error communicating with navigation device.
-12 - Navigation interface is busy.

Declaration:
```
FUNCTION navETARequest : INT;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Request an ETA update
   navETARequest();
   ...
END;
END_PROGRAM;
```

4.2.31.6.4 navETAAutoSet (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 1.40 / 1.00.00
Nav. API level: 1

This function will enable or disable raising "ETA" events at regular intervals.

Input:
freq : INT (0,10..)
The frequency of ETA reports in seconds.
0 - Disables automatic ETA reports.

Returns: INT
0 - Success.
-1 - Navigation interface is not open.
-8 - Illegal frequency.
-12 - Navigation interface is busy.

Declaration:
```
FUNCTION navETAAutoSet : INT;
VAR_INPUT
   freq : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Enable ETA events at regular intervals
   navETAAutoSet(freq := 20);
   ...
END;
END_PROGRAM;
```

4.2.31.6.5 navETASetMode (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 23

This function is used to configure what ETA events should be reported while driving.
By default only ETA events for stops created with the API are reported.
This setting is stored across reboots of the navigation device.

Input:

mode : SINT (0..3)
The wanted reporting mode:

0	- No ETA events.
1	- Only ETA events for stops created with this API.
2	- Only ETA events for user created stops.
3	- All ETA events.

Returns: INT

0	- Success.
-1	- Navigation interface is not opened.
-2	- Error communicating with navigation device.
-4	- The navigation device rejected the command.
-8	- Invalid mode.
-11	- Changing the ETA mode is not supported.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navETASetMode : INT;
VAR_INPUT
   mode     : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Set mode to 3, all ETA events
   navETASetMode(mode := 3);
   ...
```

```
END;
END_PROGRAM;
```

4.2.31.7 File transfers

4.2.31.7.1 navGPITransfer (Function)

```
Architecture:       X32 / NX32 / NX32L
Device support:     MX2, AX9, CX1 pro/flex/warp, SX1, MX2
                    turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-
                    900, LX2, LX4, LX5
Firmware version:   2.20 / 1.00.00
Nav. API level:     1
```

This function will start to transfer a GPI file to the navigation device.
A "GPI progress" event is raised when there are changes to the progress and/or status of the GPI transfer. See also navWaitEvent for event # 9.
Note: this function is not supported on the NMP.

Input:

filename : STRING
The name and path of the file to transfer.

ID : STRING
A small text message to identify the GPI file.
Must be between 1 and 16 characters.

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-3	- File not found.
-4	- The navigation device rejected the command.
-8	- Invalid ID.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navGPITransfer : INT;
VAR_INPUT
   filename : STRING;
   id       : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Start transfer of GPI
   navGPITransfer(filename := "depart01.gpi", id := "Stores - 2010");
   ...
END;
END_PROGRAM;
```

4.2.31.7.2 navGPIProgressReceive (Functionblock)

```
Architecture:       X32 / NX32 / NX32L
Device support:     MX2, AX9, CX1 pro/flex/warp, SX1, MX2
                    turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-
                    900, LX2, LX4, LX5
Firmware version:   2.20 / 1.00.00
Nav. API level:     1
```

This function block returns information about the status and progress of a file transfer.
Note that the progress is only valid after a "File progress" event is raised; the event will be removed when this function block is called.
Note: this function block is not supported on the NMP.

Input:

None.

Output:

status : INT
The status of the transfer.

1	- Transfer in progress.
0	- Transfer completed.
-2	- Error communicating with navigation device.
-3	- Error reading file.
-4	- The navigation device rejected the transfer.

progress : SINT
The progress of the transfer in percent.

type : SINT
The type of file being transferred. Available starting with Firmware v4.70 / 1.30.00.

0	- GPI file.
1	- Custom form.
2	- Routes.

error : INT
The detailed error for the file transfer. Available starting with Firmware v4.70 / 1.30.00.
The error code depends on the file type.
Common error codes:

0	- Success.
1-4	- Transfer error.
5	- Invalid file.
6	- The file is not of the expected type.
7-15	- Transfer error.

In addition to the common error codes, most file types have detailed error codes that can be found in the documentation of the function that started the transfer:

Custom form	- navFormUploadFile.
Routes	- navStopSetRouteFile

ready : BOOL

TRUE:	If data is ready.
FALSE:	If data is not ready.

Declaration:

```
FUNCTION_BLOCK navGPIProgressReceive;
VAR_OUTPUT
   status    : INT;
   progress  : SINT;
   ready     : BOOL;
   error     : INT;
   type      : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   gpiRcv : navGPIProgressReceive;
END_VAR;

FUNCTION ReadGPIProgress;
   gpiRcv();
   IF gpiRcv.ready THEN
      DebugMsg(message := "*** progress received ***");
      DebugFmt(message := "-status=\1", v1 := gpiRcv.status);
      DebugFmt(message := "-progress=\1", v1 := gpiRcv.progress);
   END_IF;
END_FUNCTION;

THREAD_BLOCK navMonitor;
VAR
   event : INT := 0;
END_VAR;

WHILE event <> -1 DO
   event := navWaitEvent(timeout := -1);
   CASE event OF
      ...
      9: ReadGPIProgress();
      ...
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;
```

4.2.31.7.3 navGPIRequestID (Function)

Architecture:	**X32 / NX32 / NX32L**
Device support:	**MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5**
Firmware version:	**2.20 / 1.00.00**
Nav. API level:	**1**

This function will request the ID of the GPI from the navigation device.
A "GPI ID" event is raised when the ID is received. See also navWaitEvent for event # 8.
Note: this function is not supported on the NMP.

Input:

None.

Returns: **INT**

0 - Success.
-1 - Navigation interface is not open.
-2 - Error communicating with navigation device.
-12 - Navigation interface is busy.

Declaration:

```
FUNCTION navGPIRequestID : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Request the ID of the GPI
   navGPIRequestID();
   ...
END;
END_PROGRAM;
```

4.2.31.7.4 navGPIReceiveID (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 2.20 / 1.00.00
Nav. API level: 1

This function returns the received ID of the GPI.
Note that the ID is only valid after a "GPI ID" event is raised; the event will be removed when this function is called.
A "GPI ID" event is raised when requested with navGPIRequestID.
Note: this function is not supported on the NMP.

Input:

None.

Returns: STRING

The text message that identifies the GPI. It is empty if an event is not raised.

Declaration:

```
FUNCTION navGPIReceiveID : STRING;
```

Example:

```
INCLUDE rtcu.inc

THREAD_BLOCK navMonitor;
VAR
   event : INT := 0;
END_VAR;

WHILE event <> -1 DO
   event := navWaitEvent(timeout := -1);
```

```
   CASE event OF
      ...
      8: DebugMsg(message := "GPI ID: " + navGPIReceiveID());
      ...
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;
```

4.2.31.8 Waypoint management

4.2.31.8.1 navWaypointSet (Function)

```
Architecture:       X32 / NX32 / NX32L
Device support:     MX2, AX9, CX1 pro/flex/warp, SX1, MX2
                    turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-
                    900, LX2, LX4, LX5
Firmware version:   2.20 / 1.00.00
Nav. API level:     2
```

This function will set a waypoint in the navigation device.

The position of the waypoint must be given in the semicircle format (see navStopSet for more info).

Input:

ID : INT
Unique ID to identify the waypoint.

symbol : INT
The symbol that is displayed on the map for the waypoint..

Airport	16384	Forest	8245
Ball Park	169	Gas Station	8
Bank	6	Ghost Town	8239
Bar	13	Glider Area	16393
Beach	8244	Golf Course	8197
Boat Ramp	150	Heliport	16388
Bridge	8233	Hotel	173
Building	8234	Hunting Area	171
Campground	151	Information	157
Car	170	Levee	8240
Car Repair	8207	Live Theater	8221
Cemetery	8235	Man Overboard	21
Church	8236	Marina	0
City (Large)	8200	Medical Facility	156
City (Medium)	8199	Military	8241
City (Small)	8198	Mine	174
Civil	8237	Movie Theater	8210
Convenience Store	8220	Oil Field	8242
Crossing	8238	Parachute Area	16395
Dam	164	Park	159
Danger Area	14	Parking Area	158
Drinking Water	154	Picnic Area	160
Fast Food	8208	Post Office	8214
Fishing Area	7	Private Field	16389
Fitness Center	8209	Residence	10
Restaurant	11		
Restroom	152		
RV Park	8215		
Scales	8226		
Scenic Area	161		
School	1		
Seaplane Base	16402		
Shipwreck	19		
Shopping Center	172		
Short Tower	16392		
Shower	153		
Skiing Area	162		
Soft Field	16390		
Summit	8246		
Swimming Area	163		
Tall Tower	16391		
Telephone	155		
Toll Booth	8227		
TracBack Point	8196		
Trail Head	175		
Truck Stop	176		
Tunnel	8243		
Ultralight Area	16394		
Waypoint	18		

latitude : DINT
The latitude of the position.

longitude : DINT
The longitude of the position.

categories : INT
A bit field that indicates which categories the waypoint belongs to.
For example, if the waypoint should be in categories with IDs 0 and 5, the categories should have the lowest bit and the 6th lowest bit set to 1 for a value of 33 decimal (00000000 00100001 in binary).

name : STRING
The name of the waypoint. Max. 30 characters.

comment : STRING
Any other notes for the waypoint. Max. 50 characters.

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-4	- The navigation device rejected the command.
-8	- The name or comment is illegal.
-11	- This is not supported by navigation device.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navWaypointSet : INT;
VAR_INPUT
   ID         : INT;
   symbol     : INT;
   latitude   : DINT;
   longitude  : DINT;
   categories : INT;
   name       : STRING;
   comment    : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Insert Logic IO waypoint
   navWaypointSet(ID := 5, symbol := 18, latitude := 666377598, longitude :=
117525954, categories := 16#1000, name := "Logic IO", comment := "Main
office");
   ...
   // Delete waypoint
   navWaypointDelete(id := 5);
   ...
END;
END_PROGRAM;
```

4.2.31.8.2 navWaypointDelete (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 2.20 / 1.00.00
Nav. API level: 2

This function will delete a waypoint.

Input:

ID : INT
The ID of the waypoint.

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-4	- The navigation device rejected the command.
-11	- This is not supported by navigation device.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navWaypointDelete : INT;
VAR_INPUT
   ID          : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Delete waypoint
   navWaypointDelete(id := 15);
   ...
END;
END_PROGRAM;
```

4.2.31.8.3 navWaypointDeleteByCat (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 2.20 / 1.00.00
Nav. API level: 2

This function will delete all waypoints in the categories selected.

Input:

categories : INT
A bit field that indicates which categories to delete waypoints from.
For example, if the waypoint should be in categories with IDs 0 and 5, the categories should have the lowest bit and the 6th lowest bit set to 1 for a value of 33 decimals (00000000 00100001 in binary).

Returns: INT

0	- Success.
-1	- Navigation interface is not opened.
-2	- Error communicating with navigation device.
-4	- The navigation device rejected the command.
-11	- This is not supported by navigation device.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navWaypointDeleteByCat : INT;
VAR_INPUT
   categories : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Delete all waypoints belonging to category 0 and 5
   navWaypointDeleteByCat(categories := 16#0041);
   ...
END;
END_PROGRAM;
```

4.2.31.9 Sensor display

4.2.31.9.1 navSensorConfig (Function)

Architecture:	**NX32 / NX32L**
Device support:	**MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5**
Firmware version:	**4.70 / 1.30.00**
Nav. API level:	**17**

This function will create or modify a sensor on the display of the navigation unit.
Sensors are used to display information to the driver and has a log that can contain the last 24 updates. The sensor value and the log are available for the driver to access from the user interface.
The sensor data are text strings that can come from any source and does not necessarily have to be from an actual sensor.

Input:

id : INT
Unique ID to identify this sensor.

icon : INT (0..22, *default: 0*)
The icon of the sensor:

0	- No icon needed

1 - Driver Behavior
2 - Tire Pressure
3 - Temperature
4 - Door Ajar
5 - Vehicle Maintenance
6 - OBD-II Generic Sensor
7 - Generic Sensor 1
8 - Generic Sensor 2
9 - Generic Sensor 3
10 - No signal
11 - Day Hours counter
12 - Weekly Hours counter
13 - Rest Hours counter
14 - Break Hours counter
15 - Dispatcher Tasks
16 - Weight
17 - Information
18 - Fuel
19 - Available (EU symbol)
20 - Driving (EU symbol)
21 - Rest (EU symbol)
22 - Working (EU symbol)

index : INT (1..32767, *default 1*)
Used for sorting the sensors.

name : STRING
The name of the sensor. Max. 39 characters.

Returns: INT

0 - Success.
-1 - Navigation interface is not open.
-2 - Error communicating with navigation device.
-4 - The navigation device rejected the command.
-8 - Invalid parameter.
-10 - There is not space for any more sensors.
-11 - Sensor display is not supported.
-12 - Navigation interface is busy.

Declaration:

```
FUNCTION navSensorConfig : INT;
VAR_INPUT
   ID       : DINT;
   icon     : INT := 0;
   index    : INT := 0;
   name     : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
```

```
   // Create sensor 55
   navSensorConfig(ID := 55, name := "Temperature", icon := 3);
   ...
   // Set the value of the sensor
   navSensorUpdate(ID := 55, status := "40$B0C", log_status := TRUE, desc :=
"Hot");
   ...
END;
END_PROGRAM;
```

4.2.31.9.2 navSensorUpdate (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 17

This function will update the status of a sensor on the display of the navigation unit.

Input:

id : INT
Unique ID to identify the sensor to update.

severity : SINT (0..2, *default: 0*)
The severity of the sensor status:
- 0 - Normal
- 1 - Medium
- 2 - High

play_sound : BOOL (*default FALSE*)
Set to true to play a sound when the update is received.

log_status : BOOL (*default FALSE*)
Set to true to store this update in the history of the sensor.

status : STRING
The new status of the sensor. Max. 79 characters.

desc : STRING
The description for the status. Max. 109 characters.

Returns: INT

- 0 - Success.
- -1 - Navigation interface is not open.
- -2 - Error communicating with navigation device.
- -3 - The sensor could not be found.
- -4 - The navigation device rejected the command.
- -8 - Invalid parameter.
- -11 - Sensor display is not supported.
- -12 - Navigation interface is busy.

Declaration:

```
FUNCTION navSensorUpdate : INT;
VAR_INPUT
   ID         : DINT;
```

```
   severity   : SINT := 0;
   play_sound : BOOL := FALSE;
   log_status : BOOL := FALSE;
   status     : STRING;
   desc       : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Create sensor 55
   navSensorConfig(ID := 55, name := "Temperature", icon := 3);
   ...
   // Set the value of the sensor
   navSensorUpdate(ID := 55, status := "40$B0C", log_status := TRUE, desc :=
"Hot");
   ...
END;
END_PROGRAM;
```

4.2.31.9.3 navSensorDelete (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 17

This function will delete a sensor from the display of the navigation unit.

Input:

id : INT
Unique ID to identify the sensor to delete.

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-3	- The sensor could not be found.
-4	- The navigation device rejected the command.
-11	- Sensor display is not supported.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navSensorDelete : INT;
VAR_INPUT
   ID         : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
```

```
PROGRAM NavigationExample;

BEGIN
   ...
   // Delete sensor 55
   navSensorDelete(ID := 55);
   ...
END;
END_PROGRAM;
```

4.2.31.10 Avoidance areas

4.2.31.10.1 navAvoidEnable (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 13

This function will enable or disable the support for avoidance areas on the navigation device. Note that disabling the support may delete all the areas on the navigation device, including areas created by the user.

Input:

enable : BOOL (*default TRUE*)
Determines if avoidance areas should be enabled.

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-4	- The navigation device rejected the command.
-11	- Avoidance areas are not supported.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navAvoidEnable : INT;
VAR_INPUT
   enable  : BOOL := TRUE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Enable support for avoidance areas
   navAvoidEnable();
   ...
END;
END_PROGRAM;
```

4.2.31.10.2 navAvoidAreaCreate (Function)

```
Architecture:       NX32 / NX32L
Device support:     MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400,
                    NX-900, LX2, LX4, LX5
Firmware version:   4.70 / 1.30.00
Nav. API level:     13
```

This function will create or modify an avoidance area on the navigation device.
Avoidance areas are areas that the routing will avoid and can e.g. be used to avoid road construction that is not reported by the traffic service.
The area is a rectangle defined by two diagonal corners.

Input:

id : INT
Unique ID to identify this area.

enable : BOOL (*default TRUE*)
Determines if the area is enabled by default.

name : STRING
The name of the area. Must be unique. Max. 48 characters.

lat_min : DINT
The latitude of the first corner.

lon_min : DINT
The longitude of the first corner.

lat_max : DINT
The latitude of the second corner.

lon_max : DINT
The longitude of the second corner.

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-4	- The navigation device rejected the command.
-7	- The name is already used by another area.
-8	- Invalid name.
-10	- There is not space for any more avoidance areas.
-11	- Avoidance areas are not supported.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navAvoidAreaCreate : INT;
VAR_INPUT
   ID      : DINT;
   enable  : BOOL := TRUE;
   name    : STRING;
   lat_min : DINT;
   lat_max : DINT;
   lon_min : DINT;
```

```
   lon_max : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Create an avoidance area
   navAvoidAreaCreate(id := 1, enable := TRUE, name := "Road construction",
         lat_max := navPositionToSemicircles(pos := 55514600),
         lon_min := navPositionToSemicircles(pos :=  9507940),
         lat_min := navPositionToSemicircles(pos := 55515120),
         lon_max := navPositionToSemicircles(pos :=  9509200)
      );
   ...
END;
END_PROGRAM;
```

4.2.31.10.3 navAvoidAreaDelete (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 13

This function will delete an avoidance area from the navigation device.

Input:

id : INT
Unique ID to identify the area.

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-4	- The navigation device rejected the command.
-11	- Avoidance areas are not supported.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navAvoidAreaEnable : INT;
VAR_INPUT
   ID      : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Delete an avoidance area
   navAvoidAreaDelete(id := 1);
```

```
   ...
END;
END_PROGRAM;
```

4.2.31.10.4 navAvoidAreaEnable (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 13

This function will enable or disable an avoidance area on the navigation device.

Input:

id : INT
Unique ID to identify the area.

enable : BOOL (*default TRUE*)
Determines if the area should be enabled.

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-4	- The navigation device rejected the command.
-11	- Avoidance areas are not supported.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navAvoidAreaEnable : INT;
VAR_INPUT
   ID       : DINT;
   enable   : BOOL := TRUE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Disable an avoidance area
   navAvoidAreaEnable(id := 1, enable := FALSE);
   ...
END;
END_PROGRAM;
```

4.2.31.11 Custom forms

4.2.31.11.1 nav: Custom Forms

The custom forms are stored on the navigation device and can be used by the user to fill out forms to report relevant information.

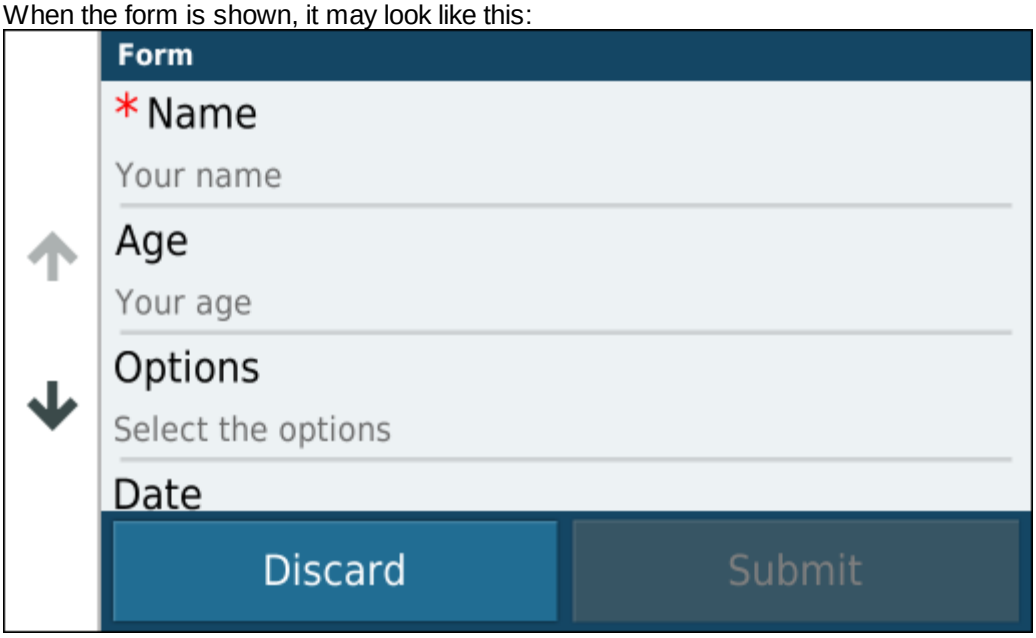
This can e.g. be used for tracking trips or for reporting errors.

The forms can be filled out and submitted also when the navigation device is not connected to the RTCU device. In this case the forms are queued and will be transmitted when connected again. The navigation device is limited to a maximum of 30 forms at the same time.

There are two ways to create custom forms:
1. Uploading a file using navFormUploadFile. This requires that the forms have been created externally and have then been placed in the file system. It makes it possible to support many different forms by e.g. placing them on a SD Card and then just uploading the needed forms.
2. Creating a form using navFormCreate and uploading it with navFormUpload. This allows for creating forms dynamically from the VPL application using the form creation functions.

When the form has been uploaded, it can be shown to the user, either by using navFormShow if it is supported, or by the user accessing the forms menu on the navigation device and selecting the form.

When the form is shown, it may look like this:



To fill out the form, the user must click on each field and enter the data. The fields marked with * are required and must be filled out before the submit button is enabled.

When the user has filled out a form and submitted it on the navigation device, the form is sent to the RTCU, where it will trigger the "Form received" event on navWaitEvent.
The form can then be read using the receive functions.
Note: While the RTCU is receiving the submitted form, no other communication can take place and the other navigation functions will report that the Navigation interface is busy.
Note: Empty fields are not included in the received form.

To read the data, there are multiple possibilities:
- Request the data for each known field. If the field is not included in the received form, the read functions will return an error that must be handled. See navFormCreate for an example of this.
- Use navFormReceiveField to iterate through the form and then handle each known ID as needed. See navFormReceiveField for an example of how this can be done.

Once all the data has been read, the navFormReceiveDone function must be called to release the "Form received" event.

Form management functions

- navFormDelete Deletes a form from the navigation device.

- navFormMove Moves a form on the navigation device.
- navFormGetPos Retrieves the position of a form on a navigation device
- navFormShow Shows a form to the user of the navigation device.

Form creation functions

The following functions are used for creating and managing forms on the navigation device.

- navFormCreate Begins the construction of a new form.
- navFormAbort Cancels the construction of a new form.
- navFormAddText Adds a text field to the form.
- navFormAddNumber Adds a number field to the form.
- navformAddSelect Adds a selection field to the form.
- navFormAddOption Adds an option to a selection field.
- navFormAddDate Adds a date field to the form.
- navFormAddTime Adds a time field to the form.
- navFormAddStop Adds a stop selection field to the form.
- navFormUpload Uploads the constructed form to the navigation device.
- navFormUploadFile Uploads a form from a file to the navigation device.

Receive functions

The following functions are used for reading the contents of a form that has been received from the navigation device.

- navFormReceive Reads information about a form that has been received from the navigation device.
- navFormReceiveDone Releases the received form once it is no longer needed.
- navFormReceiveField Reads information about a field on the received form.
- navFormReceiveReadText Reads a text field from the received form.
- navFormReceiveReadNumber Reads a number field from the received form.
- navFormReceiveReadSelect Reads a selection field from the received form.
- navFormReceiveReadDateTime Reads a date or a time field from the received form.
- navFormReceiveReadStop Reads a stop selection field from the received form.

4.2.31.11.2 navFormCreate (Function)

```
Architecture:       NX32 / NX32L
Device support:     MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400,
                    NX-900, LX2, LX4, LX5
Firmware version:   4.70 / 1.30.00
Nav. API level:     12
```

This function will begin the construction of a custom form.
Each form can contain a maximum of 30 fields, added using the navFormAdd* functions, e.g. navFormAddText to add a text field.
When the form has been constructed, call navFormUpload to send it to the navigation device.
If the form is not to be used anyway, call navFormAbort to close the form.

The custom forms are stored on the navigation device and can be used by the user to fill out forms to report relevant information.
This can e.g. be used for tracking trips or for reporting errors.
The forms can be filled out and submitted also when the navigation device is not connected to the RTCU device, in which case the forms are queued and will be transmitted when connected again.

When the RTCU device receives the submitted forms, the "Form received" event is raised and the contents of the form can be read by using navFormReceive and the functions for reading the individual fields, e.g. navFormReceiveReadText.

Input:

id : DINT
Unique ID to identify the form.
Note: the ID of -1 is reserved and will result in an error.

title : STRING
The title of the form. Max. 40 characters.

position : INT (1..32767, *default 1*)
Used for sorting the forms.

version : DINT (1..2147483647, *default 1*)
The version number of the form. When the contents of the form are changed, this should be incremented.

Returns: INT

0	- Success.
-7	- Invalid ID
-8	- Invalid parameter.
-11	- Custom forms are not supported.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navFormCreate : INT;
VAR_INPUT
   id       : DINT;
   title    : STRING;
   position : INT := 1;
   version  : DINT := 1;
END_VAR;
```

Example:

```
//------------------------------------------------------------------------------
-
// form.vpl, created 2018-11-28 14:45
//
//------------------------------------------------------------------------------
-
INCLUDE rtcu.inc

//  Input variables that can be configured via the configuration dialog (These
are global)
VAR_INPUT

END_VAR;

//  Output variables that can be configured via the configuration dialog
(These are global)
```

```
VAR_OUTPUT

END_VAR;

//  These are the global variables of the program
VAR
   msgSts         : navMessageStatusReceive;
   msgAck         : navMessageReplyReceive;
   msgRec         : navMessageReceiveX;
   etaRec         : navETAReceive;
   userStatus     : navUserStatusReceive;
   stopRcv        : navStopReceive;
   userID         : navUserIDReceiveX;
   gpiRcv         : navGPIProgressReceive;
   formReceive    : navFormReceive;
   speedAlert     : navSpeedLimitAlert;
   clock          : clockLinsecToTime;
END_VAR;

// Definition of the field IDs
#DEFINE ID_NAME      1
#DEFINE ID_AGE       2
#DEFINE ID_OPTIONS   3
#DEFINE ID_DATE      4
#DEFINE ID_TIME      5
#DEFINE ID_STOP      6

// Option IDs
#DEFINE ID_OPT_1     1
#DEFINE ID_OPT_2     2

FUNCTION TimeString : STRING;
VAR_INPUT
   linsec : DINT;
END_VAR;
   clock(linsec := linsec);
   TimeString := strFormat(format := "\1.\2.\3, ", v1 := clock.day, v2 :=
clock.month, v3 := clock.year);
   TimeString := TimeString + strFormat(format := "\1:\2:\3", v1 :=
clock.hour, v2 := clock.minute, v3 := clock.second);
END_FUNCTION;


FUNCTION ReadGPIProgress;
   gpiRcv();
   IF gpiRcv.ready THEN
      DebugMsg(message := "*** progress received ***");
      DebugFmt(message := "-status=  \1", v1 := gpiRcv.status);
      DebugFmt(message := "-progress=\1", v1 := gpiRcv.progress);
      DebugFmt(message := "-type=    \1", v1 := gpiRcv.type);
      DebugFmt(message := "-error=   \1", v1 := gpiRcv.error);
   END_IF;
END_FUNCTION;

FUNCTION createForm:INT;
VAR
   rc: INT;
END_VAR;
   // Begin constructing the form.
   rc := navFormCreate(id := 7, title:="Form", version := 1);
   DebugFmt(message := "Form create=\1", v1 := rc);

   // Add fields to the form:
   rc := navFormAddText(id := ID_NAME, title:="Name", desc:="Your name",
```

```
                        placeholder := "<Name>", required := TRUE);
   DebugFmt(message := "Add text=\1", v1 := rc);

   rc := navFormAddNumber(id:=ID_AGE, title:="Age", desc:="Your age",
placeholder:="<years>",
                        min_val:=0, max_val := 120, max_length := 3);
   DebugFmt(message := "Add number=\1", v1 := rc);

   rc := navFormAddSelect(id := ID_OPTIONS, title:="Options", desc:="Select
the options", multi:=TRUE);
   DebugFmt(message := "Add select=\1", v1 := rc);

   rc := navFormAddOption(id:=ID_OPT_1, select_id:=ID_OPTIONS, title:="Option
1");
   DebugFmt(message := "Add option=\1", v1 := rc);

   rc := navFormAddOption(id:=ID_OPT_2, select_id:=ID_OPTIONS, title:="Option
2");
   DebugFmt(message := "Add option=\1", v1 := rc);

   rc := navFormAddDate(id:=ID_DATE, title:="Date");
   DebugFmt(message := "Add date=\1", v1 := rc);

   rc := navFormAddTime(id:=ID_TIME, title:="Time");
   DebugFmt(message := "Add time=\1", v1 := rc);

   rc := navFormAddStop(id:=ID_STOP, title:="Stop", desc:="Select the stop");
   DebugFmt(message := "Add stop=\1", v1 := rc);


   // Upload the form to the navitagion device
   rc := navFormupload();
   DebugFmt(message := "Upload=\1", v1 := rc);

END_FUNCTION;


FUNCTION FormSubmit;
VAR
   rc    : INT;
   i     : INT;
   str   : STRING;
   n     : DINT;
   linsec: DINT := 0;
   stop  : INT;
END_VAR;
   formReceive();
   IF(formReceive.ready) THEN
       DebugMsg(message := "*** form received ***");
       DebugFmt(message := "-ID=      \4", v4 := formReceive.id);
       DebugFmt(message := "-sub id=  \4", v4 := formReceive.submit_id);
       DebugFmt(message := "-version= \4", v4 := formReceive.version);
       DebugFmt(message := "-time=    "+TimeString( linsec :=
formReceive.time));

      IF(formReceive.id <> 7) THEN
            DebugFmt(message:="Unknown form ID \4", v4:=formReceive.id);
            rc := navFormReceiveDone();
            DebugFmt(message:="navFormReceiveDone: \1", v1:=rc);
            RETURN;
      END_IF;

      IF(formReceive.version <> 1) THEN
            DebugFmt(message:="Unknown form version \4",
```

```
v4:=formReceive.version);
            rc := navFormReceiveDone();
            DebugFmt(message:="navFormReceiveDone: \1", v1:=rc);
            RETURN;
      END_IF;

      rc := navFormReceiveReadText(id := ID_NAME, text := str);
      DebugFmt(message:="Read text: \1", v1:=rc);
      IF rc = 0 THEN
         DebugFmt(message:=" Name: "+str);
      END_IF;

      rc := navFormReceiveReadNumber(id := ID_AGE, number := n);
      DebugFmt(message:="Read number: \1", v1:=rc);
      IF rc = 0 THEN
         DebugFmt(message:=" Age: \4", v4 := n);
      END_IF;


      FOR i := 1 TO 30 DO
         rc := navFormReceiveReadSelect(id := ID_OPTIONS, index:=i,
option:=n);
         DebugFmt(message:="Read option \1: \2", v1:=i, v2:=rc);
         IF rc < 0 THEN
            EXIT;
         END_IF;
         IF n = ID_OPT_1 THEN
            DebugMsg(message:=" Option 1 selected");
         END_IF;
         IF n = ID_OPT_2 THEN
            DebugMsg(message:=" Option 2 selected");
         END_IF;
      END_FOR;

      rc := navFormReceiveReadDateTime(id := ID_DATE, linsec := n);
      DebugFmt(message:="Read date: \1", v1:=rc);
      IF rc = 0 THEN
         DebugFmt(message:=" Date: \4", v4 := n);
         linsec := linsec + n;
      END_IF;

      rc := navFormReceiveReadDateTime(id := ID_TIME, linsec := n);
      DebugFmt(message:="Read time: \1", v1:=rc);
      IF rc = 0 THEN
         DebugFmt(message:=" Time: \4", v4 := n);
         linsec := linsec + n;
      END_IF;

      IF linsec <> 0 THEN
         DebugMsg(message:="Date/Time: "+TimeString(linsec:=linsec));
      END_IF;


      rc := navFormReceiveReadStop(id := ID_STOP, stop := stop);
      DebugFmt(message:="Read stop: \1", v1:=rc);
      IF rc = 0 THEN
         DebugFmt(message:=" Stop: \1", v1 := stop);
      END_IF;

      // Done with the form
      rc := navFormReceiveDone();
      DebugFmt(message:="navFormReceiveDone: \1", v1:=rc);
   END_IF;
```

```
END_FUNCTION;


THREAD_BLOCK navMonitor;
VAR
   event : INT := 0;
   rc    : INT;
END_VAR;

   WHILE event <> -1 DO
      event := navWaitEvent(timeout := -1);
      DebugFmt(message := "navWaitEvent - event=\1", v1 := event);
      CASE event OF
      1:
         msgAck();
         IF msgAck.ready THEN
            DebugMsg(message := "*** message reply received ***");
            DebugFmt(message := "-msgid=\1", v1 := msgAck.ID);
            DebugMsg(message := "-time= "+TimeString(linsec:=msgAck.time));
            CASE msgAck.reply OF
               1: DebugMsg(message := "-reply=[OK]");
               2: DebugMsg(message := "-reply=[YES]");
               3: DebugMsg(message := "-reply=[NO]");
            ELSE
               DebugFmt(message := "-reply=\1", v1 := msgAck.reply);
            END_CASE;
         END_IF;
      2:
         msgRec();
         IF msgRec.ready THEN
            DebugMsg(message := "*** message received ***");
            DebugMsg(message := "-time="+TimeString(linsec:=msgRec.time));
            DebugFmt(message := "-last=\1", v1 := msgRec.id);
            DebugMsg(message := "-text=" + msgRec.message);
         END_IF;

      3:
         msgSts();
         IF msgSts.ready THEN
            DebugMsg(message := "*** message status received ***");
            DebugFmt(message := "-msgid=\1", v1 := msgSts.ID);
            CASE msgSts.status OF
               1: DebugMsg(message := "-status=Unread");
               2: DebugMsg(message := "-status=Read");
               3: DebugMsg(message := "-status=Deleted/Not found");
            ELSE
               DebugFmt(message := "-status=\1", v1 := msgSts.status);
            END_CASE;
         END_IF;
      4:
         etaRec();
         IF etaRec.ready THEN
            DebugMsg(message := "*** ETA received ***");
            DebugFmt(message := "-time=     \4", v4 := etaRec.time);
            DebugFmt(message := "-distance= \4", v4 := etaRec.distance);
            DebugFmt(message := "-stop=     \1", v1 := etaRec.id);
            DebugFmt(message := "-latitude= \4", v4 := etaRec.latitude);
            DebugFmt(message := "-longitude=\4", v4 := etaRec.longitude);
         END_IF;

      5:
         stopRcv();
         IF stopRcv.ready THEN
```

```
            DebugMsg(message := "*** stop status received ***");
            DebugFmt(message := "-stop=\1", v1 := stopRcv.ID);
            DebugFmt(message := "-index=\1", v1 := stopRcv.index);
            CASE stopRcv.status OF
               1: DebugMsg(message := "-state=Active");
               2: DebugMsg(message := "-state=Done");
               3: DebugMsg(message := "-state=Unread - inactive");
               4: DebugMsg(message := "-state=Read - inactive");
               5: DebugMsg(message := "-state=Deleted/Not found");
            ELSE
               DebugFmt(message := "-state=\1", v1 := stopRcv.status);
            END_CASE;
         END_IF;
      6:
         userID();
         IF userID.status > 0 THEN
            DebugMsg(message := "*** user ID received ***");
            DebugFmt(message := "-time=\4: "+TimeString( linsec
:=userID.time), v4 := userID.time);
            DebugFmt(message := "-idx= \1", v1 := userID.index);
            DebugMsg(message := "-user=" + userID.user);
            DebugMsg(message := "-pass=" + userID.password);
         END_IF;
         IF userID.status = 2 THEN
            navUserIDAuthenticate(accept := TRUE);
         END_IF;
      7:
         userStatus();
         IF userStatus.ready THEN
            DebugMsg(message := "*** user status received ***");
            DebugFmt(message := "-time=\4: "+TimeString( linsec :=
userStatus.time), v4 := userStatus.time);
            DebugFmt(message := "-status=\1", v1 := userStatus.status);
         END_IF;

      8: DebugMsg(message := "GPI ID: " + navGPIReceiveID());
      9: ReadGPIProgress();

      14:
         FormSubmit();

      15:
         speedAlert();
         IF speedAlert.ready THEN
            DebugMsg(message := "*** Speed limit alert received ***");
            DebugFmt(message := "-time=       \4: "+TimeString( linsec :=
speedAlert.time), v4 := speedAlert.time);
            DebugFmt(message := "-type=       \1", v1 := speedAlert.type);
            DebugFmt(message := "-lat=        \4", v4 := speedAlert.latitude);
            DebugFmt(message := "-lon=        \4", v4 :=
speedAlert.longitude);
            DebugFmt(message := "-speed=      \1", v1 := speedAlert.speed);
            DebugFmt(message := "-speed limit=\1", v1 :=
speedAlert.speed_limit);
            DebugFmt(message := "-max speed=  \1", v1 :=
speedAlert.max_speed);

         END_IF;

      129: DebugMsg(message := "Request for refresh of quick messages");
      130: DebugMsg(message := "Request for refresh of message replies");
      131: DebugMsg(message := "Request for refresh of user status");
      132:
            DebugMsg(message := "Navigation device present");
```

```
            DebugFmt(message:="version = \1", v1:=navVersion ());
            DebugFmt(message:="API     = \1", v1:=navGetAPILevel ());
            // Delete all forms from the navigation device
            navDeleteData(type := 10);
            // Create custom form
            createForm();

      133: DebugMsg(message := "Navigation device not present");

      ELSE
         DebugFmt(message := "navWaitEvent - event=\1", v1 := event);
      END_CASE;
   END_WHILE;
END_THREAD_BLOCK;

PROGRAM form;
// These are the local variables of the program block
VAR

   rc     : INT;
   navMon : navMonitor;

END_VAR;
   // The next code will only be executed once after the program starts

   rc:= navOpen(port:=1);
   DebugFmt(message:="navOpen = \1", v1:=rc);

   // Start monitor thread
   navMon();

BEGIN
// Code from this point until END will be executed repeatedly


END;

END_PROGRAM;
```

4.2.31.11.3 navFormAbort (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 12

This function will abort the construction of a custom form.

Input:
None.

Returns: **INT**
0 - Success.
-1 - The form was not open.

Declaration:

```
FUNCTION navFormAbort : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Abort construction of a form.
   navFormAbort();
   ...
END;
END_PROGRAM;
```

4.2.31.11.4 navFormAddText (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 12

This function will add a text field to the form that is currently being constructed.

This field can be read from a received form by using navFormReceiveReadText.

Input:

id : DINT (1..2147483647)
Unique ID to identify the field.

title : STRING
The title of the field. Max. 50 characters.

desc : STRING
The description of the field to show below the title while no value has been entered. Max. 60 characters.

required : BOOL (*default FALSE*)
Set to true to make the field required.

max_length : INT (1..100, *default 100*)
Maximum length of the string that can be entered in the text field.

placeholder : STRING
The placeholder/help text to show in the field until a value is entered. Max 30 characters.

Returns: INT

0 - Success.
-1 - The form could not be found. Call navFormCreate first.
-3 - There is not space for more fields on the form.
-8 - Invalid parameter.

Declaration:

```
FUNCTION navFormAddText : INT;
VAR_INPUT
   id         : DINT;
   title      : STRING;
   desc       : STRING;
   required   : BOOL := FALSE;
```

```
   max_length : INT := 100;
   placeholder: STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;
VAR
   rc : INT;
END_VAR;
BEGIN
   ...
   // Add text field to the form:
   rc := navFormAddText(id := 1, title:="Name", desc:="Your name",
                        placeholder := "<Name>", required := TRUE);
   DebugFmt(message := "Add text=\1", v1 := rc);
   ...
END;
END_PROGRAM;
```

4.2.31.11.5 navFormAddNumber (Function)

Architecture:	**NX32 / NX32L**
Device support:	**MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5**
Firmware version:	**4.70 / 1.30.00**
Nav. API level:	**12**

This function will add a number field to the form that is currently being constructed.

This field can be read from a received form by using navFormReceiveReadNumber.

Input:

id : DINT (1..2147483647)
Unique ID to identify the field.

title : STRING
The title of the field. Max. 50 characters.

desc : STRING
The description of the field to show below the title while no value has been entered. Max. 60 characters.

required : BOOL (*default FALSE*)
Set to true to make the field required.

max_length : INT (1..10, *default 10*)
Maximum length of the number that can be entered in the number field.

placeholder : STRING
The placeholder/help text to show in the field until a value is entered. Max 30 characters.

min_val : DINT (*default -2147483648*)
Minimum value that can be entered in the number field.

max_val : DINT (*default 2147483647*)
Maximum value that can be entered in the number field.

Returns: INT

0 - Success.
-1 - The form could not be found. Call navFormCreate first.
-3 - There is not space for more fields on the form.
-8 - Invalid parameter.

Declaration:

```
FUNCTION navFormAddNumber : INT;
VAR_INPUT
   id          : DINT;
   title       : STRING;
   desc        : STRING;
   required    : BOOL := FALSE;
   max_length  : INT := 10;
   placeholder: STRING;
   min_val     : DINT := -2147483648;
   max_val     : DINT :=  2147483647;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;
VAR
   rc : INT;
END_VAR;
BEGIN
   ...
   // Add number field to the form
   rc := navFormAddNumber(id:=2, title:="Age", desc:="Your age",
placeholder:="<years>",
                          min_val:=0, max_val := 120, max_length := 3);
   DebugFmt(message := "Add number=\1", v1 := rc);
   ...
END;
END_PROGRAM;
```

4.2.31.11.6 navFormAddSelect (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 12

This function will add a selection field to the form that is currently being constructed.
The selection field must contain 1-30 options that are added with navFormAddOption.
It supports both single selection (radio buttons) and multiple selections (check boxes).

The selected options can be read from a received form by using navFormReceiveReadSelect.

Input:

id : DINT (1..2147483647)
Unique ID to identify the field.

title : STRING
The title of the field. Max. 50 characters.

desc : STRING
The description of the field to show below the title while no value has been entered. Max. 60 characters.

required : BOOL (*default FALSE*)
Set to true to make the field required.

multi : BOOL (*default FALSE*)
Set to true to allow selection of multiple options.

Returns: INT

0 - Success.
-1 - The form could not be found. Call navFormCreate first.
-3 - There is not space for more fields on the form.
-8 - Invalid parameter.

Declaration:

```
FUNCTION navFormAddSelect : INT;
VAR_INPUT
   id          : DINT;
   title       : STRING;
   desc        : STRING;
   required    : BOOL := FALSE;
   multi       : BOOL := FALSE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;
VAR
   rc : INT;
END_VAR;
BEGIN
   ...
   // Add a selection field with two options to the form.
   rc := navFormAddSelect(id := 3, title:="Options", desc:="Select
options",multi:=TRUE);
   DebugFmt(message := "Add select=\1", v1 := rc);

   rc := navFormAddOption(id:=1, select_id:=3, title:="Option 1");
   DebugFmt(message := "Add option=\1", v1 := rc);

   rc := navFormAddOption(id:=2, select_id:=3, title:="Option 2");
   DebugFmt(message := "Add option=\1", v1 := rc);
   ...
END;
END_PROGRAM;
```

4.2.31.11.7 navFormAddOption (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 12

This function will add an option to a selection field on the form that is currently being constructed.

Input:

id : DINT (1..2147483647)
Unique ID to identify the option.

select_id : DINT (1..2147483647)
The ID of the selection field to add this option to.

title : STRING
The title of the option. Max. 40 characters.

selected : BOOL (*default FALSE*)
Set to true to make the option selected by default.
If the selection field uses single selection, marking multiple options as selected will cause an invalid parameter error.

Returns: INT

0	- Success.
-1	- The form could not be found. Call navFormCreate first.
-3	- There is not space for more options on the form.
-7	- Selection field could not be found.
-8	- Invalid parameter.
-10	- There is not space for more options in the selection field.

Declaration:

```
FUNCTION navFormAddOption : INT;
VAR_INPUT
   id         : DINT;
   select_id  : DINT;
   title      : STRING;
   selected   : BOOL := FALSE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;
VAR
   rc : INT;
END_VAR;
BEGIN
   ...
   // Add a selection field with two options to the form.
   rc := navFormAddSelect(id := 3, title:="Options", desc:="Select
options",multi:=TRUE);
   DebugFmt(message := "Add select=\1", v1 := rc);

   rc := navFormAddOption(id:=1, select_id:=3, title:="Option 1");
   DebugFmt(message := "Add option=\1", v1 := rc);

   rc := navFormAddOption(id:=2, select_id:=3, title:="Option 2");
   DebugFmt(message := "Add option=\1", v1 := rc);
   ...
END;
END_PROGRAM;
```

4.2.31.11.8 navFormAddDate (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 12

This function will add a date field to the form that is currently being constructed.

This field can be read from a received form by using navFormReceiveReadDateTime.

Input:

id : DINT (1..2147483647)
Unique ID to identify the field.

title : STRING
The title of the field. Max. 50 characters.

desc : STRING
The description of the field to show below the title while no value has been entered. Max. 60 characters.

required : BOOL (*default FALSE*)
Set to true to make the field required.

linsec : DINT (*default -1*)
The linsec of the default date. Set to -1 to use the current date. Must be after 1990-01-01.

Returns: INT

0 - Success.
-1 - The form could not be found. Call navFormCreate first.
-3 - There is not space for more fields on the form.
-8 - Invalid parameter.

Declaration:

```
FUNCTION navFormAddDate : INT;
VAR_INPUT
   id         : DINT;
   title      : STRING;
   desc       : STRING;
   required   : BOOL := FALSE;
   linsec     : DINT := -1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;
VAR
   rc : INT;
END_VAR;
BEGIN
   ...
   // Add date field to the form.
   rc := navFormAddDate(id:=4, title:="Date", desc:="Select date");
```

```
   DebugFmt(message := "Add date=\1", v1 := rc);
   ...
END;
END_PROGRAM;
```

4.2.31.11.9 navFormAddTime (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 12

This function will add a time field to the form that is currently being constructed.

This field can be read from a received form by using navFormReceiveReadDateTime.

Input:

id : DINT (1..2147483647)
Unique ID to identify the field.

title : STRING
The title of the field. Max. 50 characters.

desc : STRING
The description of the field to show below the title while no value has been entered. Max. 60 characters.

required : BOOL (*default FALSE*)
Set to true to make the field required.

linsec : DINT (*default -1*)
The linsec of the default time. Set to -1 to use the current time.

Returns: INT

0 - Success.
-1 - The form could not be found. Call navFormCreate first.
-3 - There is not space for more fields on the form.
-8 - Invalid parameter.

Declaration:

```
FUNCTION navFormAddTime : INT;
VAR_INPUT
   id          : DINT;
   title       : STRING;
   desc        : STRING;
   required    : BOOL := FALSE;
   linsec      : DINT := -1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;
VAR
   rc : INT;
END_VAR;
```

```
BEGIN
   ...
   // Add time field to the form.
   rc := navFormAddTime(id:=5, title:="Time", desc:="Select time");
   DebugFmt(message := "Add time=\1", v1 := rc);
   ...
END;
END_PROGRAM;
```

4.2.31.11.10 navFormAddStop (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 12

This function will add a stop selection list to the form that is currently being constructed.

The selected stop can be read from a received form by using navFormReceiveReadStop.

Input:

id : DINT (1..2147483647)
Unique ID to identify the field.

title : STRING
The title of the field. Max. 50 characters.

desc : STRING
The description of the field to show below the title while no value has been entered. Max. 60 characters.

required : BOOL (*default FALSE*)
Set to true to make the field required.

Returns: INT

0	- Success.
-1	- The form could not be found. Call navFormCreate first.
-3	- There is not space for more fields on the form.
-8	- Invalid parameter.

Declaration:

```
FUNCTION navFormAddStop : INT;
VAR_INPUT
   id         : DINT;
   title      : STRING;
   desc       : STRING;
   required   : BOOL := FALSE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;
VAR
   rc : INT;
END_VAR;
```

```
BEGIN
   ...
   // Add a stop field to the form.
   rc := navFormAddStop(id:=6, title:="Stop", desc:="Select stop");
   DebugFmt(message := "Add stop=\1", v1 := rc);
   ...
END;
END_PROGRAM;
```

4.2.31.11.11 navFormUpload (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 12

This function will upload a form created with navFormCreate to the navigation device.
The upload progress will be reported with GPI progress events that can be read with navGPIProgressReceive.
When the upload is completed, navGPIProgressReceive will report any errors with the form.
See navFormuploadFile for a list of the additional error codes.

Input:

None.

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-10	- The form is too large to upload. Call navFormAbort and try again with fewer fields.
-11	- Forms are not supported.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navFormUpload : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // upload form
   navFormUpload();
   ...
END;
END_PROGRAM;
```

4.2.31.11.12 navFormUploadFile (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 12

This function will start to transfer a custom form file to the navigation device.
A "GPI progress" event is raised when there are changes to the progress and/or status of the file transfer. See also navWaitEvent for event # 9.
The custom form file is an XML file that follows the format specified in "custom_forms.xsd", which is part of the Garmin Fleet Management SDK, available at http://developer.garmin.com/fleet-management/overview/.
The file can optionally be compressed with the GZIP format.

When the upload is completed, navGPIProgressReceive will report any errors with the form.
In addition to the common errors, the following errors can happen:

267	- Form element is incomplete
268	- Form title length is too long
269	- Item element is incomplete
270	- An Item elements parent is not a Form element
271	- Item ID is used more than once
272	- Item title length is too long
273	- Item default value length is too long
274	- Item count is out of range
275	- A Type elements parent is not an Item element
276	- Type count is out of range
277	- Text length is out of range
278	- Placeholder text is too long
279	- Integer length is out of range
280	- Minimum integer value range error
281	- Maximum integer value range error
282	- Minimum integer value is greater than the maximum integer value
283	- Option element is incomplete
284	- An Option elements parent is not a single select or multiple select element
285	- Option ID is used more than one time for a given Item
286	- Option title length is too long
287	- Option count is out of range for an item
288	- Month value is out of range
289	- Day value is out of range
290	- Year value is out of range
291	- Number of days in the month is not valid
292	- “Use Current” is set to False and no date was specified
293	- Hour value is out of range
294	- Minute value is out of range
295	- Second value is out of range
296	- “Use Current” is set to False and no time was specified
297	- Unknown form Item type
298	- Picture count is out of range
357-395	- XML Parsing Error. There can be many causes for this error, a common cause is that it contains invalid characters.
461	- Exceeds 30 forms on device

Note: To create the forms on the RTCU device, use the navFormCreate and navFormUpload functions instead.

Input:

filename : STRING
The name and path of the file to transfer.

ID : STRING
A small text message to identify the file.
Must be between 1 and 16 characters.

Returns: INT

0 - Success.
-1 - Navigation interface is not open.
-2 - Error communicating with navigation device.
-3 - File not found.
-4 - The navigation device rejected the command.
-8 - Invalid ID.
-11 - Forms are not supported.
-12 - Navigation interface is busy.

Declaration:

```
FUNCTION navFormUploadFile : INT;
VAR_INPUT
   filename : STRING;
   id       : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Start transfer of custom form
   navFormUploadFile(filename := "form01.xml", id := "Form 1");
   ...
END;
END_PROGRAM;
```

4.2.31.11.13 navFormDelete (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 12

This function will delete a form from the navigation device.
To delete all the forms, use navDeleteData.

Input:

id : DINT
ID of the form to delete.

Returns: INT

0 - Success.
-1 - Navigation interface is not open.
-2 - Error communicating with navigation device.
-4 - The navigation device rejected the command.
-11 - Forms are not supported.
-12 - Navigation interface is busy.

Declaration:

```
FUNCTION navFormDelete : INT;
VAR_INPUT
   id       : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Delete form 55
   navFormDelete(id := 55);
   ...
END;
END_PROGRAM;
```

4.2.31.11.14 navFormMove (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 12

This function will move a form on the navigation device.
Depending on the position of the other forms, it may cause other forms to change position as well.

Input:

id : DINT
ID of the form to move.

new_pos : INT (1..32767)
The new position of the form.

Returns: INT

>0	- The position of the form.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-4	- The navigation device rejected the command.
-8	- Invalid position.
-11	- Forms are not supported.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navFormMove : INT;
VAR_INPUT
   id       : DINT;
   new_pos  : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Move form 55 to position 10
   navFormMove(id := 55, new_pos := 10);
   ...
```

```
END;
END_PROGRAM;
```

4.2.31.11.15 navFormGetPos (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 12

This function retrieves the position of a form on the navigation device.

Input:

id : DINT
ID of the form to get the position of.

Returns: INT

>0	- The position of the form
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-11	- Forms are not supported.
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION navFormGetPos : INT;
VAR_INPUT
   id       : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;
VAR
   rc : INT;
END_VAR;
BEGIN
   ...
   // Get position of form 55
   rc := navFormGetPos(id := 55);
   ...
END;
END_PROGRAM;
```

4.2.31.11.16 navFormShow (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 21

This function will show a form on the navigation device. This can e.g. be used to show the correct form when the destination is reached, so that there is no need to look through the list of forms to find the correct one.

Note that some navigation devices do not support showing the form while the user is in the main menu or in some of the applications on the navigation device.
If the user must be notified to fill out the form, one possibility is to also call navPopupAlert.

Input:
id : DINT
ID of the form to show.

Returns: INT
0 - Success.
-1 - Navigation interface is not open.
-2 - Error communicating with navigation device.
-4 - The navigation device rejected the command.
-11 - Showing a form is not supported.
-12 - Navigation interface is busy.

Declaration:
```
FUNCTION navFormShow : INT;
VAR_INPUT
   id       : DINT;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // show form 55
   navFormShow(id := 55);
   ...
END;
END_PROGRAM;
```

4.2.31.11.17 navFormReceive (Functionblock)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 12

---

navFormReceive returns information about a form that has been received from the navigation device.

Note that the received form is only valid after a "Form received" event is raised; the event will not be removed until the navFormReceiveDone function is called.
A "Form received" event is raised when a custom form is submitted by the user on the navigation device.

To create custom forms, see navFormCreate, navFormUpload and navFormUploadFile.

Input:
None.

Output:

ready : BOOL
TRUE: If data is ready.
FALSE: If data is not ready.

count : SINT
The number of valid items available in the form.

ID : DINT
The ID of the form that was received.
This is the value that is specified in navFormCreate when the form was created.

version : DINT
The version number of the received form.

submit_id : DINT
An unique ID for the received form. When a custom form is submitted multiple times, this can be used for keeping track of each submission.

time : DINT
The timestamp when the form was submitted.

Declaration:

```
FUNCTION_BLOCK navFormReceive;
VAR_OUTPUT
   ready     : BOOL;
   count     : SINT;
   ID        : DINT;
   version   : DINT;
   submit_id : DINT;
   time      : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   form : navFormReceive;
END_VAR;

FUNCTION ReadForm;
   form();
   IF form.ready THEN
      DebugMsg(message := "*** submitted form received ***");
      DebugFmt(message := "-id=\4", v4 := form.ID);
      DebugFmt(message := "-version=\4", v4 := form.version);
      DebugFmt(message := "-time=\4", v4 := form.time);

      navFormReceiveDone();
   END_IF;
END_FUNCTION;

THREAD_BLOCK navMonitor;
VAR
   event : INT := 0;
END_VAR;

WHILE event <> -1 DO
   event := navWaitEvent(timeout := -1);
```

```
   CASE event OF
      ...
      14: ReadForm();
      ...
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;
```

4.2.31.11.18 navFormReceiveDone (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 12

This function must be called when the program is done using the received form.
Note that the received form is only valid after a "Form received" event is raised; the event will be removed when this function is called.

Input:

None.

Returns: INT

0 - Success.
-1 - Navigation interface is not open.
-3 - No received form could be found.

Declaration:

```
FUNCTION navFormReceiveDone : INT;
```

Example:

```
INCLUDE rtcu.inc

VAR
   form : navFormReceive;
   str  : STRING;
END_VAR;

FUNCTION ReadForm;
   form();
   IF form.ready THEN
      DebugMsg(message := "*** submitted form received ***");
      DebugFmt(message := "-id=\4", v4 := form.ID);
      DebugFmt(message := "-version=\4", v4 := form.version);
      DebugFmt(message := "-time=\4", v4 := form.time);

      // Read field 13
      navFormReceiveReadText(ID := 13, text := str);

      navFormReceiveDone();
   END_IF;
END_FUNCTION;

THREAD_BLOCK navMonitor;
VAR
   event : INT := 0;
END_VAR;
```

```
WHILE event <> -1 DO
   event := navWaitEvent(timeout := -1);
   CASE event OF
      ...
      14: ReadForm();
      ...
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;
```

4.2.31.11.19 navFormReceiveField (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 12

This function will read the ID and type of a field on a received form.
Note that the received form is only valid after a "Form received" event is raised; the event will not be removed until the navFormReceiveDone function is called.

This can be used to determine the available fields on the received form.

Input:

index : INT (1..30 , *default 1*)
Index of the field to get information about. The maximum index for the specific form can be read in navFormReceive.count.

Output:

id : DINT
The unique id of the field.

type : SINT
The type of field:

- 1 - Text field.
- 2 - Number field.
- 3 - Selection field with single selection.
- 4 - Selection field with multiple selections.
- 5 - Date
- 6 - Time
- 7 - Stop

Returns: INT

- 0 - Success.
- -1 - Navigation interface is not open.
- -3 - No received form could be found.
- -7 - The field could not be found.

Declaration:

```
FUNCTION navFormReceiveField : INT;
VAR_INPUT
   index       : INT := 1;
   id          : ACCESS DINT;
   type        : ACCESS SINT;
END_VAR;
```

Example:

```
//-------------------------------------------------------------------------
-
// form.vpl, created 2018-11-28 14:45
// Shows how to use navFormReceiveField to iterate through the received
fields.
//-------------------------------------------------------------------------
-
INCLUDE rtcu.inc

//  Input variables that can be configured via the configuration dialog (These
are global)
VAR_INPUT

END_VAR;

//  Output variables that can be configured via the configuration dialog
(These are global)
VAR_OUTPUT

END_VAR;

//  These are the global variables of the program
VAR
   msgSts          : navMessageStatusReceive;
   msgAck          : navMessageReplyReceive;
   msgRec          : navMessageReceiveX;
   etaRec          : navETAReceive;
   userStatus      : navUserStatusReceive;
   stopRcv         : navStopReceive;
   userID          : navUserIDReceiveX;
   gpiRcv          : navGPIProgressReceive;
   formReceive     : navFormReceive;
   speedAlert      : navSpeedLimitAlert;
   clock           : clockLinsecToTime;
END_VAR;

// Definition of the field IDs
#DEFINE ID_NAME      1
#DEFINE ID_AGE       2
#DEFINE ID_OPTIONS   3
#DEFINE ID_DATE      4
#DEFINE ID_TIME      5
#DEFINE ID_STOP      6

// Option IDs
#DEFINE ID_OPT_1     1
#DEFINE ID_OPT_2     2

FUNCTION TimeString : STRING;
VAR_INPUT
   linsec : DINT;
END_VAR;
   clock(linsec := linsec);
   TimeString := strFormat(format := "\1.\2.\3, ", v1 := clock.day, v2 :=
clock.month, v3 := clock.year);
   TimeString := TimeString + strFormat(format := "\1:\2:\3", v1 :=
clock.hour, v2 := clock.minute, v3 := clock.second);
END_FUNCTION;


FUNCTION ReadGPIProgress;
   gpiRcv();
```

```
   IF gpiRcv.ready THEN
      DebugMsg(message := "*** progress received ***");
      DebugFmt(message := "-status=  \1", v1 := gpiRcv.status);
      DebugFmt(message := "-progress=\1", v1 := gpiRcv.progress);
      DebugFmt(message := "-type=    \1", v1 := gpiRcv.type);
      DebugFmt(message := "-error=   \1", v1 := gpiRcv.error);
   END_IF;
END_FUNCTION;

FUNCTION createForm:INT;
VAR
   rc: INT;
END_VAR;
   // Begin constructing the form.
   rc := navFormCreate(id := 7, title:="Form", version := 1);
   DebugFmt(message := "Form create=\1", v1 := rc);

   // Add fields to the form:
   rc := navFormAddText(id := ID_NAME, title:="Name", desc:="Your name",
                        placeholder := "<Name>", required := TRUE);
   DebugFmt(message := "Add text=\1", v1 := rc);

   rc := navFormAddNumber(id:=ID_AGE, title:="Age", desc:="Your age",
placeholder:="<years>",
                        min_val:=0, max_val := 120, max_length := 3);
   DebugFmt(message := "Add number=\1", v1 := rc);

   rc := navFormAddSelect(id := ID_OPTIONS, title:="Options", desc:="Select
the options", multi:=TRUE);
   DebugFmt(message := "Add select=\1", v1 := rc);

   rc := navFormAddOption(id:=ID_OPT_1, select_id:=ID_OPTIONS, title:="Option
1");
   DebugFmt(message := "Add option=\1", v1 := rc);

   rc := navFormAddOption(id:=ID_OPT_2, select_id:=ID_OPTIONS, title:="Option
2");
   DebugFmt(message := "Add option=\1", v1 := rc);

   rc := navFormAddDate(id:=ID_DATE, title:="Date");
   DebugFmt(message := "Add date=\1", v1 := rc);

   rc := navFormAddTime(id:=ID_TIME, title:="Time");
   DebugFmt(message := "Add time=\1", v1 := rc);

   rc := navFormAddStop(id:=ID_STOP, title:="Stop", desc:="Select the stop");
   DebugFmt(message := "Add stop=\1", v1 := rc);


   // Upload the form to the navitagion device
   rc := navFormupload();
   DebugFmt(message := "Upload=\1", v1 := rc);

END_FUNCTION;


FUNCTION FormSubmit;
VAR
   rc    : INT;
   i,o   : INT;
   str   : STRING;
   n     : DINT;
   linsec: DINT := 0;
   id    : DINT;
```

```
   stop  : INT;
   type  : SINT;
END_VAR;
   formReceive();
   IF(formReceive.ready) THEN
       DebugMsg(message := "*** form received ***");
       DebugFmt(message := "-ID=      \4", v4 := formReceive.id);
       DebugFmt(message := "-sub id=  \4", v4 := formReceive.submit_id);
       DebugFmt(message := "-version= \4", v4 := formReceive.version);
       DebugFmt(message := "-time=    "+TimeString( linsec :=
formReceive.time));

      IF(formReceive.id <> 7) THEN
            DebugFmt(message:="Unknown form ID \4", v4:=formReceive.id);
            rc := navFormReceiveDone();
            DebugFmt(message:="navFormReceiveDone: \1", v1:=rc);
            RETURN;
      END_IF;

      IF(formReceive.version <> 1) THEN
            DebugFmt(message:="Unknown form version \4",
v4:=formReceive.version);
            rc := navFormReceiveDone();
            DebugFmt(message:="navFormReceiveDone: \1", v1:=rc);
            RETURN;
      END_IF;


      FOR i := 1 TO formReceive.count DO
         rc := navFormReceiveField(index := i, id := id, type :=type );
         DebugFmt(message:="navFormReceiveField(\2): \1, id \4, type \3",
         v1:=rc, v2:=i, v3:=type, v4:=id);

         CASE INT(id) OF
         ID_NAME:
            rc := navFormReceiveReadText(id := ID_NAME, text := str);
            DebugFmt(message:="Read text: \1", v1:=rc);
            IF rc = 0 THEN
               DebugFmt(message:=" Name: "+str);
            END_IF;
         ID_AGE:
            rc := navFormReceiveReadNumber(id := ID_AGE, number := n);
            DebugFmt(message:="Read number: \1", v1:=rc);
            IF rc = 0 THEN
               DebugFmt(message:=" Age: \4", v4 := n);
            END_IF;
         ID_OPTIONS:
            FOR o := 1 TO 30 DO
               rc := navFormReceiveReadSelect(id := ID_OPTIONS, index:=o,
option:=n);
               DebugFmt(message:="Read option \1: \2", v1:=o, v2:=rc);
               IF rc < 0 THEN
                  EXIT;
               END_IF;
               IF n = ID_OPT_1 THEN
                  DebugMsg(message:=" Option 1 selected");
               END_IF;
               IF n = ID_OPT_2 THEN
                  DebugMsg(message:=" Option 2 selected");
               END_IF;
            END_FOR;
         ID_DATE:
            rc := navFormReceiveReadDateTime(id := ID_DATE, linsec := n);
            DebugFmt(message:="Read date: \1", v1:=rc);
```

```
            IF rc = 0 THEN
               DebugFmt(message:=" Date: \4", v4 := n);
               linsec := linsec + n;
            END_IF;
         ID_TIME:
            rc := navFormReceiveReadDateTime(id := ID_TIME, linsec := n);
            DebugFmt(message:="Read time: \1", v1:=rc);
            IF rc = 0 THEN
               DebugFmt(message:=" Time: \4", v4 := n);
               linsec := linsec + n;
            END_IF;
         ID_STOP:
            rc := navFormReceiveReadStop(id := ID_STOP, stop := stop);
            DebugFmt(message:="Read stop: \1", v1:=rc);
            IF rc = 0 THEN
               DebugFmt(message:=" Stop: \1", v1 := stop);
            END_IF;
         END_CASE;
      END_FOR;
      // Done with the form
      rc := navFormReceiveDone();
      DebugFmt(message:="navFormReceiveDone: \1", v1:=rc);
    END_IF;
END_FUNCTION;


THREAD_BLOCK navMonitor;
VAR
   event : INT := 0;
   rc    : INT;
END_VAR;

   WHILE event <> -1 DO
      event := navWaitEvent(timeout := -1);
      DebugFmt(message := "navWaitEvent - event=\1", v1 := event);
      CASE event OF
      1:
         msgAck();
         IF msgAck.ready THEN
            DebugMsg(message := "*** message reply received ***");
            DebugFmt(message := "-msgid=\1", v1 := msgAck.ID);
            DebugMsg(message := "-time= "+TimeString(linsec:=msgAck.time));
            CASE msgAck.reply OF
               1: DebugMsg(message := "-reply=[OK]");
               2: DebugMsg(message := "-reply=[YES]");
               3: DebugMsg(message := "-reply=[NO]");
            ELSE
               DebugFmt(message := "-reply=\1", v1 := msgAck.reply);
            END_CASE;
         END_IF;
      2:
         msgRec();
         IF msgRec.ready THEN
            DebugMsg(message := "*** message received ***");
            DebugMsg(message := "-time="+TimeString(linsec:=msgRec.time));
            DebugFmt(message := "-last=\1", v1 := msgRec.id);
            DebugMsg(message := "-text=" + msgRec.message);
         END_IF;

      3:
         msgSts();
         IF msgSts.ready THEN
            DebugMsg(message := "*** message status received ***");
            DebugFmt(message := "-msgid=\1", v1 := msgSts.ID);
```

```
            CASE msgSts.status OF
               1: DebugMsg(message := "-status=Unread");
               2: DebugMsg(message := "-status=Read");
               3: DebugMsg(message := "-status=Deleted/Not found");
            ELSE
               DebugFmt(message := "-status=\1", v1 := msgSts.status);
            END_CASE;
         END_IF;
      4:
         etaRec();
         IF etaRec.ready THEN
            DebugMsg(message := "*** ETA received ***");
            DebugFmt(message := "-time=     \4", v4 := etaRec.time);
            DebugFmt(message := "-distance= \4", v4 := etaRec.distance);
            DebugFmt(message := "-stop=     \1", v1 := etaRec.id);
            DebugFmt(message := "-latitude= \4", v4 := etaRec.latitude);
            DebugFmt(message := "-longitude=\4", v4 := etaRec.longitude);
         END_IF;

      5:
         stopRcv();
         IF stopRcv.ready THEN
            DebugMsg(message := "*** stop status received ***");
            DebugFmt(message := "-stop=\1", v1 := stopRcv.ID);
            DebugFmt(message := "-index=\1", v1 := stopRcv.index);
            CASE stopRcv.status OF
               1: DebugMsg(message := "-state=Active");
               2: DebugMsg(message := "-state=Done");
               3: DebugMsg(message := "-state=Unread - inactive");
               4: DebugMsg(message := "-state=Read - inactive");
               5: DebugMsg(message := "-state=Deleted/Not found");
            ELSE
               DebugFmt(message := "-state=\1", v1 := stopRcv.status);
            END_CASE;
         END_IF;
      6:
         userID();
         IF userID.status > 0 THEN
            DebugMsg(message := "*** user ID received ***");
            DebugFmt(message := "-time=\4: "+TimeString( linsec
:=userID.time), v4 := userID.time);
            DebugFmt(message := "-idx= \1", v1 := userID.index);
            DebugMsg(message := "-user=" + userID.user);
            DebugMsg(message := "-pass=" + userID.password);
         END_IF;
         IF userID.status = 2 THEN
            navUserIDAuthenticate(accept := TRUE);
         END_IF;
      7:
         userStatus();
         IF userStatus.ready THEN
            DebugMsg(message := "*** user status received ***");
            DebugFmt(message := "-time=\4: "+TimeString( linsec :=
userStatus.time), v4 := userStatus.time);
            DebugFmt(message := "-status=\1", v1 := userStatus.status);
         END_IF;

      8: DebugMsg(message := "GPI ID: " + navGPIReceiveID());
      9: ReadGPIProgress();

      14:
         FormSubmit();

      15:
```

```
         speedAlert();
         IF speedAlert.ready THEN
            DebugMsg(message := "*** Speed limit alert received ***");
            DebugFmt(message := "-time=       \4: "+TimeString( linsec :=
speedAlert.time), v4 := speedAlert.time);
            DebugFmt(message := "-type=       \1", v1 := speedAlert.type);
            DebugFmt(message := "-lat=        \4", v4 := speedAlert.latitude);
            DebugFmt(message := "-lon=        \4", v4 :=
speedAlert.longitude);
            DebugFmt(message := "-speed=      \1", v1 := speedAlert.speed);
            DebugFmt(message := "-speed limit=\1", v1 :=
speedAlert.speed_limit);
            DebugFmt(message := "-max speed=  \1", v1 :=
speedAlert.max_speed);

         END_IF;

      129: DebugMsg(message := "Request for refresh of quick messages");
      130: DebugMsg(message := "Request for refresh of message replies");
      131: DebugMsg(message := "Request for refresh of user status");
      132:
            DebugMsg(message := "Navigation device present");
            DebugFmt(message:="version = \1", v1:=navVersion ());
            DebugFmt(message:="API     = \1", v1:=navGetAPILevel ());
            // Delete all forms from the navigation device
            navDeleteData(type := 10);
            // Create custom form
            createForm();

      133: DebugMsg(message := "Navigation device not present");

      ELSE
         DebugFmt(message := "navWaitEvent - event=\1", v1 := event);
      END_CASE;
   END_WHILE;
END_THREAD_BLOCK;

PROGRAM form;
// These are the local variables of the program block
VAR

   rc     : INT;
   navMon : navMonitor;

END_VAR;
   // The next code will only be executed once after the program starts

   rc:= navOpen(port:=1);
   DebugFmt(message:="navOpen = \1", v1:=rc);

   // Start monitor thread
   navMon();

BEGIN
// Code from this point until END will be executed repeatedly


END;

END_PROGRAM;
```

4.2.31.11.20 navFormReceiveReadText (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 12

This function will read the value of a text field on a received form.
Note that the received form is only valid after a "Form received" event is raised; the event will not be removed until the navFormReceiveDone function is called.

Input:

id : DINT (1..2147483647)
ID of the field to read.

Output:

text : STRING
The value of the field.

Returns: INT

0 - Success.
-1 - Navigation interface is not open.
-3 - No received form could be found.
-7 - The field could not be found.

Declaration:

```
FUNCTION navFormReceiveReadText : INT;
VAR_INPUT
   id          : DINT;
   text        : ACCESS STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;
VAR
   str : STRING;
END_VAR;
BEGIN
   ...
   // Read field 13
   navFormReceiveReadText(ID := 13, text := str);
   ...
END;
END_PROGRAM;
```

4.2.31.11.21 navFormReceiveReadNumber (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 12

This function will read the value of a number field on a received form.

Note that the received form is only valid after a "Form received" event is raised; the event will not be removed until the navFormReceiveDone function is called.

Input:
id : DINT (1..2147483647)
ID of the field to read.

Output:
number : DINT
The value of the field.

Returns: INT
0 - Success.
-1 - Navigation interface is not open.
-3 - No received form could be found.
-7 - The field could not be found.

Declaration:
```
FUNCTION navFormReceiveReadNumber : INT;
VAR_INPUT
   id         : DINT;
   number     : ACCESS DINT;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM NavigationExample;
VAR
   val : DINT;
END_VAR;
BEGIN
   ...
   // Read field 14
   navFormReceiveReadNumber(ID := 14, number := val);
   ...
END;
END_PROGRAM;
```

4.2.31.11.22 navFormReceiveReadSelect (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 12

This function will read a selected option from a selection field on a received form.
Note that the received form is only valid after a "Form received" event is raised; the event will not be removed until the navFormReceiveDone function is called.
To read all the selected options of a field with multiple selections, call this function with incrementing index until all the selected options have been found.

Input:
id : DINT (1..2147483647)
ID of the field to read.

index : INT (1..30)
Index of the option to read.

Output:
option : DINT
The id of a selected option.

Returns: INT
>=0 - Success. The value is the number of selected options that have a larger index.
-1 - Navigation interface is not open.
-3 - No received form could be found.
-7 - The selection field could not be found.
-8 - The option could not be found.

Declaration:
```
FUNCTION navFormReceiveReadSelect : INT;
VAR_INPUT
   id          : DINT;
   index       : INT := 1;
   option      : ACCESS DINT;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM NavigationExample;
VAR
   val : DINT;
END_VAR;
BEGIN
   ...
   // Read the selection from field  15
   navFormReceiveReadSelect(ID := 15, index := 1, option := val);
   ...
END;
END_PROGRAM;
```

4.2.31.11.23 navFormReceiveReadDateTime (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 12

This function will read the value of a date or time field on a received form.
Note that the received form is only valid after a "Form received" event is raised; the event will not be removed until the navFormReceiveDone function is called.

Input:
id : DINT (1..2147483647)
ID of the field to read.

Output:
linsec : DINT
The value of the field.
For time fields, this is the number of seconds since midnight on 1980-01-01.

Returns: INT

0 - Success.
-1 - Navigation interface is not open.
-3 - No received form could be found.
-7 - The field could not be found.

Declaration:

```
FUNCTION navFormReceiveReadDateTime : INT;
VAR_INPUT
   id         : DINT;
   linsec     : ACCESS DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;
VAR
   date : DINT;
   time : DINT;
END_VAR;
BEGIN
   ...
   // Read the date from field 16
   navFormReceiveReadDateTime(ID := 16, linsec := date);
   // Read the time from field 17
   navFormReceiveReadDateTime(ID := 17, linsec := time);
   // Combine date and time in the date variable.
   date := date + time;
   ...
END;
END_PROGRAM;
```

4.2.31.11.24 navFormReceiveReadStop (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 12

This function will read the id of the selected stop from a field on a received form.
Note that the received form is only valid after a "Form received" event is raised; the event will not be removed until the navFormReceiveDone function is called.

Input:

id : DINT (1..2147483647)
ID of the field to read.

Output:

stop : INT
The id of the stop.

Returns: INT

0 - Success.
-1 - Navigation interface is not open.
-3 - No received form could be found.

-7 - The field could not be found.

Declaration:

```
FUNCTION navFormReceiveReadStop : INT;
VAR_INPUT
   id       : DINT;
   stop     : ACCESS INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;
VAR
   val : INT;
END_VAR;
BEGIN
   ...
   // Read field 18
   navFormReceiveReadStop(ID := 18, stop := val);
   ...
END;
END_PROGRAM;
```

4.2.31.12 Speed Limit Alerts

4.2.31.12.1 navSpeedLimitAlertSetup (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 4.70 / 1.30.00
Nav. API level: 8

This function will enable and configure the Speed Limit Alerts(SLA) on the navigation unit.
SLAs are used to alert the application of speed limit violations.
An "SLA" event is raised when the vehicle has exceeded the speed limit for the provided time.
See also navWaitEvent for event # 15.
If the database on the navigation device does not contain the speed limit, it will behave as if the speed limit is arbitrarily large.
Some navigation devices allow the user to update the posted speed. In that case the original speed limit will still be used for SLA.

Input:

mode : SINT (*default: 0*)
The mode of the SLA:

0 - Off. Do not monitor the speed limit.
1 - Car mode. Uses speed limits for cars.
2 - Truck mode. Uses speed limits for trucks.

alert_user : BOOL (*default: FALSE*)
Determines if the user should be alerted when the threshold is exceeded.

time_over : SINT (0..255, *default 10*)
The number of seconds the speed limit must be exceeded to trigger the alert.

time_under : SINT (0..255, *default 10*)
The number of seconds the speed must be kept under the speed limit before the alert is stopped.

threshold : INT (*default: 5*)
Speed in km/hr above or below the actual speed limit to consider speeding. Negative values should only be used for testing.

Returns: INT

```
   0   - Success.
  -1   - Navigation interface is not open.
  -2   - Error communicating with navigation device.
  -4   - The navigation device rejected the command.
  -7   - The selected mode is not supported on this device.
  -8   - Invalid parameter.
 -11   - SLA is not supported.
 -12   - Navigation interface is busy.
```

Declaration:

```
FUNCTION navSpeedLimitAlertSetup : INT;
VAR_INPUT
   mode       : SINT := 0;
   alert_user : BOOL := FALSE;
   time_over  : INT := 10;
   time_under : INT := 10;
   threshold  : INT := 5;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   // Create sensor 55
   navSpeedLimitAlertSetup(alert_user := TRUE, threshold := 10);
   ...
END;
END_PROGRAM;
```

4.2.31.12.2 navSpeedLimitAlert (Functionblock)

Architecture:	**NX32 / NX32L**
Device support:	**MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5**
Firmware version:	**4.70 / 1.30.00**
Nav. API level:	**8**

The function block returns information about Speed Limit Alerts(SLA).
Note that the SLA is only valid after an "SLA" event is raised; the event will be removed when this function block is called.
An "SLA" event is raised when SLA is enabled with navSpeedLimitAlertSetup and the speed has been exceeded.

Input:

None.

Output:

time : DINT
The timestamp for the alert.

latitude : DINT
The latitude of the position of the alert.

longitude : DINT
The longitude of the position of the alert.

speed : INT
The speed at the time of the alert.

speed_limit : INT
The speed limit at the time of the alert.

max_speed : INT
The maximum speed since the last alert.

type : SINT
The type of alert:

0 - Speeding has begun.
1 - Speed limit has changed.
2 - Speeding has ended.
3 - Internal error.
4 - Invalid alert. If over 50 alerts have been queued without being received by the RTCU device, the queue is cleared and this alert is added.
Can also happen if changing the SLA setup while driving.

ready : BOOL
TRUE: If data is ready.
FALSE: If data is not ready.

Declaration:

```
FUNCTION_BLOCK navSpeedLimitAlert;
VAR_OUTPUT
   ready       : BOOL;
   type        : SINT;
   time        : DINT;
   latitude    : DINT;
   longitude   : DINT;
   speed       : INT;
   speed_limit : INT;
   max_speed   : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   slaRcv : navSpeedLimitAlert;
END_VAR;

FUNCTION ReadSLA;
   slaRcv();
   IF slaRcv.ready THEN
      DebugMsg(message := "*** SLA received ***");
      DebugFmt(message := "-time=\4", v4 := slaRcv.time);
```

```
      DebugFmt(message := "-type=\1", v1 := slaRcv.type);
      DebugFmt(message := "-latitude=\4", v4 := slaRcv.latitude);
      DebugFmt(message := "-longitude=\4", v4 := slaRcv.longitude);
      DebugFmt(message := "-speed=\1", v1 := slaRcv.speed);
      DebugFmt(message := "-speed_limit=\1", v1 := slaRcv.speed_limit);
      DebugFmt(message := "-max_speed=\1", v1 := slaRcv.max_speed);
   END_IF;
END_FUNCTION;

THREAD_BLOCK navMonitor;
VAR
   event : INT := 0;
END_VAR;

WHILE event <> -1 DO
   event := navWaitEvent(timeout := -1);
   CASE event OF
      ...
      15: ReadSLA();
      ...
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;
```

4.2.31.13 Conversion

4.2.31.13.1 navPositionToSemicircles (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 1.40 / 1.00.00
Nav. API level: 1

This function will convert a GPS position from the standard RTCU format to the semicircle format.

Input:

pos : DINT
The latitude or longitude to convert.

Returns: DINT

The GPS position in semicircle format.

Declaration:

```
FUNCTION navPositionToSemicircles : DINT;
VAR_INPUT
   pos : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
```

```
   result : DINT;
END_VAR;

PROGRAM test;

BEGIN
   ...
   result := navPositionToSemicircles(pos := 55508400);
   // result will contain 666284640 after the call
   ...
END;
END_PROGRAM;
```

4.2.31.13.2 navSemicirclesToPosition (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, AX9, CX1 pro/flex/warp, SX1, MX2 turbo/encore/warp, AX9 turbo, NX-200, NX-400, NX-900, LX2, LX4, LX5
Firmware version: 1.40 / 1.00.00
Nav. API level: 1

This function will convert a GPS position from the semicircle format to the standard RTCU format.

Input:

pos : DINT
The latitude or longitude to convert.

Returns: DINT

The GPS position in standard RTCU format (ddmm.mmmm (multiplied by 10000)).

Declaration:

```
FUNCTION navSemicirclesToPosition : DINT;
VAR_INPUT
   pos : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   result : DINT;
END_VAR;

PROGRAM test;

BEGIN
   ...
   result := navSemicirclesToPosition(pos := 117210055);
   // result will contain 9494660 after the call
   ...
END;
END_PROGRAM;
```

4.2.32 net: Network

4.2.32.1 net: Network

The Network functions are used for establishing a connection either to the internet or to a local network.

The following network interfaces are supported:

IFACE	Name	Description
1	Mobile	Cellular connection.
2	LAN 1	Primary Ethernet connection.
3	LAN 2	Secondary / USB Ethernet connection.
4	WLAN	Wireless network connection.
5	AP	Wireless Access Point connection.

When using the 'net' API the external WiFi and Ethernet add-on modules are not supported.

The following defines are present representing each interface:

```
1       _NET_IFACE_MOBILE // _NET_IFACE_CELLULAR
2       _NET_IFACE_LAN1
3       _NET_IFACE_LAN2
4       _NET_IFACE_WLAN
5       _NET_IFACE_WLAN_AP
```

Network Address Translation (NAT)
This is a way to route socket connections between different network interfaces.
It is possible to configure a network interface to be a gateway to the Internet.
When this is done, NAT is used to masquerade outgoing socket connection, as if sending from the device.
Please note that only one network interface can be set as gateway at a time.
The NAT functionality is only available on devices with NX32L architecture.

Dynamic Host Configuration Protocol (DHCP) server
This server uses the DHCP to dynamically assign an IP address and other network configuration parameters to each device on a network.
The configuration parameters the DHCP server assigns are configured for each network interface.
The DHCP server is only available on devices with NX32L architecture.

Port Forwarding
This is a way to forward incoming connections from a network interface to a client device located on a different network interface.
The forwarding rules always apply to the network interface which is configured as the NAT gateway.
If the NAT gateway configuration is moved to a different network interface, the port forwarding rules will follow automatically.
For example:
If the WLAN network interface is configured as NAT Gateway, then only incoming connections from this interface will be forwarded.
Later the Mobile network interface is configured as NAT Gateway,and only incoming connections from this interface will be forwarded, no changes to the port forwarding rules are necessary.

If no rules are defined, no incoming socket connections will be forwarded to other network interfaces.

The following functions are available:

- netOpen — Opens a network interface.
- netClose — Closes a network interface.
- netConnected — Queries if an interface is connected to the network.
- netPresent — Determines if the network interface is available.
- netGetInformation — Returns information about a network interface.
- netGetStatistics — Returns statistics for a network interface.
- netGetSignalLevel — Returns the signal strength of a network interface.
- netPing — Ping a host (advanced)
- netPingS — Ping a host (simple)

The following functions are available for configuring the network interfaces:

- netSetLANParam — Sets TCP/IP configuration parameters for the LAN interface.
- netGetLANParam — Retrieves the TCP/IP configuration parameters for the LAN interface.
- netSetMobileParam — Sets TCP/IP configuration parameters for the Mobile interface.
- netGetMobileParam — Retrieves the TCP/IP configuration parameters for the Mobile interface.
- netSetWLANParam — Sets wireless network configuration parameters for the WLAN interface
- netGetWLANParam — Retrieves the wireless network configuration parameters for the WLAN interface.
- netGetWLANSecurity — Retrieves the security configuration for a wireless network.
- netWLANSecurityWPA — The parameters for WPA/WPA2 pre-shared key security
- netSetAPParam — Sets wireless network configuration parameters for the AP interface
- netGetAPParam — Retrieves the wireless network configuration parameters for the AP interface.
- netGetAPSecurity — Retrieves the security configuration for the AP interface

The following functions are available for network services:

- netGetMonitorParam — Retrieve the network monitoring service parameters for a network interface.
- netSetMonitorParam — Set the network monitoring service parameters for a network interface.
- netSetNATParam — Sets the NAT parameters for a network interface.
- netGetNATParam — Retrieve the NAT parameters for a network interface.
- netSetDHCPParam — Sets the DHCP parameters for a network interface.
- netGetDHCPParam — Retrieve the DHCP parameters for a network interface.
- netPortForwardClear — Clear all port forward rules.
- netPortForwardGet — Retrieve a port forward rule.
- netPortForwardSet — Set a port forward rule.
- netRouteAdd — Add a user route to the system.
- netRouteDel — Remove a user route from the system.
- netRouteGet — Get information about a user route.

4.2.32.2 netOpen (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 4.34 / 1.00.00

Opens a network interface and prepares it for use.
Must be called before any communication using the network interface can occur.
`netOpen(iface := 1)` is identical to gprsOpen(), and must be called after gsmPower.

Note that the connection to the network is not established when this function returns.
The netConnected() function will indicate when the connection to the network is established and ready for communication.

Input:

iface : SINT
The network interface to open. 1 = Mobile network, 2 = LAN network, etc. (See Network)

Returns: INT

0	- Success.
-1	- Unsupported interface.
-2	- Interface is powered off.
-3	- Interface can not be opened at the selected CPU speed (see pmSetSpeed).
-4	- Interface is unable to connect because of the configuration.
-5	- Interface is not accessible. This may be because the cellular connection is using the highspeed link. See gsmPower().

Declaration:

```
FUNCTION netOpen : INT;
VAR_INPUT
   iface : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
//  These are the global variables of the program
VAR
   netInfo  :  netGetInformation;
   iface    :  SINT := _NET_IFACE_LAN1; // Network interface to use: 1 for
Mobile, 2 for LAN.
END_VAR;

FUNCTION printInfo
   netInfo(iface:=iface);
   DebugFmt(message:="Network interface \1:", v1 := iface);
   DebugFmt(message:=" Status \1", v1 := netInfo.status);
   IF netInfo.status > 1 THEN
      DebugMsg(message:=" Phy addr " + netInfo.phy_addr);
      IF netInfo.dhcp THEN
         DebugMsg(message:=" Dynamic IP address");
      ELSE
         DebugMsg(message:=" Static IP address");
      END_IF;
      DebugMsg(message:=" IP addr " + sockIPToName(ip := netInfo.ip));
      DebugMsg(message:=" Mask    " + sockIPToName(ip := netInfo.subnetMask));
      DebugMsg(message:=" Gateway " + sockIPToName(ip := netInfo.gateway));
```

```
      IF netInfo.AutoDNS THEN
         DebugMsg(message:=" Automatic DNS");
      ELSE
         DebugMsg(message:=" Manual DNS");
      END_IF;
      DebugMsg(message:=" DNS1    " + sockIPToName(ip := netInfo.DNS1));
      DebugMsg(message:=" DNS2    " + sockIPToName(ip := netInfo.DNS2));
   END_IF;
END_FUNCTION;

PROGRAM test;
VAR
   rc : INT;
END_VAR;
   IF iface = 1 THEN
      // Turn on power to GSM module
      gsmPower(power:=ON);
   END_IF;
   IF NOT netPresent(iface:=iface) THEN
      DebugFmt(message:="Network interface \1 is not available", v1:=iface);
      WHILE TRUE DO
         Sleep(delay:=5000);
      END_WHILE;
   END_IF;
   rc := netOpen(iface:=iface);
   DebugFmt(message:="netOpen: \1", v1:=rc);
   WHILE NOT netConnected(iface:=iface) DO
      DebugMsg(message:="Waiting for network connection");
      Sleep(delay:=2500);
   END_WHILE;
   // Print network information
   printInfo();
   // Network is ready
BEGIN

   //...

END;
END_PROGRAM;
```

4.2.32.3 netClose (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 4.34 / 1.00.00

Closes a network interface.
When this function returns, no further communication is possible until the interface is re-opened using netOpen.

After this functions returns the netConnected() function will indicate than the connection to the network is unavailable.

Input:

iface : SINT
The network interface to close. 1 = Mobile network, 2 = LAN network, etc. (See Network)

Returns: INT

0 - Success.
-1 - Unknown network interface.

Declaration:

```
FUNCTION netClose : INT;
VAR_INPUT
   iface : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Open the network interface
netOpen(iface:=1);

BEGIN
  ...
  // Close the network interface
  netClose(iface:=1);
  ...
END;
END_PROGRAM;
```

4.2.32.4 netConnected (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 4.34 / 1.00.00

This function tells if a connection to the network has been established.

Input:

iface : SINT
The network interface to query. 0 = Any network, 1 = Mobile network, 2 = LAN network, etc. (See Network)

Returns: BOOL

true if connection to the network has been established, false if not.

Declaration:

```
FUNCTION netConnected : BOOL;
VAR_INPUT
   iface : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
//  These are the global variables of the program
VAR
   netInfo  :  netGetInformation;
   iface    :  SINT := 2; // Network interface to use: 1 for Mobile, 2 for
LAN.
END_VAR;

FUNCTION printInfo
```

```
   netInfo(iface:=iface);
   DebugFmt(message:="Network interface \1:", v1 := iface);
   DebugFmt(message:=" Status \1", v1 := netInfo.status);
   IF netInfo.status > 1 THEN
      DebugMsg(message:=" Phy addr " + netInfo.phy_addr);
      IF netInfo.dhcp THEN
         DebugMsg(message:=" Dynamic IP address");
      ELSE
         DebugMsg(message:=" Static IP address");
      END_IF;
      DebugMsg(message:=" IP addr " + sockIPToName(ip := netInfo.ip));
      DebugMsg(message:=" Mask    " + sockIPToName(ip := netInfo.subnetMask));
      DebugMsg(message:=" Gateway " + sockIPToName(ip := netInfo.gateway));
      IF netInfo.AutoDNS THEN
         DebugMsg(message:=" Automatic DNS");
      ELSE
         DebugMsg(message:=" Manual DNS");
      END_IF;
      DebugMsg(message:=" DNS1    " + sockIPToName(ip := netInfo.DNS1));
      DebugMsg(message:=" DNS2    " + sockIPToName(ip := netInfo.DNS2));
   END_IF;
END_FUNCTION;

PROGRAM test;
VAR
   rc : INT;
END_VAR;
   IF iface = 1 THEN
      // Turn on power to GSM module
      gsmPower(power:=ON);
   END_IF;
   IF NOT netPresent(iface:=iface) THEN
      DebugFmt(message:="Network interface \1 is not available", v1:=iface);
      WHILE TRUE DO
         Sleep(delay:=5000);
      END_WHILE;
   END_IF;
   rc := netOpen(iface:=iface);
   DebugFmt(message:="netOpen: \1", v1:=rc);
   WHILE NOT netConnected(iface:=iface) DO
      DebugMsg(message:="Waiting for network connection");
      Sleep(delay:=2500);
   END_WHILE;
   // Print network information
   printInfo();
   // Network is ready
BEGIN

   //...

END;
END_PROGRAM;
```

4.2.32.5 netPresent (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 4.34 / 1.00.00

Returns information about whether a network interface is present in the device.

Input:

iface : SINT
The network interface to query. 1 = Mobile network, 2 = LAN network, etc. (See Network)

Returns: BOOL
true if the network interface is present, false if not.

Declaration:
```
FUNCTION netPresent : BOOL;
VAR_INPUT
   iface : SINT;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc
//  These are the global variables of the program
VAR
   netInfo  :  netGetInformation;
   iface    :  SINT := 2; // Network interface to use: 1 for Mobile, 2 for
LAN.
END_VAR;

FUNCTION printInfo
   netInfo(iface:=iface);
   DebugFmt(message:="Network interface \1:", v1 := iface);
   DebugFmt(message:=" Status \1", v1 := netInfo.status);
   IF netInfo.status > 1 THEN
      DebugMsg(message:=" Phy addr " + netInfo.phy_addr);
      IF netInfo.dhcp THEN
         DebugMsg(message:=" Dynamic IP address");
      ELSE
         DebugMsg(message:=" Static IP address");
      END_IF;
      DebugMsg(message:=" IP addr " + sockIPToName(ip := netInfo.ip));
      DebugMsg(message:=" Mask    " + sockIPToName(ip := netInfo.subnetMask));
      DebugMsg(message:=" Gateway " + sockIPToName(ip := netInfo.gateway));
      IF netInfo.AutoDNS THEN
         DebugMsg(message:=" Automatic DNS");
      ELSE
         DebugMsg(message:=" Manual DNS");
      END_IF;
      DebugMsg(message:=" DNS1    " + sockIPToName(ip := netInfo.DNS1));
      DebugMsg(message:=" DNS2    " + sockIPToName(ip := netInfo.DNS2));
   END_IF;
END_FUNCTION;

PROGRAM test;
VAR
   rc : INT;
END_VAR;
   IF iface = 1 THEN
      // Turn on power to GSM module
      gsmPower(power:=ON);
   END_IF;
   IF NOT netPresent(iface:=iface) THEN
      DebugFmt(message:="Network interface \1 is not available", v1:=iface);
      WHILE TRUE DO
         Sleep(delay:=5000);
      END_WHILE;
   END_IF;
   rc := netOpen(iface:=iface);
```

```
   DebugFmt(message:="netOpen: \1", v1:=rc);
   WHILE NOT netConnected(iface:=iface) DO
      DebugMsg(message:="Waiting for network connection");
      Sleep(delay:=2500);
   END_WHILE;
   // Print network information
   printInfo();
   // Network is ready
BEGIN

   //...

END;
END_PROGRAM;
```

4.2.32.6 netGetInformation (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 4.34 / 1.00.00

This returns information about a network interface.
Depending on the kind of interface and the status, some variables may be empty.

The format of an IP address is the same as returned by the sockGetLocalIP function.

Input:

iface : SINT
The network interface to query. 1 = Mobile network, 2 = LAN network, etc. (See Network)

Output:

status : SINT;
0 = Interface is not present.
1 = Interface is not opened.
2 = Interface is not connected.
3 = Interface is connected.
4 = Interface link is up and waiting for IP configuration.

phy_addr : STRING
The physical address. For example the MAC address of the LAN interface.

IP : DINT
The local IP address.

SubnetMask : DINT
The subnet mask.

Gateway : DINT
The network gateway address.

DHCP : BOOL
Whether DHCP is used to get IP configuration.

dhcp_addr : DINT
The IP address of the DHCP server used.

AutoDNS : BOOL
If the DNS server addresses are automatically configured.

DNS1 : DINT
The IP address of the primary DNS server.

DNS2 : DINT
The IP address of the secondary DNS server.

SSID : STRING
The SSID of the wireless network.

index_ap : INT
The index of the used wireless network settings.

NAT_gw : BOOL
Does this interface act as a gateway to the internet?

NAT_fwd : BOOL
Is NAT forwarding activated for this interface?

DHCP_enable : BOOL
The DHCP server is enabled on this interface if true, otherwise not-

DHCP_lease : INT
THe lease time in hours of the DHCP addresses [1h..24h].

DHCP_begin : DINT
The low end of the DHCP server IP pool range.

DHCP_end : DINT
The high end of the DHCP server IP pool range.

DHCP_staticDNS : BOOL
IF true, the DHCP server configures the client with the following static DNS IP addresses. Otherwise the DNS server configuration from the gateway is used. See netSetLANParam and netGetLANParam.

DHCP_DNS1 : DINT
The first priority static DNS server IP address.

DHCP_DNS2 : DINT
The second priority static DNS server IP address.

Declaration:

```
FUNCTION_BLOCK netGetInformation;
VAR_INPUT
   iface          : MANDATORY SINT;
END_VAR;
VAR_OUTPUT
   status         : SINT;
   phy_addr       : STRING;
   IP             : DINT;
   SubnetMask     : DINT;
   Gateway        : DINT;
   DHCP           : BOOL;
   dhcp_addr      : DINT;
   AutoDNS        : BOOL;
   DNS1           : DINT;
   DNS2           : DINT;
```

```
   SSID           : STRING;
   index_ap       : INT;
   NAT_gw         : BOOL;
   NAT_fwd        : BOOL;
   DHCP_enable    : BOOL;
   DHCP_lease     : INT;
   DHCP_begin     : DINT;
   DHCP_end       : DINT;
   DHCP_staticDNS : BOOL;
   DHCP_DNS1      : DINT;
   DHCP_DNS2      : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
VAR
   netInfo  :  netGetInformation;
   iface    :  SINT := 2; // Network interface to use: 1 for Mobile, 2 for
LAN.
END_VAR;

FUNCTION printInfo
   netInfo(iface:=iface);
   DebugFmt(message:="Network interface \1:", v1 := iface);
   DebugFmt(message:=" Status \1", v1 := netInfo.status);
   IF netInfo.status > 1 THEN
      DebugMsg(message:=" Phy addr " + netInfo.phy_addr);
      IF netInfo.dhcp THEN
         DebugMsg(message:=" Dynamic IP address");
      ELSE
         DebugMsg(message:=" Static IP address");
      END_IF;
      DebugMsg(message:=" IP addr " + sockIPToName(ip := netInfo.ip));
      DebugMsg(message:=" Mask    " + sockIPToName(ip := netInfo.subnetMask));
      DebugMsg(message:=" Gateway " + sockIPToName(ip := netInfo.gateway));
      IF netInfo.AutoDNS THEN
         DebugMsg(message:=" Automatic DNS");
      ELSE
         DebugMsg(message:=" Manual DNS");
      END_IF;
      DebugMsg(message:=" DNS1    " + sockIPToName(ip := netInfo.DNS1));
      DebugMsg(message:=" DNS2    " + sockIPToName(ip := netInfo.DNS2));
      IF netInfo.NAT_gw THEN
         DebugMsg(message:=" This interface acts as gateway for NAT");
      ELSE
         DebugMsg(message:=" This interface is not a NAT gateway");
      END_IF;
      IF netInfo.NAT_fwd THEN
         DebugMsg(message:=" NAT forwarding is active");
      ELSE
         DebugMsg(message:=" NAT forwarding is not active");
      END_IF;
      IF netInfo.DHCP_enable THEN
         DebugMsg(message:=" DHCP server is active");
         DebugFmt(message:=" DHCP server IP address lease time is \1 hours",
v1 := netInfo.DHCP_lease);
         DebugMsg(message:=" DHCP IP pool range start    " + sockIPToName(ip
:= netInfo.DHCP_begin));
         DebugMsg(message:=" DHCP IP pool range end      " + sockIPToName(ip
:= netInfo.DHCP_end));
         IF netInfo.DHCP_staticDNS THEN
            DebugMsg(message:=" DHCP server uses following static IP addresses
to configure the client:");
```

```
            DebugMsg(message:=" DHCP DNS1    " + sockIPToName(ip :=
netInfo.DHCP_DNS1));
            DebugMsg(message:=" DHCP DNS2    " + sockIPToName(ip :=
netInfo.DHCP_DNS2));
         ELSE
            DebugMsg(message:=" DHCP server DNS configuration is from the
gateway.");
         END_IF;
      ELSE
         DebugMsg(message:=" DHCP server is not active");
      END_IF;
   END_IF;
END_FUNCTION;

PROGRAM test;
VAR
   rc : INT;
END_VAR;
   IF iface = 1 THEN
      // Turn on power to GSM module
      gsmPower(power:=ON);
   END_IF;
   IF NOT netPresent(iface:=iface) THEN
      DebugFmt(message:="Network interface \1 is not available", v1:=iface);
      WHILE TRUE DO
         Sleep(delay:=5000);
      END_WHILE;
   END_IF;
   rc := netOpen(iface:=iface);
   DebugFmt(message:="netOpen: \1", v1:=rc);
   WHILE NOT netConnected(iface:=iface) DO
      DebugMsg(message:="Waiting for network connection");
      Sleep(delay:=2500);
   END_WHILE;
   // Print network information
   printInfo();
   // Network is ready
BEGIN

   //...

END;
END_PROGRAM;
```

4.2.32.7 netGetStatistics (Functionblock)

Architecture: NX32L
Device support: All
Firmware version: 2.27.00

This returns the current statistics for a network interface.
Depending on the kind of interface and the status, some variables may be empty.
The variables will wrap around when reaching the max value(2147483647). This means that fast-growing values such as rx_bytes will turn negative when the max value is reached, which the application must handle.

Input:

iface : SINT
The network interface to query. 1 = Mobile network, 2 = LAN network, etc. (See Network)

Output:

status : SINT;
0 = Not valid statistics found for this interface. The other fields have not been updated.
1 = The statistics are valid.

rx_packets : DINT
The number of good packets received by the interface.

tx_packets : DINT
The number of successfully sent packets.

rx_bytes : DINT
The number of good received bytes.

tx_bytes : DINT
The number of good transmitted bytes.

rx_dropped : DINT
The number of packets received but not processed.

tx_dropped : DINT
The number of packets dropped on their way to transmission.

rx_compressed : DINT
The number of received compressed frames(Cellular only)

tx_compressed : DINT
The number of transmitted compressed frames(Cellular only)

rx_errors : DINT
The total number of bad packets received on this network.
Sum of the following errors:

rx_length_errors : DINT
The number of packets dropped due to invalid length.

rx_over_errors : DINT
The number of packets dropped due to overflow.

rx_crc_errors : DINT
The number of packets received with a CRC error.

rx_frame_errors : DINT
Receiver frame alignment errors.

rx_fifo_errors : DINT
Receive fifo error count.

rx_missed_errors : DINT
The number of packets missed by the host.

tx_errors : DINT
The total number of transmit problems.
Sum of the following errors:

tx_aborted_errors : DINT
The number of discarded frames.

tx_carrier_errors : DINT

The number of frame transmission errors due to loss of carrier.

tx_fifo_errors : DINT
The number of frame transmission errors due to FIFO underrun/underflow.

tx_heartbeat_errors : DINT
The number of heartbeat errors.

tx_window_errors : DINT
The number of frame transmission errors due to late collisions.

multicast : DINT
The number of multicast packets received.

collisions : DINT
The number of collisions during packet transmissions.

Declaration:

```
FUNCTION_BLOCK netGetStatistics;
VAR_INPUT
   iface              : SINT;
END_VAR;
VAR_OUTPUT
   status             : SINT;
   rx_packets         : DINT;
   tx_packets         : DINT;
   rx_bytes           : DINT;
   tx_bytes           : DINT;
   rx_errors          : DINT;
   tx_errors          : DINT;
   rx_dropped         : DINT;
   tx_dropped         : DINT;
   multicast          : DINT;
   collisions         : DINT;
   rx_length_errors   : DINT;
   rx_over_errors     : DINT;
   rx_crc_errors      : DINT;
   rx_frame_errors    : DINT;
   rx_fifo_errors     : DINT;
   rx_missed_errors   : DINT;
   tx_aborted_errors  : DINT;
   tx_carrier_errors  : DINT;
   tx_fifo_errors     : DINT;
   tx_heartbeat_errors: DINT;
   tx_window_errors   : DINT;
   rx_compressed      : DINT;
   tx_compressed      : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
VAR
   netStats :  netGetStatistics;
   iface    :  SINT := 2; // Network interface to use: 1 for Mobile, 2 for
LAN.
END_VAR;

FUNCTION printStats
   netStats(iface:=iface);
```

```
   DebugFmt(message:="Network interface \1:", v1 := iface);
   DebugFmt(message:=" Status \1", v1 := netStats.status);
   IF netStats.status > 1 THEN
         DebugFmt(message:=" RX packets: \4", v4:=netStats.rx_packets);
         DebugFmt(message:=" TX packets: \4", v4:=netStats.tx_packets);
         DebugFmt(message:=" RX bytes:   \4", v4:=netStats.rx_bytes);
         DebugFmt(message:=" TX bytes:   \4", v4:=netStats.tx_bytes);
         DebugFmt(message:=" RX errors:  \4", v4:=netStats.rx_errors);
         DebugFmt(message:=" TX errors:  \4", v4:=netStats.tx_errors);
         DebugFmt(message:=" RX dropped: \4", v4:=netStats.rx_dropped);
         DebugFmt(message:=" TX dropped: \4", v4:=netStats.tx_dropped);
   END_IF;
END_FUNCTION;

PROGRAM test;
VAR
   rc : INT;
END_VAR;
   IF iface = 1 THEN
      // Turn on power to GSM module
      gsmPower(power:=ON);
   END_IF;
   IF NOT netPresent(iface:=iface) THEN
      DebugFmt(message:="Network interface \1 is not available", v1:=iface);
      WHILE TRUE DO
         Sleep(delay:=5000);
      END_WHILE;
   END_IF;
   rc := netOpen(iface:=iface);
   DebugFmt(message:="netOpen: \1", v1:=rc);
   WHILE NOT netConnected(iface:=iface) DO
      DebugMsg(message:="Waiting for network connection");
      Sleep(delay:=2500);
   END_WHILE;
   // Print network statistics
   printStats();
   // Network is ready
BEGIN

   //...

END;
END_PROGRAM;
```

4.2.32.8 netPing (Functionblock)

Architecture: NX32L
Device support: ALL
Firmware version: 1.52.00

This function sends ICMP-echo (ping) packets to a designated host IP interface and reports statistics on the reply.
For situation that do not require the detailed statistics, the simpler netPingS function can be used instead.

Input:

iface : SINT (*default 2*)
The network interface. 1 = Mobile network, 2 = LAN network, etc. (See Network)

IP : DINT
The IP address of the interface on the host to check if it answers.

Count : UINT (default 5)
How many ping packets to send.

Reply_timeout : UINT (default 1000)
The time to wait for a reply from the host in milliseconds.

Output:

Status : INT

1	- Success.
0	- Function is not supported.
-1	- General error.
-2	- Invalid input
-3	- Interface is not connected. Use netOpen.

Packets : INT
Number of ping packets received from the request.

Loss : INT
Number of ping packets with no reply.

TimeMin : FLOAT
Minimum round trip time in milliseconds.

TimeMax : FLOAT
Maximum round trip time in milliseconds.

TimeAv : FLOAT
Average round trip time in milliseconds.

Declaration:

```
FUNCTION_BLOCK netPing;
VAR_INPUT
   iface          : SINT := 2;
   IP             : DINT;
   Count          : UINT := 5;
   Reply_timeout  : UINT := 1000;
END_VAR;
VAR_OUTPUT
   Status         : INT;
   Packets        : INT;
   Loss           : INT;
   TimeMin        : FLOAT;
   TimeMax        : FLOAT;
   TimeAv         : FLOAT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   ping     : netPing;
   netInfo  : netGetInformation;
   iface    : SINT := 2;
END_VAR;

FUNCTION printInfo
  netInfo(iface:=iface);
```

```
  DebugFmt(message:="Network interface \1:", v1 := iface);
  CASE netInfo.status OF
     0 : DebugMsg(message:="Interface is not present.");
     1 : DebugMsg(message:="Interface is not opened.");
     2 : DebugMsg(message:="Interface is not connected.");
     3 : DebugMsg(message:="Interface is connected.");
     4 : DebugMsg(message:="Interface link is up and waiting for IP
configuration.");
  ELSE
     DebugFmt(message:="Illegal net status? (\1)", v1:=netInfo.status);
  END_CASE;
  IF netInfo.status > 1 THEN
    DebugMsg(message:=" Phy addr " + netInfo.phy_addr);
    IF netInfo.dhcp THEN
        DebugMsg(message:=" Dynamic IP address");
    ELSE
        DebugMsg(message:=" Static IP address");
    END_IF;
    DebugMsg(message:=" IP addr " + sockIPToName(ip := netInfo.ip));
    DebugMsg(message:=" Mask    " + sockIPToName(ip := netInfo.subnetMask));
    DebugMsg(message:=" Gateway " + sockIPToName(ip := netInfo.gateway));
    IF netInfo.AutoDNS THEN
        DebugMsg(message:=" Automatic DNS");
    ELSE
        DebugMsg(message:=" Manual DNS");
    END_IF;
    DebugMsg(message:=" DNS1    " + sockIPToName(ip := netInfo.DNS1));
    DebugMsg(message:=" DNS2    " + sockIPToName(ip := netInfo.DNS2));
  END_IF;
END_FUNCTION;

FUNCTION PrintPing;
   DebugMsg(message:="---------------input-----------------------------");
   DebugFmt(message:="Network interface \1:", v1 := ping.iface);
   DebugMsg(message:="IP address is "+soAddrFromIP(ip:=ping.ip));
   DebugFmt(message:="Send \1 packets.", v1:=ping.count);
   DebugFmt(message:="Send for time: \1 seconds.", v1:=ping.timeout);
   DebugMsg(message:="---------------output----------------------------");
   IF ping.status > 0 THEN
      DebugFmt(message:="Received \1 packets.", v1:=ping.packets);
      DebugFmt(message:="Loss ratio was: \1", v1:=ping.loss);
      DebugMsg(message:="Minimum round time was "+floatToStr(v:=ping.timemin)
+"ms.");
      DebugMsg(message:="Maximum round time was "+floatToStr(v:=ping.timemax)
+"ms.");
      DebugMsg(message:="Average round time was "+floatToStr(v:=ping.timeav)
+"ms.");
   ELSE
      DebugFmt(message:="Send status is \1", v1:=ping.status);
   END_IF;
   DebugMsg(message:="-------------------------------------------------");
END_FUNCTION;

PROGRAM pingtest;
VAR
   rc : INT;
END_VAR;
rc := netOpen(iface:=iface);
DebugFmt(message:="netOpen (rc=\1)",v1:=rc);
WHILE NOT netConnected(iface:=iface) DO
   Sleep(delay:=2000);
END_WHILE;
printinfo();
```

```
// Send 10 ping packets to host
ping(iface:=iface,
     ip:=sockIPFromName(str:="gw.rtcu.dk"),
     count:=10,
     reply_timeout:=500);

// Print ping input settings and output statistics
printPing();
END_PROGRAM;
```

4.2.32.9 netPingS (Function)

Architecture: NX32 / NX32L
Device support: All
Firmware version: 5.12 / 1.62.00

This function sends a ICMP-echo (ping) packet to a designated host IP address and returns the reply time.
To get more detailed statistics on NX32L devices, the alternative function netPing can be used.

On NX32 devices this function only supports iface=1 (Mobile network).

Input:

IP : DINT
The IP address of the host.

iface : SINT
The network interface to use. 1 = Mobile network, 2 = LAN network, etc. (See Network)

timeout : UINT (default 500)
The time to wait for a reply from the host in milliseconds.

Returns: INT

1	- Success.
0	- Not available.
-1	- Timeout.
-2	- Invalid parameter.
-3	- The network interface is not available.
-4	- Communication error.

Declaration:

```
FUNCTION netPingS : INT;
VAR_INPUT
   ip             : DINT;
   iface          : SINT := 1;
   timeout        : UINT := 500;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   iface    : SINT := 1;
END_VAR;

PROGRAM pingtest;
VAR
   rc : INT;
```

```
END_VAR;

// Network
rc := netOpen(iface:=iface);
DebugFmt(message:="netOpen (rc=\1)",v1:=rc);
WHILE NOT netConnected(iface:=iface) DO
   Sleep(delay:=2000);
END_WHILE;

// Ping host
rc := netPingS(iface:=iface, ip:=sockIPFromName(str:="gw.rtcu.dk"),
timeout:=1000);
DebugFmt(message := "netPingS. reply time=\1)", v1 := rc);
END_PROGRAM;
```

4.2.32.10 netGetSignalLevel (Function)

Architecture: NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

---

netGetSignalLevel will return the signal strength for a connected network interface.
The returned value will either be the real signal strength (expressed in dBm) or as a percentage (between 0 and 100), depending of the value of the "dbm" input parameter.
Please note that the signal level is logarithmic in nature and that the value in percentage is just a linear representation of the real signal level from the lowest to the highest level. 10 dB (which is 10 times the RF power) of extra signal will only give an extra percentage reading of 16.

The range of the real signal level depends on the type of interface:

Interface type	Lowest	Highest
Mobile	-113 dBm	-51 dBm
WLAN	-100 dBm	-40 dBm
LAN (Simulated values)	-100 dBm	-40 dBm

Input:

iface : SINT
The network interface to read configuration for. 1 = Mobile network, 2 = LAN 1 network, etc. (See Network).
Note that the values for the LAN networks are simulated and are only included for completeness.

dbm : BOOL *(default FALSE)*
Set to `true` to get the level as the real signal strength in dBm, set to `false` to get it as a percentage.

Output:

ssid : STRING
The SSID of the network, if the network is WLAN.

level : SINT
The signal strength.

Returns: INT

- 1 - Success.
- 0 - Interface not available or not supported.
- -1 - Invalid interface.
- -2 - The network is not open.
- -3 - Error

Declaration:

```
FUNCTION netGetSignalLevel : INT;
VAR_INPUT
   iface          : SINT;
   dbm            : BOOL := FALSE;
   ssid           : ACCESS STRING;
   level          : ACCESS SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   level : SINT;
   ssid  : STRING;
END_VAR;

BEGIN
   ...
   // Get signal level in % (from 0 to 100)
   netGetSignalLevel(iface := 4, dbm := FALSE, ssid := ssid, level := level);
   ...
END;

END_PROGRAM;
```

4.2.32.11 netSetLANParam (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 4.34 / 1.00.00

Sets TCP/IP-specific parameters for establishing connections over the LAN network interface. It has the same effect as setting the parameters via Device: Network: Network Settings.

The LAN network interface must be closed and opened again before the changes can take effect if already opened (see also netClose / netOpen).

The format of an IP address is the same as returned by the sockGetLocalIP function.

Input:

iface : SINT ***(default 2)***
The network interface to configure. 2 = LAN 1 network, 3 = LAN 2 network, etc. (See Network)
Please note that the iface parameter is only supported on NX32L.

DHCP : BOOL ***(default TRUE)***
Use DHCP to obtain TCP/IP parameters automatically.

IP : DINT ***(default 0)***
The IP address of the device.

SubnetMask : DINT ***(default 0)***
The Subnet mask of the device.

Gateway : DINT ***(default 0)***
The TCP/IP gateway the device must use.

AutoDNS : BOOL ***(default TRUE)***
Obtain the DNS parameters automatically. Can only be used in combination with DHCP.

DNS1 : DINT ***(default 0)***
The first Domain Name Server the device should use to look up symbolic names.

DNS2 : DINT ***(default 0)***
The second Domain Name Server the device should use to look up symbolic names.

Returns: INT

0 - Success.
2 - AutoDNS enabled without DHCP.

Declaration:

```
FUNCTION netSetLANParam : INT;
VAR_INPUT
   iface          : SINT := 2;
   DHCP           : BOOL := TRUE;
   IP             : DINT := 0;
   SubnetMask     : DINT := 0;
   Gateway        : DINT := 0;
   AutoDNS        : BOOL := TRUE;
   DNS1           : DINT := 0;
   DNS2           : DINT := 0;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   LANParam     : netGetLANParam;
END_VAR;

LANParam();

IF LANParam.DHCP = FALSE OR LANParam.AutoDNS = FALSE THEN
   // Turn on DHCP and Automatic DNS
   netSetLANParam(DHCP := TRUE, AutoDNS := TRUE);
END_IF;

netOpen(iface := 2);
BEGIN

  // ...

END;

END_PROGRAM;
```

4.2.32.12 netGetLANParam (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 4.34 / 1.00.00

---

netGetLANParam will read out the TCP/IP parameters previously set from the RTCU IDE or by the netSetLANParam function.

The format of an IP address is the same as returned by the sockGetLocalIP function.

Input:

iface : SINT *(default 2)*
The network interface to read configuration for. 2 = LAN 1 network, 3 = LAN 2 network, etc. (See Network)
Please note that the iface parameter is only supported on NX32L.

Output:

DHCP : BOOL
Use DHCP to obtain TCP/IP parameters automatically.

IP : DINT
The IP address of the device.

SubnetMask : DINT
The Subnet mask of the device.

Gateway : DINT
The TCP/IP gateway the device uses.

AutoDNS : BOOL
Obtain the DNS parameters automatically.

DNS1 : DINT
The first Domain Name Server the device uses to look up symbolic names.

DNS2 : DINT
The second Domain Name Server the device uses to look up symbolic names.

Declaration:

```
FUNCTION_BLOCK netGetLANParam;
VAR_INPUT
   iface          : SINT := 2;
END_VAR;
VAR_OUTPUT
   DHCP           : BOOL;
   IP             : DINT;
   SubnetMask     : DINT;
   Gateway        : DINT;
   AutoDNS        : BOOL;
   DNS1           : DINT;
   DNS2           : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   LANParam     : netGetLANParam;
END_VAR;

LANParam();

IF LANParam.DHCP = FALSE OR LANParam.AutoDNS = FALSE THEN
   // Turn on DHCP and Automatic DNS
   netSetLANParam(DHCP := TRUE, AutoDNS := TRUE);
END_IF;

netOpen(iface := 2);
BEGIN

  // ...

END;

END_PROGRAM;
```

4.2.32.13 netSetMobileParam (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 4.34 / 1.00.00

Sets TCP/IP-specific parameters for establishing connections over the mobile network interface. It has the same effect as setting the parameters via Device: Network: Network Settings.

The mobile network interface must be closed and opened again before the changes can take effect if already opened (see also netClose / netOpen).

The format of an IP address is the same as returned by the sockGetLocalIP function.

Input:

IP : DINT *(default 0)*
The IP address of the device (normally set by negotiation).

SubnetMask : DINT *(default 0)*
The Subnet mask of the device (normally set by negotiation).

Gateway : DINT *(default 0)*
The TCP/IP gateway the device must use (normally set by negotiation).

DNS1 : DINT *(default 0)*
The first Domain Name Server the device should use to look up symbolic names (normally set by negotiation).

DNS2 : DINT *(default 0)*
The second Domain Name Server the device should use to look up symbolic names (normally set by negotiation).

Authenticate : INT *(Default 3)*
Authentication type:
0 = None.

1 = PAP
2 = CHAP
3 = PAP/CHAP

Username : STRING ***(Max 33 characters)***
The user name the device should use in order to connect to the mobile network.

Password : STRING ***(Max 33 characters)***
The password the device should use in order to connect to the mobile network.

APN : STRING ***(Max 33 characters)***
The APN the device should use in order to connect to the mobile network.

Returns: INT

0 - Success.
2 - String too long.

Declaration:

```
FUNCTION netSetMobileParam : INT;
VAR_INPUT
   IP             : DINT := 0;
   SubnetMask     : DINT := 0;
   Gateway        : DINT := 0;
   DNS1           : DINT := 0;
   DNS2           : DINT := 0;
   Username       : STRING;
   Password       : STRING;
   APN            : STRING;
   Authenticate   : INT  := 3;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   mobileParam  : netGetMobileParam;
END_VAR;

mobileParam();
IF NOT (mobileParam.IP = 0 AND mobileParam.SubnetMask = 0 AND
mobileParam.Gateway = 0 AND
   mobileParam.DNS1 = 0 AND mobileParam.DNS2 = 0 AND mobileParam.Authenticate
= 3 AND
   mobileParam.Username = "" AND mobileParam.Password = "" AND mobileParam.APN
= "internet") THEN
   // Set APN and keep the default values for the others
   netSetMobileParam(APN := "internet");
END_IF;

// Turn on power to the GSM module
gsmPower(power := ON);
netOpen(iface := 1);
BEGIN

  // ...

END;

END_PROGRAM;
```

4.2.32.14 netGetMobileParam (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 4.34 / 1.00.00

---

netGetMobileParam will read out the TCP/IP parameters previously set from the RTCU IDE or by the netSetMobileParam or sockSetTCPIPParm functions.

The format of an IP address is the same as returned by the sockGetLocalIP function.

Input:

None.

Output:

IP : DINT
The IP address of the device.

SubnetMask : DINT
The Subnet mask of the device.

Gateway : DINT
The TCP/IP gateway the device uses.

DNS1 : DINT
The first Domain Name Server the device uses to look up symbolic names.

DNS2 : DINT
The second Domain Name Server the device uses to look up symbolic names.

Authenticate : INT
The PPP authentication type:
0 = None.
1 = PAP.
2 = CHAP.
3 = PAP/CHAP.

Username : STRING
The user name the device uses in order to connect to the mobile network.

Password : STRING
The password the device uses in order to connect to the mobile network.

APN : STRING
The APN the device uses in order to connect to the mobile network.

Declaration:

```
FUNCTION_BLOCK netGetMobileParam;
VAR_OUTPUT
   IP             : DINT;
   SubnetMask     : DINT;
   Gateway        : DINT;
   DNS1           : DINT;
   DNS2           : DINT;
   Username       : STRING;
   Password       : STRING;
```

```
   APN            : STRING;
   Authenticate   : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   mobileParam  : netGetMobileParam;
END_VAR;

mobileParam();
IF NOT (mobileParam.IP = 0 AND mobileParam.SubnetMask = 0 AND
mobileParam.Gateway = 0 AND
   mobileParam.DNS1 = 0 AND mobileParam.DNS2 = 0 AND mobileParam.Authenticate
= 3 AND
   mobileParam.Username = "" AND mobileParam.Password = "" AND mobileParam.APN
= "internet") THEN
   // Set APN and keep the default values for the others
   netSetMobileParam(APN := "internet");
END_IF;

// Turn on power to the GSM module
gsmPower(power := ON);
netOpen(iface := 1);
BEGIN

  // ...

END;

END_PROGRAM;
```

4.2.32.15 netSetWLANParam (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900, LX5
Firmware version: 1.00.00

Sets the parameters for establishing connections to a wireless network with the WLAN network interface. It has the same effect as setting the parameters via Device: Network: Network Settings. In cases where multiple wireless networks are present, the network with the lowest index is connected to.

The WLAN network interface must be closed and opened again before the changes can take effect if already opened (see also netClose / netOpen).
The format of an IP address is the same as returned by the sockGetLocalIP function.

Input:

index : INT (1..10)
The index of the wireless network.

SSID : STRING
The SSID of the wireless network.

sec_config : PTR
The security configuration for the wireless network.

The security is configured by way of a pointer to a struct block which contains the parameters.

0	No security	None
1	WPA/WPA2 PSK	netWLANSecurityWPA

DHCP : BOOL *(default TRUE)*
Use DHCP to obtain TCP/IP parameters automatically.

IP : DINT
The IP address of the device.

SubnetMask : DINT
The Subnet mask of the device.

Gateway : DINT
The TCP/IP gateway the device must use.

AutoDNS : BOOL *(default TRUE)*
Obtain the DNS parameters automatically. Can only be used in combination with DHCP.

DNS1 : DINT
The first Domain Name Server the device should use to look up symbolic names.

DNS2 : DINT
The second Domain Name Server the device should use to look up symbolic names.

Returns: INT

0 - Success.
1 - Illegal wireless network index
2 - Illegal SSID
3 - Unknown security configuration
4 - Illegal security configuration
5 - AutoDNS enabled without DHCP.

Declaration:

```
FUNCTION netSetWLANParam : INT;
VAR_INPUT
   index          : INT;
   ssid           : STRING;
   sec_config     : PTR;
   DHCP           : BOOL := TRUE;
   IP             : DINT;
   SubnetMask     : DINT;
   Gateway        : DINT;
   AutoDNS        : BOOL := TRUE;
   DNS1           : DINT;
   DNS2           : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   wpa_psk  : netWLANSecurityWPA;
   wlan     : netGetWLANParam;
END_VAR;

FUNCTION booltostr : STRING;
```

```
VAR_INPUT
   v : BOOL;
END_VAR;
IF v THEN booltostr := "Yes"; ELSE booltostr := "No"; END_IF;
END_FUNCTION;

FUNCTION wlan_set;
VAR
   rc       : INT;
END_VAR;

   // Setup security
   wpa_psk.phrase  := "passphrase";

   // Setup WLAN network
   rc := netSetWLANParam(
                         index       := 1
                         ,ssid       := "SSID"
                         ,sec_config := ADDR(wpa_psk)
                         ,DHCP       := ON
                         ,AutoDNS    := ON
                        );
   DebugFmt(message := "netSetWLANParam=    \1", v1 := rc);
END_FUNCTION;

FUNCTION wlan_show;
VAR_INPUT
   index : INT := 1;
END_VAR;

   wlan(index := index);
   DebugMsg(message := " SSID=         " + wlan.ssid);
   IF wlan.security = 0 THEN
      DebugMsg(message := " Security=     None");
   ELSIF wlan.security = 1 THEN
      DebugFmt(message := " Security=     WPA/WPA2 PSK");
      netGetWLANSecurity(index := index, sec_config := ADDR(wpa_psk));
      DebugMsg(message := "   phrase=         " + wpa_psk.phrase);
   ELSE
      DebugFmt(message := " Security=     Unknown (\1)", v1 := wlan.security);
   END_IF;
   DebugMsg(message := " Network=      IPv4");
   DebugMsg(message := "   IP=             " + sockIPToName(ip := wlan.ip));
   DebugMsg(message := "   Subnet=         " + sockIPToName(ip :=
wlan.SubnetMask));
   DebugMsg(message := "   Gw=             " + sockIPToName(ip :=
wlan.Gateway));
   DebugMsg(message := "   DHCP=           " + booltostr(v := wlan.dhcp));
   DebugMsg(message := "   Auto DNS=       " + booltostr(v := wlan.AutoDNS));
   DebugMsg(message := "   DNS=            " + sockIPToName(ip := wlan.DNS1));
   DebugMsg(message := "   DNS=            " + sockIPToName(ip := wlan.DNS2));
END_FUNCTION;

PROGRAM test;

wlan_set();
wlan_show(index := 1);

BEGIN
  // ...
END;
END_PROGRAM;
```

4.2.32.16 netGetWLANParam (Functionblock)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900, LX5
Firmware version: 1.00.00

---

netGetWLANParam will read out the parameters for a wireless network previously set from the RTCU IDE or by the netSetWLANParam function.

The format of an IP address is the same as returned by the sockGetLocalIP function.

Input:

index : INT (1..10)
The index of the wireless network.

Output:

SSID : STRING
The SSID of the wireless network.

security : INT
The type of security used by the wireless network. Use the netGetWLANSecurity function to get the security configuration.

0	None	None
1	WPA/WPA2 PSK	netWLANSecurityWPA

DHCP : BOOL
Use DHCP to obtain TCP/IP parameters automatically.

IP : DINT
The IP address of the device.

SubnetMask : DINT
The Subnet mask of the device.

Gateway : DINT
The TCP/IP gateway the device uses.

AutoDNS : BOOL
Obtain the DNS parameters automatically.

DNS1 : DINT
The first Domain Name Server the device uses to look up symbolic names.

DNS2 : DINT
The second Domain Name Server the device uses to look up symbolic names.

Declaration:

```
FUNCTION_BLOCK netGetWLANParam;
VAR_INPUT
   index          : INT;
END_VAR;
VAR_OUTPUT
   DHCP           : BOOL;
```

```
   IP             : DINT;
   SubnetMask     : DINT;
   Gateway        : DINT;
   AutoDNS        : BOOL;
   DNS1           : DINT;
   DNS2           : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   wpa_psk  : netWLANSecurityWPA;
   wlan     : netGetWLANParam;
END_VAR;

FUNCTION booltostr : STRING;
VAR_INPUT
   v : BOOL;
END_VAR;
IF v THEN booltostr := "Yes"; ELSE booltostr := "No"; END_IF;
END_FUNCTION;

FUNCTION wlan_set;
VAR
   rc       : INT;
END_VAR;

   // Setup security
   wpa_psk.phrase  := "passphrase";

   // Setup WLAN network
   rc := netSetWLANParam(
                         index       := 1
                         ,ssid       := "SSID"
                         ,sec_config := ADDR(wpa_psk)
                         ,DHCP       := ON
                         ,AutoDNS    := ON
                        );
   DebugFmt(message := "netSetWLANParam=    \1", v1 := rc);
END_FUNCTION;

FUNCTION wlan_show;
VAR_INPUT
   index : INT := 1;
END_VAR;

   wlan(index := index);
   DebugMsg(message := " SSID=         " + wlan.ssid);
   IF wlan.security = 0 THEN
      DebugMsg(message := " Security=     None");
   ELSIF wlan.security = 1 THEN
      DebugFmt(message := " Security=     WPA/WPA2 PSK");
      netGetWLANSecurity(index := index, sec_config := ADDR(wpa_psk));
      DebugMsg(message := "   phrase=         " + wpa_psk.phrase);
   ELSE
      DebugFmt(message := " Security=     Unknown (\1)", v1 := wlan.security);
   END_IF;
   DebugMsg(message := " Network=      IPv4");
   DebugMsg(message := "   IP=             " + sockIPToName(ip := wlan.ip));
   DebugMsg(message := "   Subnet=         " + sockIPToName(ip :=
wlan.SubnetMask));
   DebugMsg(message := "   Gw=             " + sockIPToName(ip :=
```

```
wlan.Gateway));
   DebugMsg(message := "   DHCP=           " + booltostr(v := wlan.dhcp));
   DebugMsg(message := "   Auto DNS=       " + booltostr(v := wlan.AutoDNS));
   DebugMsg(message := "   DNS=            " + sockIPToName(ip := wlan.DNS1));
   DebugMsg(message := "   DNS=            " + sockIPToName(ip := wlan.DNS2));
END_FUNCTION;

PROGRAM test;

wlan_set();
wlan_show(index := 1);

BEGIN
  // ...
END;
END_PROGRAM;
```

4.2.32.17 netGetWLANSecurity (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900, LX5
Firmware version: 1.00.00

---

netGetWLANSecurity will read out the security parameters for a wireless network previously set from the RTCU IDE or by the netSetWLANParam function.

The function is usually called after calling netGetWLANParam function block.

Input:

index : INT (1..10)
The index of the wireless network.

Output:

sec_config : PTR
The security configuration for the wireless network. The type of which is read by the netGetWLANParam function.

Returns: INT

0 - Success.
1 - Illegal wireless network index
2 - Not a valid security structure
3 - Unknown security configuration / Not matching requested security
4 - Missing security structure

Declaration:

```
FUNCTION netGetWLANSecurity : INT;
VAR_INPUT
   index          : INT;
   sec_config     : PTR;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
```

```
VAR
   wpa_psk  : netWLANSecurityWPA;
   wlan     : netGetWLANParam;
END_VAR;

FUNCTION booltostr : STRING;
VAR_INPUT
   v : BOOL;
END_VAR;
IF v THEN booltostr := "Yes"; ELSE booltostr := "No"; END_IF;
END_FUNCTION;

FUNCTION wlan_set;
VAR
   rc       : INT;
END_VAR;

   // Setup security
   wpa_psk.phrase  := "passphrase";

   // Setup WLAN network
   rc := netSetWLANParam(
                         index       := 1
                         ,ssid       := "SSID"
                         ,sec_config := ADDR(wpa_psk)
                         ,DHCP       := ON
                         ,AutoDNS    := ON
                        );
   DebugFmt(message := "netSetWLANParam=    \1", v1 := rc);
END_FUNCTION;

FUNCTION wlan_show;
VAR_INPUT
   index : INT := 1;
END_VAR;

   wlan(index := index);
   DebugMsg(message := " SSID=         " + wlan.ssid);
   IF wlan.security = 0 THEN
      DebugMsg(message := " Security=     None");
   ELSIF wlan.security = 1 THEN
      DebugFmt(message := " Security=     WPA/WPA2 PSK");
      netGetWLANSecurity(index := index, sec_config := ADDR(wpa_psk));
      DebugMsg(message := "   phrase=         " + wpa_psk.phrase);
   ELSE
      DebugFmt(message := " Security=     Unknown (\1)", v1 := wlan.security);
   END_IF;
   DebugMsg(message := " Network=      IPv4");
   DebugMsg(message := "   IP=             " + sockIPToName(ip := wlan.ip));
   DebugMsg(message := "   Subnet=         " + sockIPToName(ip :=
wlan.SubnetMask));
   DebugMsg(message := "   Gw=             " + sockIPToName(ip :=
wlan.Gateway));
   DebugMsg(message := "   DHCP=           " + booltostr(v := wlan.dhcp));
   DebugMsg(message := "   Auto DNS=       " + booltostr(v := wlan.AutoDNS));
   DebugMsg(message := "   DNS=            " + sockIPToName(ip := wlan.DNS1));
   DebugMsg(message := "   DNS=            " + sockIPToName(ip := wlan.DNS2));
END_FUNCTION;

PROGRAM test;

wlan_set();
wlan_show(index := 1);
```

```
BEGIN
  // ...
END;
END_PROGRAM;
```

4.2.32.18 netSetAPParam (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900, LX5
Firmware version: 1.50.00

Sets the parameters for a wireless network with the AP network interface. It has the same effect as setting the parameters via Device: Network: Network Settings.

The AP network interface must be closed and opened again before the changes can take effect if already opened (see also netClose / netOpen).
The format of an IP address is the same as returned by the sockGetLocalIP function.

Input:

SSID : STRING
The SSID of the wireless network.

Country : STRING
The country code where the device operates.
The ISO 3166-1 standard is used for the country code, as f.ex. 'DK' for Denmark.

Hidden : BOOL
Control whether the SSID will be broadcast.

Chnl : SINT *(1..14)*
The channel to use.
Please note:

- That not all 14 channels is available in every country. Channel 1 to 11 is always available, but it should be checked if channel 12, 13 or 14 is available.
- That if both the WLAN and AP network interfaces are open, then the channel can be changed when a connection to a WLAN network is established.

Sec_config : PTR
The security configuration for the wireless network (passwords must be of at least 8 characters length).
The security is configured by way of a pointer to a struct block which contains the parameters.

0	No security	None
1	WPA/WPA2 PSK	netWLANSecurityWPA

Address : DINT
The IP address of the device.

Subnet : DINT
The Subnet mask of the device.

Returns: INT

0 - Success.
2 - Illegal SSID
3 - Unknown security configuration
4 - Illegal security configuration

6 - Illegal parameter.

Declaration:

```
FUNCTION netSetAPParam : INT;
VAR_INPUT
   SSID           : STRING;
   Country        : MANDATORY STRING;
   Hidden         : BOOL;
   Chnl           : SINT;
   Sec_config     : PTR;
   Address        : DINT;
   Subnet         : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   wpa_psk  : netWLANSecurityWPA;
   net      : netGetAPParam;
END_VAR;

FUNCTION ap_set;
VAR
   rc       : INT;
END_VAR;

   // Setup security
   wpa_psk.phrase  := "passphrase";

   // Setup WLAN network
   rc := netSetAPParam(
                       ssid        := "SSID"
                       ,hidden     := FALSE
                       ,chnl       := 1
                       ,sec_config := ADDR(wpa_psk)
                       ,Address    := sockIPFromName(str := "192.168.1.1")
                       ,Subnet     := sockIPFromName(str := "255.255.255.0")
                      );
   DebugFmt(message := "netSetAPParam=    \1", v1 := rc);
END_FUNCTION;

FUNCTION ap_show;
   net();
   DebugMsg(message := " SSID=         " + net.ssid);
   IF net.security = 0 THEN
      DebugMsg(message := " Security=     None");
   ELSIF net.security = 1 THEN
      DebugFmt(message := " Security=     WPA/WPA2 PSK");
      netGetAPSecurity(sec_config := ADDR(wpa_psk));
      DebugMsg(message := "   phrase=         " + wpa_psk.phrase);
   ELSE
      DebugFmt(message := " Security=     Unknown (\1)", v1 := net.security);
   END_IF;
   DebugMsg(message := " Network=      IPv4");
   DebugMsg(message := "   Address=        " + sockIPToName(ip :=
net.Address));
   DebugMsg(message := "   Subnet=         " + sockIPToName(ip :=
net.Subnet));
END_FUNCTION;
```

```
PROGRAM example;

ap_set();
ap_show();

BEGIN
  // ...
END;
END_PROGRAM;
```

4.2.32.19 netGetAPParam (Functionblock)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900, LX5
Firmware version: 1.50.00

---

netGetAPParam will read out the parameters for a wireless network previously set from the RTCU IDE or by the netSetAPParam function.

The format of an IP address is the same as returned by the sockGetLocalIP function.

Output:

SSID : STRING
The SSID of the wireless network.

Security : INT
The type of security used by the wireless network. Use the netGetAPSecurity function to get the security configuration.

0	None	None
1	WPA/WPA2 PSK	netWLANSecurityWPA

Country : STRING
The country code where the device operates.
The ISO 3166-1 standard is used for the country code, as f.ex. 'DK' for Denmark.

Address : DINT
The IP address of the device.

Subnet : DINT
The Subnet mask of the device.

Hidden : BOOL
Whether the SSID is broadcast.

Chnl : SINT
The channel used for communication.

Declaration:

```
FUNCTION_BLOCK netGetAPParam;
VAR_OUTPUT
   SSID           : STRING;
   Security       : INT;
   Country        : STRING;
   Address        : DINT;
   Subnet         : DINT;
   Hidden         : BOOL;
```

```
   Chnl           : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   wpa_psk  : netWLANSecurityWPA;
   net      : netGetAPParam;
END_VAR;

FUNCTION ap_set;
VAR
   rc       : INT;
END_VAR;

   // Setup security
   wpa_psk.phrase  := "passphrase";

   // Setup WLAN network
   rc := netSetAPParam(
                       ssid        := "SSID"
                       ,hidden     := FALSE
                       ,chnl       := 1
                       ,sec_config := ADDR(wpa_psk)
                       ,Address    := sockIPFromName(str := "192.168.1.1")
                       ,Subnet     := sockIPFromName(str := "255.255.255.0")
                      );
   DebugFmt(message := "netSetAPParam=    \1", v1 := rc);
END_FUNCTION;

FUNCTION ap_show;
   net();
   DebugMsg(message := " SSID=         " + net.ssid);
   IF net.security = 0 THEN
      DebugMsg(message := " Security=     None");
   ELSIF net.security = 1 THEN
      DebugFmt(message := " Security=     WPA/WPA2 PSK");
      netGetAPSecurity(sec_config := ADDR(wpa_psk));
      DebugMsg(message := "   phrase=         " + wpa_psk.phrase);
   ELSE
      DebugFmt(message := " Security=     Unknown (\1)", v1 := net.security);
   END_IF;
   DebugMsg(message := " Network=      IPv4");
   DebugMsg(message := "   Address=        " + sockIPToName(ip :=
net.Address));
   DebugMsg(message := "   Subnet=         " + sockIPToName(ip :=
net.Subnet));
END_FUNCTION;

PROGRAM example;

ap_set();
ap_show();

BEGIN
  // ...
END;
END_PROGRAM;
```

4.2.32.20 netGetAPSecurity (Function)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900, LX5
Firmware version: 1.42.00

---

netGetAPSecurity will read out the security parameters for a wireless network previously set from the RTCU IDE or by the netSetAPParam function.

The function is usually called after calling netGetAPParam function block.

Output:

sec_config : PTR
The security configuration for the wireless network. The type of which is read by the netGetAPParam function block.

Returns: INT

0 - Success.
2 - Not a valid security structure
3 - Unknown security configuration / Not matching requested security
4 - Missing security structure

Declaration:

```
FUNCTION netGetAPSecurity : INT;
VAR_INPUT
   sec_config     : MANDATORY PTR;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   wpa_psk  : netWLANSecurityWPA;
   net      : netGetAPParam;
END_VAR;

FUNCTION ap_set;
VAR
   rc       : INT;
END_VAR;

   // Setup security
   wpa_psk.phrase  := "passphrase";

   // Setup WLAN network
   rc := netSetAPParam(
                       ssid        := "SSID"
                       ,hidden     := FALSE
                       ,chnl       := 1
                       ,sec_config := ADDR(wpa_psk)
                       ,Address    := sockIPFromName(str := "192.168.1.1")
                       ,Subnet     := sockIPFromName(str := "255.255.255.0")
                      );
   DebugFmt(message := "netSetAPParam=    \1", v1 := rc);
END_FUNCTION;
```

```
FUNCTION ap_show;
   net();
   DebugMsg(message := " SSID=         " + net.ssid);
   IF net.security = 0 THEN
      DebugMsg(message := " Security=     None");
   ELSIF net.security = 1 THEN
      DebugFmt(message := " Security=     WPA/WPA2 PSK");
      netGetAPSecurity(sec_config := ADDR(wpa_psk));
      DebugMsg(message := "   phrase=         " + wpa_psk.phrase);
   ELSE
      DebugFmt(message := " Security=     Unknown (\1)", v1 := net.security);
   END_IF;
   DebugMsg(message := " Network=      IPv4");
   DebugMsg(message := "   Address=        " + sockIPToName(ip :=
net.Address));
   DebugMsg(message := "   Subnet=         " + sockIPToName(ip :=
net.Subnet));
END_FUNCTION;

PROGRAM example;

ap_set();
ap_show();

BEGIN
  // ...
END;
END_PROGRAM;
```

4.2.32.21 netWLANSecurityWPA (Structure)

Architecture: NX32L
Device support: NX-200, NX-400, NX-900, LX5
Firmware version: 1.00.00

The netWLANSecurityWPA structure contains the parameters for using WPA/WPA2 pre-shared key (PSK) security.

The structure is used by the netSetWLANParam, netGetWLANSecurity, netSetAPParam and netGetAPParam functions to set and get the security parameters.

Fields:

phrase : STRING
The PSK pass phrase.

Declaration:

```
STRUCT_BLOCK netWLANSecurityWPA;
   phrase    : STRING;
END_STRUCT_BLOCK;
```

4.2.32.22 netSetMonitorParam (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 4.70 / 1.30.00

This function will set the parameters for the network monitor service.
The parameters for the network monitoring service are non-volatile and therefore need only to be set once. Parameters take effect after a device resets (by using, for example, the boardReset function).
The parameter set by netSetMonitorParam can be read by using netGetMonitorParam.

Use of the network monitoring service is recommended on network connections where the Gateway is not in use. When using the Gateway, the health of the connection is automatically monitored with the periodic self-transaction.

Network monitor service
The network monitor is a service which will monitor the health of one or more network connections, and will automatically take the necessary recovery actions required to reestablish service.
The network monitor will periodically check the health using one of the following methods:

0. Disabled
No monitoring of the connection health will occur. When the monitoring is disabled there is no dependencies on DNS or PING servers in the network environment.
For networks where there are no APN present, it is recommended to use the monitoring in the RTCU Gateway or enable the ICMP PING monitoring.
Disabling the monitoring will have the same effect as setting $DNS=0 in the Network Settings dialog.

1. A DNS lookup.
In this case, the monitor will perform a domain name look up, first on the primary DNS server, and if this fails because the DNS server did not respond, it will try again on the secondary DNS server (if present).
The DNS server IP addresses will be derived from the network information.

2. A ICMP PING.
In this case, the monitor will send an ICMP PING to the primary server, and if the fails, then again on the secondary server (if present).
The primary and secondary servers used, must be set by the application.

The gprsSetMonitorParm function sets the parameters for the mobile network, but, unlike the netSetMonitorParam function, is limited to using the DNS lookup method of health check.

Input:

iface : SINT (*default 1*)
The network interface. 1 = Mobile network, 2 = LAN network, etc. (See Network)
Please note that X32, NX32 and NX32L devices only support monitoring of the mobile network.

Enabled : BOOL
Enable/Disable monitoring of the network connection.

method : SINT (*default 1*)
The method used to check the health of the connection. (See above)
1 = DNS lookup, 2 = Ping address.

MaxAttempt : INT (1..60, *default 3*)
Maximum number of failed health check attempts before the necessary recovery actions are initiated.
Recommended is 3 times.

AliveFreq : DINT (30..60000 seconds, *default 300*)
The number of seconds between successful health checks.
Recommended alive-frequency is 300 seconds.

ping1 : DINT
The IP address of the primary server. (only used by ping)

ping2 : DINT
The IP address of the secondary server. (only used by ping)

Returns:

1	- Success.
0	- Interface not available or not supported.
-1	- Illegal network interface.
-2	- Illegal method.
-3	- Illegal parameter.

Declaration:

```
FUNCTION netSetMonitorParam;
VAR_INPUT
   iface      : SINT := 1;
   Enabled    : BOOL;
   method     : SINT := 1;
   MaxAttempt : INT  := 3;
   AliveFreq  : DINT := 300;
   ping1      : DINT;
   ping2      : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   monitor : netGetMonitorParam;
END_VAR;

// Check settings
monitor(iface := _NET_IFACE_CELLULAR);
IF NOT monitor.Enabled OR monitor.method <> 1 OR monitor.MaxAttempt <> 3 OR
monitor.AliveFreq <> 180 THEN
   // Set network monitor parameters
   netSetMonitorParam(iface:=_NET_IFACE_CELLULAR, Enabled := TRUE, method :=
1, MaxAttempt := 3, AliveFreq := 180);
   boardReset();
END_IF;

// Turn on power to the GSM module
gsmPower(power := ON);
gprsOpen();

BEGIN
   ...
END;
END_PROGRAM;
```

4.2.32.23 netGetMonitorParam (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 4.70 / 1.30.00

This function will read out the parameters for the network monitoring service previously set by using the netSetMonitorParam function.

Input:

iface : SINT (*default 1*)
The network interface. 1 = Mobile network, 2 = LAN network, etc. (See Network)
Please note that X32 and NX32 devices only supports monitoring of the mobile network.

Output:

Status : INT
1 - Success.
0 - Interface not available or not supported.
-1 - Illegal network interface.

Enabled : BOOL
Determines whether the network monitor service is enabled.

method : SINT
The method used by the network monitoring service to check the health of the connection.
1 = DNS lookup, 2 = Ping address.

MaxAttempt : INT
Maximum number of failed health check attempts before the necessary recovery actions are initiated.

AliveFreq : DINT
The number of seconds between successful health checks.

ping1 : DINT
The IP address of the primary server. (only used by ping)

ping2 : DINT
The IP address of the secondary server. (only used by ping)

Declaration:

```
FUNCTION_BLOCK netGetMonitorParam;
VAR_INPUT
   iface      : SINT := 1;
END_VAR;
VAR_OUTPUT
   status     : INT;
   Enabled    : BOOL;
   method     : SINT;
   MaxAttempt : INT;
   AliveFreq  : DINT;
   ping1      : DINT;
   ping2      : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   monitor : netGetMonitorParam;
END_VAR;

// Check settings
monitor(iface := _NET_IFACE_CELLULAR);
IF NOT monitor.Enabled OR monitor.method <> 1 OR monitor.MaxAttempt <> 3 OR
monitor.AliveFreq <> 180 THEN
   // Set network monitor parameters
   netSetMonitorParam(iface:=_NET_IFACE_CELLULAR, Enabled := TRUE, method :=
1, MaxAttempt := 3, AliveFreq := 180);
   boardReset();
END_IF;

// Turn on power to the GSM module
gsmPower(power := ON);
gprsOpen();

BEGIN
   ...
END;
END_PROGRAM;
```

4.2.32.24 netSetNATParam (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.50.00

This sets the NAT parameters for a network interface. It has the same effect as setting the parameters via Device: Network: Network Settings.
The NAT parameters for the network interface are non-volatile and therefore need only to be set once.
The parameters set by netSetNATParam can be read using netGetNATParam.

Please note that not all interfaces support both forward and gateway. (See Device: Network: Network Settings for which interfaces support which)

Input:

Iface : SINT
The network interface. (See Network)

Forward : BOOL *(default FALSE)*
Controls whether requests from the interface will be forwarded to other interfaces.

Gateway : BOOL *(default FALSE)*
Controls whether the interface will be used as a gateway for requests.
Only one interface can be set as Gateway at the same time.

Returns:

1 - Success.
0 - Interface not available or not supported.
-2 - Illegal network interface.

-3 - Illegal parameter.
-4 - Another interface is set as gateway.

Declaration:

```
FUNCTION netSetNATParam;
VAR_INPUT
   Iface      : MANDATORY SINT;
   Forward    : BOOL := FALSE;
   Gateway    : BOOL := FALSE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   ap   : netGetAPParam;
   dhcp : netGetDHCPParam;
   lan  : netGetLANParam;
   nat  : netGetNATParam;
   wpa  : netWLANSecurityWPA;
END_VAR;

FUNCTION booltostr : STRING;
VAR_INPUT
  v : BOOL;
END_VAR;
IF v THEN booltostr := "Yes"; ELSE booltostr := "No"; END_IF;
END_FUNCTION;

FUNCTION setup_lan;

   // Configure LAN
   netSetLANParam(iface := 2, DHCP := TRUE, AutoDNS := TRUE);

   // Configure NAT
   netSetNATParam(iface := 2, forward := FALSE, gateway := TRUE);

END_FUNCTION;

FUNCTION setup_ap;

   // Configure Access Point
   wpa.phrase               := "pass phrase";
   netSetAPParam(
                 ssid       := "RTCU network",
                 sec_config := ADDR(wpa),
                 hidden     := FALSE,
                 chnl       := 1,
                 Address    := sockIPFromName(str := "192.168.1.1"),
                 Subnet     := sockIPFromName(str := "255.255.255.0")
                );

   // Configure NAT
   netSetNATParam(iface := 5, forward := TRUE, gateway := FALSE);

   // Configure DHCP
   netSetDHCPParam(
                   iface    := 5,
                   enable   := ON,
                   time     := 24,
```

```
                    first    := sockIPFromName(str := "192.168.1.100"),
                    last     := sockIPFromName(str := "192.168.1.200"),
                    dns_type := 1,
                    applynow := TRUE
                   );

END_FUNCTION;

FUNCTION show_lan;

   // Show
   DebugMsg(message := "===== LAN =====");

   // Show LAN setup
   lan(iface := 2);
   DebugMsg(message := " Network=      IPv4");
   DebugMsg(message := "   DHCP=           " + booltostr(v := lan.dhcp));
   DebugMsg(message := "   IP=             " + sockIPToName(ip := lan.ip));
   DebugMsg(message := "   Subnet=         " + sockIPToName(ip :=
lan.SubnetMask));
   DebugMsg(message := "   Gw=             " + sockIPToName(ip :=
lan.Gateway));
   DebugMsg(message := "   Auto DNS=       " + booltostr(v := lan.AutoDNS));
   DebugMsg(message := "   DNS=            " + sockIPToName(ip := lan.DNS1));
   DebugMsg(message := "   DNS=            " + sockIPToName(ip := lan.DNS2));

   // Show NAT setup
   nat(iface := 2);
   DebugMsg(message := " NAT");
   DebugMsg(message := "   forward=        " + booltostr(v := nat.forward));
   DebugMsg(message := "   gateway=        " + booltostr(v := nat.gateway));

END_FUNCTION;

FUNCTION show_ap;

   // Show
   DebugMsg(message := "===== Access Point =====");

   // Show AP setup
   ap();
   DebugMsg(message := " SSID=         " + ap.ssid);
   IF ap.security = 0 THEN
      DebugMsg(message := " Security=     None");
   ELSIF ap.security = 1 THEN
      DebugFmt(message := " Security=     WPA/WPA2 PSK");
      netGetAPSecurity(sec_config := ADDR(wpa));
      DebugMsg(message := "   phrase=         " + wpa.phrase);
   ELSE
      DebugFmt(message := " Security=     Unknown (\1)", v1 := ap.security);
   END_IF;
   DebugMsg(message := " Network=      IPv4");
   DebugMsg(message := "   Address=        " + sockIPToName(ip :=
ap.Address));
   DebugMsg(message := "   Subnet=         " + sockIPToName(ip := ap.Subnet));

   // Show NAT setup
   nat(iface := 5);
   DebugMsg(message := " NAT");
   DebugMsg(message := "   forward=        " + booltostr(v := nat.forward));
   DebugMsg(message := "   gateway=        " + booltostr(v := nat.gateway));

   // Show DHCP setup
   dhcp(iface := 5);
```

```
   DebugMsg(message := " DHCP");
   DebugMsg(message := "   enable=          " + booltostr(v := dhcp.enable));
   DebugFmt(message := "   lease time=      \1 Hours", v1 := dhcp.time);
   DebugMsg(message := "   subnet=          " + sockIPToName(ip :=
dhcp.Subnet));
   DebugMsg(message := "   range, start=    " + sockIPToName(ip :=
dhcp.first));
   DebugMsg(message := "   range, end=      " + sockIPToName(ip := dhcp.last));
   IF dhcp.dns_type = 1 THEN
      DebugMsg(message := "   DNS type=        Self/Auto");
   ELSIF dhcp.dns_type = 2 THEN
      DebugMsg(message := "   DNS type=        Static");
   ELSE
      DebugFmt(message := "   DNS type=        Unknown (\1)", v1 :=
dhcp.dns_type);
   END_IF;
   DebugMsg(message := "   DNS=             " + sockIPToName(ip := dhcp.DNS1));
   DebugMsg(message := "   DNS=             " + sockIPToName(ip := dhcp.DNS2));

END_FUNCTION;

PROGRAM example;

// Setup
setup_lan();
setup_ap();

// Show
show_lan();
show_ap();

// Start
netOpen(iface := 2);
netOpen(iface := 5);

BEGIN
  // ...
END;
END_PROGRAM;
```

4.2.32.25 netGetNATParam (Functionblock)

Architecture: NX32L
Device support: All
Firmware version: 1.50.00

This function will read out the NAT parameters for the network interface previously set by using the netSetNATParam function.

Input:

Iface : SINT
The network interface. (See Network)

Output:

Forward : BOOL
Determines whether requests from the interface will be forwarded to other interfaces

Gateway : BOOL
Determines whether the interface will be used as a gateway for requests.

Declaration:

```
FUNCTION_BLOCK netGetNATParam;
VAR_INPUT
   Iface      : MANDATORY SINT;
END_VAR;
VAR_OUTPUT
   Forward    : BOOL;
   Gateway    : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   ap   : netGetAPParam;
   dhcp : netGetDHCPParam;
   lan  : netGetLANParam;
   nat  : netGetNATParam;
   wpa  : netWLANSecurityWPA;
END_VAR;

FUNCTION booltostr : STRING;
VAR_INPUT
   v : BOOL;
END_VAR;
IF v THEN booltostr := "Yes"; ELSE booltostr := "No"; END_IF;
END_FUNCTION;

FUNCTION setup_lan;

   // Configure LAN
   netSetLANParam(iface := 2, DHCP := TRUE, AutoDNS := TRUE);

   // Configure NAT
   netSetNATParam(iface := 2, forward := FALSE, gateway := TRUE);

END_FUNCTION;

FUNCTION setup_ap;

   // Configure Access Point
   wpa.phrase               := "pass phrase";
   netSetAPParam(
                 ssid       := "RTCU network",
                 sec_config := ADDR(wpa),
                 hidden     := FALSE,
                 chnl       := 1,
                 Address    := sockIPFromName(str := "192.168.1.1"),
                 Subnet     := sockIPFromName(str := "255.255.255.0")
                );

   // Configure NAT
   netSetNATParam(iface := 5, forward := TRUE, gateway := FALSE);

   // Configure DHCP
   netSetDHCPParam(
                   iface    := 5,
```

```
                    enable   := ON,
                    time     := 24,
                    first    := sockIPFromName(str := "192.168.1.100"),
                    last     := sockIPFromName(str := "192.168.1.200"),
                    dns_type := 1,
                    applynow := TRUE
                   );

END_FUNCTION;

FUNCTION show_lan;

   // Show
   DebugMsg(message := "===== LAN =====");

   // Show LAN setup
   lan(iface := 2);
   DebugMsg(message := " Network=      IPv4");
   DebugMsg(message := "   DHCP=           " + booltostr(v := lan.dhcp));
   DebugMsg(message := "   IP=             " + sockIPToName(ip := lan.ip));
   DebugMsg(message := "   Subnet=         " + sockIPToName(ip :=
lan.SubnetMask));
   DebugMsg(message := "   Gw=             " + sockIPToName(ip :=
lan.Gateway));
   DebugMsg(message := "   Auto DNS=       " + booltostr(v := lan.AutoDNS));
   DebugMsg(message := "   DNS=            " + sockIPToName(ip := lan.DNS1));
   DebugMsg(message := "   DNS=            " + sockIPToName(ip := lan.DNS2));

   // Show NAT setup
   nat(iface := 2);
   DebugMsg(message := " NAT");
   DebugMsg(message := "   forward=        " + booltostr(v := nat.forward));
   DebugMsg(message := "   gateway=        " + booltostr(v := nat.gateway));

END_FUNCTION;

FUNCTION show_ap;

   // Show
   DebugMsg(message := "===== Access Point =====");

   // Show AP setup
   ap();
   DebugMsg(message := " SSID=         " + ap.ssid);
   IF ap.security = 0 THEN
      DebugMsg(message := " Security=     None");
   ELSIF ap.security = 1 THEN
      DebugFmt(message := " Security=     WPA/WPA2 PSK");
      netGetAPSecurity(sec_config := ADDR(wpa));
      DebugMsg(message := "   phrase=         " + wpa.phrase);
   ELSE
      DebugFmt(message := " Security=     Unknown (\1)", v1 := ap.security);
   END_IF;
   DebugMsg(message := " Network=      IPv4");
   DebugMsg(message := "   Address=        " + sockIPToName(ip :=
ap.Address));
   DebugMsg(message := "   Subnet=         " + sockIPToName(ip := ap.Subnet));

   // Show NAT setup
   nat(iface := 5);
   DebugMsg(message := " NAT");
   DebugMsg(message := "   forward=        " + booltostr(v := nat.forward));
   DebugMsg(message := "   gateway=        " + booltostr(v := nat.gateway));
```

```
   // Show DHCP setup
   dhcp(iface := 5);
   DebugMsg(message := " DHCP");
   DebugMsg(message := "   enable=          " + booltostr(v := dhcp.enable));
   DebugFmt(message := "   lease time=      \1 Hours", v1 := dhcp.time);
   DebugMsg(message := "   subnet=          " + sockIPToName(ip :=
dhcp.Subnet));
   DebugMsg(message := "   range, start=    " + sockIPToName(ip :=
dhcp.first));
   DebugMsg(message := "   range, end=      " + sockIPToName(ip := dhcp.last));
   IF dhcp.dns_type = 1 THEN
      DebugMsg(message := "   DNS type=        Self/Auto");
   ELSIF dhcp.dns_type = 2 THEN
      DebugMsg(message := "   DNS type=        Static");
   ELSE
      DebugFmt(message := "   DNS type=        Unknown (\1)", v1 :=
dhcp.dns_type);
   END_IF;
   DebugMsg(message := "   DNS=             " + sockIPToName(ip := dhcp.DNS1));
   DebugMsg(message := "   DNS=             " + sockIPToName(ip := dhcp.DNS2));

END_FUNCTION;

PROGRAM example;

// Setup
setup_lan();
setup_ap();

// Show
show_lan();
show_ap();

// Start
netOpen(iface := 2);
netOpen(iface := 5);

BEGIN
  // ...
END;
END_PROGRAM;
```

4.2.32.26 netSetDHCPParam (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.50.00

This sets the DHCP parameters for a network interface. It has the same effect as setting the parameters via Device: Network: Network Settings.
The DHCP parameters for the network interface are non-volatile and therefore need only to be set once. Parameters take effect after a device resets (by using, for example, the boardReset function), or immediately if the applynow parameter is set.
The parameters set by netSetDHCPParam can be read by using netGetDHCPParam.

Input:

Iface : SINT
The network interface. (See Network)

Enable : BOOL *(default FALSE)*
Control whether the DHCP server will listen for DHCP requests from the network interface.

Applynow : BOOL *(default FALSE)*
If TRUE, the DHCP server will be restarted, so the parameters are applied. Otherwise the device must be reset for the parameters to be applied.

Time : INT *(1..24, default 12)*
The lease time in hours.
This is the amount of time a connected client has the IP address, before it needs to be renewed.

First : DINT
The first IP address in the range of dynamic IP addresses.

Last : DINT
The last IP address in the range of dynamic IP addresses.

DNS_type : SINT *(default 1)*
1 - Use the device as DNS server. Requests will be forwarded to the DNS servers from the NAT gateway interface.
2 - Use static DNS servers from the DNS1 and DNS2 parameters below.

DNS1 : DINT
The IP address of the primary DNS server.

DNS2 : DINT
The IP address of the secondary DNS server.

Returns:

1 - Success.
0 - Not supported.
-1 - Illegal network interface.
-2 - The network interface is not supported.
-3 - DHCP server can not be enabled if the network interface uses dynamic address
-4 - Illegal lease time.
-5 - The dynamic IP address range is not in the correct subnet.
-6 - The dynamic IP address range is in the wrong order.
-7 - The dynamic IP address range is too small. (10 or more addresses are required)
-8 - The dynamic IP address range includes the interface IP address
-9 - Illegal dns_type selected.
-10 - DNS server IP address is missing.

Declaration:

```
FUNCTION netSetDHCPParam;
VAR_INPUT
   Iface      : SINT;
   Enable     : BOOL := FALSE;
   Aplynow    : BOOL := FALSE;
   Time       : INT := 12;
   First      : DINT;
   Last       : DINT;
   DNS_type   : SINT := 1;
   DNS1       : DINT;
   DNS2       : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   ap   : netGetAPParam;
   dhcp : netGetDHCPParam;
   lan  : netGetLANParam;
   nat  : netGetNATParam;
   wpa  : netWLANSecurityWPA;
END_VAR;

FUNCTION booltostr : STRING;
VAR_INPUT
   v : BOOL;
END_VAR;
IF v THEN booltostr := "Yes"; ELSE booltostr := "No"; END_IF;
END_FUNCTION;

FUNCTION setup_lan;

   // Configure LAN
   netSetLANParam(iface := 2, DHCP := TRUE, AutoDNS := TRUE);

   // Configure NAT
   netSetNATParam(iface := 2, forward := FALSE, gateway := TRUE);

END_FUNCTION;

FUNCTION setup_ap;

   // Configure Access Point
   wpa.phrase               := "pass phrase";
   netSetAPParam(
                 ssid       := "RTCU network",
                 sec_config := ADDR(wpa),
                 hidden     := FALSE,
                 chnl       := 1,
                 Address    := sockIPFromName(str := "192.168.1.1"),
                 Subnet     := sockIPFromName(str := "255.255.255.0")
                );

   // Configure NAT
   netSetNATParam(iface := 5, forward := TRUE, gateway := FALSE);

   // Configure DHCP
   netSetDHCPParam(
                   iface    := 5,
                   enable   := ON,
                   time     := 24,
                   first    := sockIPFromName(str := "192.168.1.100"),
                   last     := sockIPFromName(str := "192.168.1.200"),
                   dns_type := 1,
                   applynow := TRUE
                  );

END_FUNCTION;

FUNCTION show_lan;

   // Show
   DebugMsg(message := "===== LAN =====");

   // Show LAN setup
```

```
   lan(iface := 2);
   DebugMsg(message := " Network=      IPv4");
   DebugMsg(message := "   DHCP=           " + booltostr(v := lan.dhcp));
   DebugMsg(message := "   IP=             " + sockIPToName(ip := lan.ip));
   DebugMsg(message := "   Subnet=         " + sockIPToName(ip :=
lan.SubnetMask));
   DebugMsg(message := "   Gw=             " + sockIPToName(ip :=
lan.Gateway));
   DebugMsg(message := "   Auto DNS=       " + booltostr(v := lan.AutoDNS));
   DebugMsg(message := "   DNS=            " + sockIPToName(ip := lan.DNS1));
   DebugMsg(message := "   DNS=            " + sockIPToName(ip := lan.DNS2));

   // Show NAT setup
   nat(iface := 2);
   DebugMsg(message := " NAT");
   DebugMsg(message := "   forward=        " + booltostr(v := nat.forward));
   DebugMsg(message := "   gateway=        " + booltostr(v := nat.gateway));

END_FUNCTION;

FUNCTION show_ap;

   // Show
   DebugMsg(message := "===== Access Point =====");

   // Show AP setup
   ap();
   DebugMsg(message := " SSID=         " + ap.ssid);
   IF ap.security = 0 THEN
      DebugMsg(message := " Security=     None");
   ELSIF ap.security = 1 THEN
      DebugFmt(message := " Security=     WPA/WPA2 PSK");
      netGetAPSecurity(sec_config := ADDR(wpa));
      DebugMsg(message := "   phrase=         " + wpa.phrase);
   ELSE
      DebugFmt(message := " Security=     Unknown (\1)", v1 := ap.security);
   END_IF;
   DebugMsg(message := " Network=      IPv4");
   DebugMsg(message := "   Address=        " + sockIPToName(ip :=
ap.Address));
   DebugMsg(message := "   Subnet=         " + sockIPToName(ip := ap.Subnet));

   // Show NAT setup
   nat(iface := 5);
   DebugMsg(message := " NAT");
   DebugMsg(message := "   forward=        " + booltostr(v := nat.forward));
   DebugMsg(message := "   gateway=        " + booltostr(v := nat.gateway));

   // Show DHCP setup
   dhcp(iface := 5);
   DebugMsg(message := " DHCP");
   DebugMsg(message := "   enable=         " + booltostr(v := dhcp.enable));
   DebugFmt(message := "   lease time=     \1 Hours", v1 := dhcp.time);
   DebugMsg(message := "   range, start=   " + sockIPToName(ip :=
dhcp.first));
   DebugMsg(message := "   range, end=     " + sockIPToName(ip := dhcp.last));
   IF dhcp.dns_type = 1 THEN
      DebugMsg(message := "   DNS type=       Self/Auto");
   ELSIF dhcp.dns_type = 2 THEN
      DebugMsg(message := "   DNS type=       Static");
   ELSE
      DebugFmt(message := "   DNS type=       Unknown (\1)", v1 :=
dhcp.dns_type);
   END_IF;
```

```
    DebugMsg(message := "   DNS=            " + sockIPToName(ip := dhcp.DNS1));
    DebugMsg(message := "   DNS=            " + sockIPToName(ip := dhcp.DNS2));

END_FUNCTION;

PROGRAM example;

// Setup
setup_lan();
setup_ap();

// Show
show_lan();
show_ap();

// Start
netOpen(iface := 2);
netOpen(iface := 5);

BEGIN
  // ...
END;
END_PROGRAM;
```

4.2.32.27 netGetDHCPParam (Functionblock)

Architecture: NX32L
Device support: All
Firmware version: 1.50.00

This function will read out the DHCP parameters for the network interface previously set by the netSetDHCPParam function.

Input:

Iface : SINT
The network interface. (See Network)

Output:

Status : SINT

1	- Success.
0	- Interface not available or not supported.
-1	- Illegal network interface.

Enable : BOOL
Determines whether the DHCP service is enabled.

Time : INT
The lease time in hours.

First : DINT
The first IP address in the range of dynamic IP addresses.

Last : DINT
The last IP address in the range of dynamic IP addresses.

DNS_type : SINT

1 - The device is used as a DNS server. Requests are forwarded to the DNS servers from the NAT gateway interface.
2 - Static DNS servers are used.

DNS1 : DINT
The IP address of the primary DNS server. (only used when dns_type = 2)

DNS2 : DINT
The IP address of the secondary DNS server. (only used when dns_type = 2)

Declaration:

```
FUNCTION_BLOCK netGetDHCPParam;
VAR_INPUT
   Iface      : SINT;
END_VAR;
VAR_OUTPUT
   Status     : SINT;
   Enable     : BOOL;
   Time       : INT;
   First      : DINT;
   Last       : DINT;
   DNS_type   : SINT;
   DNS1       : DINT;
   DNS2       : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   ap   : netGetAPParam;
   dhcp : netGetDHCPParam;
   lan  : netGetLANParam;
   nat  : netGetNATParam;
   wpa  : netWLANSecurityWPA;
END_VAR;

FUNCTION booltostr : STRING;
VAR_INPUT
   v : BOOL;
END_VAR;
IF v THEN booltostr := "Yes"; ELSE booltostr := "No"; END_IF;
END_FUNCTION;

FUNCTION setup_lan;

   // Configure LAN
   netSetLANParam(iface := 2, DHCP := TRUE, AutoDNS := TRUE);

   // Configure NAT
   netSetNATParam(iface := 2, forward := FALSE, gateway := TRUE);

END_FUNCTION;

FUNCTION setup_ap;

   // Configure Access Point
   wpa.phrase              := "pass phrase";
```

```
   netSetAPParam(
                 ssid       := "RTCU network",
                 sec_config := ADDR(wpa),
                 hidden     := FALSE,
                 chnl       := 1,
                 Address    := sockIPFromName(str := "192.168.1.1"),
                 Subnet     := sockIPFromName(str := "255.255.255.0")
                );

   // Configure NAT
   netSetNATParam(iface := 5, forward := TRUE, gateway := FALSE);

   // Configure DHCP
   netSetDHCPParam(
                  iface    := 5,
                  enable   := ON,
                  time     := 24,
                  first    := sockIPFromName(str := "192.168.1.100"),
                  last     := sockIPFromName(str := "192.168.1.200"),
                  dns_type := 1,
                  applynow := TRUE
                 );

END_FUNCTION;

FUNCTION show_lan;

   // Show
   DebugMsg(message := "===== LAN =====");

   // Show LAN setup
   lan(iface := 2);
   DebugMsg(message := " Network=      IPv4");
   DebugMsg(message := "   DHCP=           " + booltostr(v := lan.dhcp));
   DebugMsg(message := "   IP=             " + sockIPToName(ip := lan.ip));
   DebugMsg(message := "   Subnet=         " + sockIPToName(ip :=
lan.SubnetMask));
   DebugMsg(message := "   Gw=             " + sockIPToName(ip :=
lan.Gateway));
   DebugMsg(message := "   Auto DNS=       " + booltostr(v := lan.AutoDNS));
   DebugMsg(message := "   DNS=            " + sockIPToName(ip := lan.DNS1));
   DebugMsg(message := "   DNS=            " + sockIPToName(ip := lan.DNS2));

   // Show NAT setup
   nat(iface := 2);
   DebugMsg(message := " NAT");
   DebugMsg(message := "   forward=        " + booltostr(v := nat.forward));
   DebugMsg(message := "   gateway=        " + booltostr(v := nat.gateway));

END_FUNCTION;

FUNCTION show_ap;

   // Show
   DebugMsg(message := "===== Access Point =====");

   // Show AP setup
   ap();
   DebugMsg(message := " SSID=         " + ap.ssid);
   IF ap.security = 0 THEN
      DebugMsg(message := " Security=     None");
   ELSIF ap.security = 1 THEN
      DebugFmt(message := " Security=     WPA/WPA2 PSK");
      netGetAPSecurity(sec_config := ADDR(wpa));
```

```
      DebugMsg(message := "   phrase=         " + wpa.phrase);
   ELSE
      DebugFmt(message := " Security=     Unknown (\1)", v1 := ap.security);
   END_IF;
   DebugMsg(message := " Network=      IPv4");
   DebugMsg(message := "   Address=        " + sockIPToName(ip :=
ap.Address));
   DebugMsg(message := "   Subnet=         " + sockIPToName(ip := ap.Subnet));

   // Show NAT setup
   nat(iface := 5);
   DebugMsg(message := " NAT");
   DebugMsg(message := "   forward=        " + booltostr(v := nat.forward));
   DebugMsg(message := "   gateway=        " + booltostr(v := nat.gateway));

   // Show DHCP setup
   dhcp(iface := 5);
   DebugMsg(message := " DHCP");
   DebugMsg(message := "   enable=         " + booltostr(v := dhcp.enable));
   DebugFmt(message := "   lease time=     \1 Hours", v1 := dhcp.time);
   DebugMsg(message := "   range, start=   " + sockIPToName(ip :=
dhcp.first));
   DebugMsg(message := "   range, end=     " + sockIPToName(ip := dhcp.last));
   IF dhcp.dns_type = 1 THEN
      DebugMsg(message := "   DNS type=       Self/Auto");
   ELSIF dhcp.dns_type = 2 THEN
      DebugMsg(message := "   DNS type=       Static");
   ELSE
      DebugFmt(message := "   DNS type=       Unknown (\1)", v1 :=
dhcp.dns_type);
   END_IF;
   DebugMsg(message := "   DNS=            " + sockIPToName(ip := dhcp.DNS1));
   DebugMsg(message := "   DNS=            " + sockIPToName(ip := dhcp.DNS2));

END_FUNCTION;

PROGRAM example;

// Setup
setup_lan();
setup_ap();

// Show
show_lan();
show_ap();

// Start
netOpen(iface := 2);
netOpen(iface := 5);

BEGIN
  // ...
END;
END_PROGRAM;
```

4.2.32.28 netPortForwardClear (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.74.00

This clears the port forwarding rules previously set by the netPortForwardSet function.

The rules are cleared immediately and no more connections will be forwarded; existing connections are not affected.

Input:
None.

Returns:
1 - Success.
0 - Not supported.

Declaration:
```
FUNCTION netPortForwardClear;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM example;

// ...
netPortForwardClear();
// ...

BEGIN
  // ...
END;
END_PROGRAM;
```

4.2.32.29 netPortForwardGet (Functionblock)

Architecture: NX32L
Device support: All
Firmware version: 1.74.00

This function will read out a port forward rule previously set by the netPortForwardSet function.

Input:
index : INT (1..32)
The index of the rule to read.

Output:
Status : SINT
1 - Success.
0 - Not supported.
-1 - Illegal index.

dst_addr : DINT
This is the internal IP address where the communication is forwarded to.

dst_port : DINT
This is the internal port where the communication is forwarded to

src_port : DINT
This is the external port where the communication is forwarded from.

tcp : BOOL
This determines if TCP communication is forwarded.

udp : BOOL
This determines if UDP communication is forwarded.

Declaration:

```
FUNCTION_BLOCK netPortForwardGet;
VAR_INPUT
   index      : INT;
END_VAR;
VAR_OUTPUT
   Status     : SINT;
   dst_addr   : DINT;
   dst_port   : DINT;
   src_port   : DINT;
   tcp        : BOOL;
   udp        : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   _rule_get_   : netPortForwardGet;
END_VAR;

FUNCTION pad_string : STRING;
VAR_INPUT
   str      : STRING;
   len      : INT;
END_VAR;

   pad_string := str;
   WHILE strLen(str := pad_string) < len DO
      pad_string := pad_string + " ";
   END_WHILE;

END_FUNCTION;

FUNCTION show_rule
VAR
   str   : STRING;
END_VAR;

   // Add index
   IF _rule_get_.index < 10 THEN
      str := pad_string(str := "", len := 3);
   ELSE
      str := pad_string(str := "", len := 2);
   END_IF;
   str := str + intToStr(v := _rule_get_.index);
   str := str + ": ";

   IF _rule_get_.status = 1 THEN
      // Add port
      str := str + dintToStr(v := _rule_get_.src_port);
```

```
      str := pad_string(str := str, len := 12);
      str := str + "=> ";

      // Add dest
      str := str + sockIPToName(ip := _rule_get_.dst_addr);
      str := str + ":";
      str := str + dintToStr(v := _rule_get_.dst_port);
      str := pad_string(str := str, len := 38);

      // Add protocol
      IF _rule_get_.tcp THEN str := str + "[TCP]"; END_IF;
      IF _rule_get_.udp THEN str := str + "[UDP]"; END_IF;

   ELSE
      str := str + "<NO RULE>";

   END_IF;

   // Show
   DebugMsg(message := str);

END_FUNCTION;

FUNCTION show_rules
VAR
   i : INT;
END_VAR;

   DebugMsg(message :=
"------------------------------------------------------------");
   DebugMsg(message := " Get port forwarding rules...");

   // Iterate
   FOR i := 1 TO 32 DO
      _rule_get_(index := i);
      show_rule();

   END_FOR;

END_FUNCTION;

PROGRAM example;
VAR
   rc : INT;
END_VAR;

   // Set port forward
   rc := netPortForwardSet(
                           index    := 1,
                           dst_addr := sockIPFromName(str := "192.168.1.153"),
                           dst_port := 5020,
                           src_port := 2000,
                           tcp      := TRUE,
                           udp      := FALSE
                          );
   DebugFmt(message := "netPortForwardSet = \1", v1 := rc);

   // Show rules
   show_rules();

   // Clear port forward
   rc := netPortForwardSet(index := 1, src_port := 0);
   DebugFmt(message := "netPortForwardSet = \1", v1 := rc);
```

```
   // Show rules
   show_rules();

BEGIN
END;
END_PROGRAM;
```

4.2.32.30 netPortForwardSet (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.74.00

This sets a port forward rule. It has the same effect as setting the port forwarding rules via Device: Network: Port Forward Rules in the RTCU IDE.
When a new rule is set, new connections will be forwarded immediately.
A rule can be removed by setting the src_port to 0.

Input:

index : INT (1..32)
The index of the rule to set.

dst_addr : DINT
This is the internal IP address where the communication is forwarded to.

dst_port : DINT
This is the internal port where the communication is forwarded to

src_port : DINT (*default 0*)
This is the external port where the communication is forwarded from.
If this is set to zero the rule will be removed.

tcp : BOOL
This determines if TCP communication is forwarded.

udp : BOOL
This determines if UDP communication is forwarded.

Returns:

1	- Success.
0	- Not supported.
-1	- Illegal index.
-2	- Illegal parameter.

Declaration:

```
FUNCTION netPortForwardSet;
VAR_INPUT
   index      : INT;
   dst_addr   : DINT;
   dst_port   : DINT;
   src_port   : DINT := 0;
   tcp        : BOOL;
   udp        : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   _rule_get_    : netPortForwardGet;
END_VAR;

FUNCTION pad_string : STRING;
VAR_INPUT
   str      : STRING;
   len      : INT;
END_VAR;

   pad_string := str;
   WHILE strLen(str := pad_string) < len DO
      pad_string := pad_string + " ";
   END_WHILE;

END_FUNCTION;

FUNCTION show_rule
VAR
   str   : STRING;
END_VAR;

   // Add index
   IF _rule_get_.index < 10 THEN
      str := pad_string(str := "", len := 3);
   ELSE
      str := pad_string(str := "", len := 2);
   END_IF;
   str := str + intToStr(v := _rule_get_.index);
   str := str + ": ";

   IF _rule_get_.status = 1 THEN
      // Add port
      str := str + dintToStr(v := _rule_get_.src_port);
      str := pad_string(str := str, len := 12);
      str := str + "=> ";

      // Add dest
      str := str + sockIPToName(ip := _rule_get_.dst_addr);
      str := str + ":";
      str := str + dintToStr(v := _rule_get_.dst_port);
      str := pad_string(str := str, len := 38);

      // Add protocol
      IF _rule_get_.tcp THEN str := str + "[TCP]"; END_IF;
      IF _rule_get_.udp THEN str := str + "[UDP]"; END_IF;

   ELSE
      str := str + "<NO RULE>";

   END_IF;

   // Show
   DebugMsg(message := str);

END_FUNCTION;

FUNCTION show_rules
VAR
   i : INT;
```

```
END_VAR;

   DebugMsg(message :=
"------------------------------------------------------------");
   DebugMsg(message := " Get port forwarding rules...");

   // Iterate
   FOR i := 1 TO 32 DO
      _rule_get_(index := i);
      show_rule();

   END_FOR;

END_FUNCTION;

PROGRAM example;
VAR
   rc : INT;
END_VAR;

   // Set port forward
   rc := netPortForwardSet(
                           index    := 1,
                           dst_addr := sockIPFromName(str := "192.168.1.153"),
                           dst_port := 5020,
                           src_port := 2000,
                           tcp      := TRUE,
                           udp      := FALSE
                          );
   DebugFmt(message := "netPortForwardSet = \1", v1 := rc);

   // Show rules
   show_rules();

   // Clear port forward
   rc := netPortForwardSet(index := 1, src_port := 0);
   DebugFmt(message := "netPortForwardSet = \1", v1 := rc);

   // Show rules
   show_rules();

BEGIN
END;
END_PROGRAM;
```

4.2.32.31 netRouteAdd (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.80.00

This function adds a network route to the system.

The routes tells the system how to send socket data to the receiver.
Normally the system handles it automatically based on the subnet for each network, using the NAT Gateway as the default route, see netSetNATParam.
When using more complex networks (Multiple subnets, VPN, or similar) it may be necessary to access devices on different subnets that can not use the default route.
In these cases, the netRouteAdd function can be used to create new routes for specific IP address ranges.

It is also possible to create multiple prioritized routes to the same destination by using different metric values, providing automatic fallback when an interface is disconnected.

The routes added with this function are not persistent across reset.

Input:

index : INT (1..25)
The index of the route.

iface : SINT
The network interface to use for the route. (See Network for available interfaces)

ip : DINT
The destination IP address or network for the route.
This is used in conjunction with the netmask to define the IP ranges handled by this route.
To match a single IP address, set `ip` to the IP address and `netmask` to `255.255.255.255`.
To mach an IP range, set `netmask` to the netmask for the range and IP to the first IP address that matches the mask, i.e. if using the netmask `255.255.255.0`, the IP address could be `192.168.1.0` and would match `192.168.1.1 - 192.168.1.254`.

netmask : DINT (*default -1 (255.255.255.255)*)
The subnet mask that, when combined with the destination IP address, determines the subnet which the route applies to.

gateway : DINT (*default -1*)
The IP address of the gateway to route the communication through.
When the gateway is all zeros, no gateway will be used and the destination IP range must be directly available on the network interface.
If set to -1, the gateway address of the network interface is used automatically.
If the gateway to the destination is on a different subnet than the directly attached network, a route must first be created to that gateway.

metric : INT (0..32000, *default 0*)
The metric of the route, which determines which route to use.
Lower values give higher priority.

Returns:

1	- Success.
0	- Not supported.
-1	- Illegal parameter.
-2	- Index is in use.

Declaration:

```
FUNCTION netRouteAdd;
VAR_INPUT
   index      : INT;
   ip         : DINT;
   netmask    : DINT := -1; // Mask : 255.255.255.255
   gateway    : DINT := -1;
   metric     : INT := 0;
   iface      : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   _route_     : netRouteGet;
END_VAR;

FUNCTION show_iface : STRING;
VAR_INPUT
   iface    : SINT;
END_VAR;

CASE iface OF
   1: show_iface := "Cellular";
   2: show_iface := "LAN1";
   3: show_iface := "LAN2";
   4: show_iface := "WLAN";
   5: show_iface := "AP";
ELSE
      show_iface := "<unknown>";
END_CASE;
END_FUNCTION;

FUNCTION route_list;
VAR
   str         : STRING;
   tmp         : STRING;
   i           : INT;
END_VAR;

   // Show header
   DebugMsg(message := " Destination      Gateway          Subnet mask
Metric  Iface  Status");

   // Iterate routes
   FOR i := 1 TO 100 DO
      // Get route
      _route_(index := i);
      IF _route_.status < 1 THEN EXIT; END_IF;

      // Build output
      str := " ";
      tmp := sockIPToName(ip := _route_.ip);
      WHILE strLen(str := tmp) < 17 DO tmp := tmp + " "; END_WHILE;
      str := str + tmp;
      tmp := sockIPToName(ip := _route_.gateway);
      WHILE strLen(str := tmp) < 17 DO tmp := tmp + " "; END_WHILE;
      str := str + tmp;
      tmp := sockIPToName(ip := _route_.netmask);
      WHILE strLen(str := tmp) < 17 DO tmp := tmp + " "; END_WHILE;
      str := str + tmp;
      tmp := intToStr(v := _route_.metric);
      WHILE strLen(str := tmp) < 8 DO tmp := tmp + " "; END_WHILE;
      str := str + tmp;
      tmp := show_iface(iface := _route_.iface);
      WHILE strLen(str := tmp) < 7 DO tmp := tmp + " "; END_WHILE;
      str := str + tmp;
      str := str + intToStr(v := _route_.status);

      // Show
      DebugMsg(message := str);

   END_FOR;
```

```
END_FUNCTION;

PROGRAM example;
VAR
   rc    : INT;
END_VAR;

   // Add route
   rc := netRouteAdd(
                     index   := 1
                    ,iface   := 2
                    ,ip      := sockIPFromName(str := "10.0.0.0")
                    ,netmask := sockIPFromName(str := "255.0.0.0")
                    ,gateway := sockIPFromName(str := "192.168.1.1")
                    ,metric  := 100
                    );
   IF rc < 1 THEN
      DebugFmt(message := "  netRouteAdd=\1", v1 := rc);
      RETURN;
   END_IF;

   // List routes
   route_list();

   // Remove route
   rc := netRouteDelete(index := 1);
   IF rc < 1 THEN
      DebugFmt(message := "  netRouteDel=\1", v1 := rc);
      RETURN;
   END_IF;

BEGIN
END;
END_PROGRAM;
```

4.2.32.32 netRouteDelete (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.80.00

This function will delete a route previously set by the netRouteAdd function.

Input:

index : INT (1..25)
The index of the route.

Returns:

1 - Success.
0 - Not supported.
-1 - Illegal parameter.
-2 - Index is not used..

Declaration:

```
FUNCTION netRouteDelete;
VAR_INPUT
```

```
   index      : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   _route_     : netRouteGet;
END_VAR;

FUNCTION show_iface : STRING;
VAR_INPUT
   iface    : SINT;
END_VAR;

CASE iface OF
   1: show_iface := "Cellular";
   2: show_iface := "LAN1";
   3: show_iface := "LAN2";
   4: show_iface := "WLAN";
   5: show_iface := "AP";
ELSE
      show_iface := "<unknown>";
END_CASE;
END_FUNCTION;

FUNCTION route_list;
VAR
   str         : STRING;
   tmp         : STRING;
   i           : INT;
END_VAR;

   // Show header
   DebugMsg(message := " Destination      Gateway          Subnet mask
Metric  Iface  Status");

   // Iterate routes
   FOR i := 1 TO 100 DO
      // Get route
      _route_(index := i);
      IF _route_.status < 1 THEN EXIT; END_IF;

      // Build output
      str := " ";
      tmp := sockIPToName(ip := _route_.ip);
      WHILE strLen(str := tmp) < 17 DO tmp := tmp + " "; END_WHILE;
      str := str + tmp;
      tmp := sockIPToName(ip := _route_.gateway);
      WHILE strLen(str := tmp) < 17 DO tmp := tmp + " "; END_WHILE;
      str := str + tmp;
      tmp := sockIPToName(ip := _route_.netmask);
      WHILE strLen(str := tmp) < 17 DO tmp := tmp + " "; END_WHILE;
      str := str + tmp;
      tmp := intToStr(v := _route_.metric);
      WHILE strLen(str := tmp) < 8 DO tmp := tmp + " "; END_WHILE;
      str := str + tmp;
      tmp := show_iface(iface := _route_.iface);
      WHILE strLen(str := tmp) < 7 DO tmp := tmp + " "; END_WHILE;
      str := str + tmp;
      str := str + intToStr(v := _route_.status);
```

```
      // Show
      DebugMsg(message := str);

   END_FOR;
END_FUNCTION;

PROGRAM example;
VAR
   rc    : INT;
END_VAR;

   // Add route
   rc := netRouteAdd(
                     index   := 1
                    ,iface   := 2
                    ,ip      := sockIPFromName(str := "10.0.0.0")
                    ,netmask := sockIPFromName(str := "255.0.0.0")
                    ,gateway := sockIPFromName(str := "192.168.1.1")
                    ,metric  := 100
                    );
   IF rc < 1 THEN
      DebugFmt(message := "  netRouteAdd=\1", v1 := rc);
      RETURN;
   END_IF;

   // List routes
   route_list();

   // Remove route
   rc := netRouteDelete(index := 1);
   IF rc < 1 THEN
      DebugFmt(message := "  netRouteDel=\1", v1 := rc);
      RETURN;
   END_IF;

BEGIN
END;
END_PROGRAM;
```

4.2.32.33 netRouteGet (Functionblock)

Architecture: NX32L
Device support: All
Firmware version: 1.80.00

This function will get information about a route previously set by the netRouteAdd function.

Input:

index : INT (1..25)
The index of the route.

Output:

status : SINT

5 - The gateway is unreachable.
4 - The route is duplicated.
3 - The route is bad. Can be caused by a netmask that does not match the IP.
2 - The route is not used. Happens when the interface is not connected.
1 - The route is used.

0 - No route found.
-1 - Invalid parameter.

iface : SINT
The network interface to use for the route.

ip : DINT
The destination IP address.

netmask : DINT
The subnet mask.

gateway : DINT
The gateway IP address.

metric : INT
The metric of the route.

Declaration:

```
FUNCTION_BLOCK netRouteGet;
VAR_INPUT
   index       : INT;
END_VAR;
VAR_OUTPUT
   status      : SINT;
   ip          : DINT;
   netmask     : DINT;
   gateway     : DINT;
   metric      : INT;
   iface       : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   _route_      : netRouteGet;
END_VAR;

FUNCTION show_iface : STRING;
VAR_INPUT
   iface     : SINT;
END_VAR;

CASE iface OF
   1: show_iface := "Cellular";
   2: show_iface := "LAN1";
   3: show_iface := "LAN2";
   4: show_iface := "WLAN";
   5: show_iface := "AP";
ELSE
      show_iface := "<unknown>";
END_CASE;
END_FUNCTION;

FUNCTION route_list;
VAR
   str          : STRING;
   tmp          : STRING;
```

```
   i           : INT;
END_VAR;

   // Show header
   DebugMsg(message := " Destination      Gateway          Subnet mask
Metric  Iface  Status");

   // Iterate routes
   FOR i := 1 TO 100 DO
      // Get route
      _route_(index := i);
      IF _route_.status < 1 THEN EXIT; END_IF;

      // Build output
      str := " ";
      tmp := sockIPToName(ip := _route_.ip);
      WHILE strLen(str := tmp) < 17 DO tmp := tmp + " "; END_WHILE;
      str := str + tmp;
      tmp := sockIPToName(ip := _route_.gateway);
      WHILE strLen(str := tmp) < 17 DO tmp := tmp + " "; END_WHILE;
      str := str + tmp;
      tmp := sockIPToName(ip := _route_.netmask);
      WHILE strLen(str := tmp) < 17 DO tmp := tmp + " "; END_WHILE;
      str := str + tmp;
      tmp := intToStr(v := _route_.metric);
      WHILE strLen(str := tmp) < 8 DO tmp := tmp + " "; END_WHILE;
      str := str + tmp;
      tmp := show_iface(iface := _route_.iface);
      WHILE strLen(str := tmp) < 7 DO tmp := tmp + " "; END_WHILE;
      str := str + tmp;
      str := str + intToStr(v := _route_.status);

      // Show
      DebugMsg(message := str);

   END_FOR;
END_FUNCTION;

PROGRAM example;
VAR
   rc    : INT;
END_VAR;

   // Add route
   rc := netRouteAdd(
                     index   := 1
                    ,iface   := 2
                    ,ip      := sockIPFromName(str := "10.0.0.0")
                    ,netmask := sockIPFromName(str := "255.0.0.0")
                    ,gateway := sockIPFromName(str := "192.168.1.1")
                    ,metric  := 100
                    );
   IF rc < 1 THEN
      DebugFmt(message := "  netRouteAdd=\1", v1 := rc);
      RETURN;
   END_IF;

   // List routes
   route_list();

   // Remove route
   rc := netRouteDelete(index := 1);
   IF rc < 1 THEN
      DebugFmt(message := "  netRouteDel=\1", v1 := rc);
```

```
      RETURN;
   END_IF;

BEGIN
END;
END_PROGRAM;
```

4.2.33 nmp: Navigation and Messaging Platform

4.2.33.1 nmp: Navigation and Messaging Platform

The NMP, Navigation and Messaging Platform, is a powerful platform that provides the combined functionality of a navigation device with fleet management and advanced messaging support.

The NMP is compatible with the Navigation and MDT functions. Existing programs that use these functions will normally not need to be changed. Only in the initialization phase is it required that the navigation part of the NMP is working before the MDT part is initialized. To start the navigation interface, simply call navOpen the same way as with a non-NMP device. Additionally, it may be required to call nmpPower depending on the hardware configuration used.
To use the MDT functionality, call mdtOpen and then mdtPower when the navigation interface is connected (navWaitEvent #132). Alternatively a loop checking the return code of mdtPower can be used.

The connection between the NMP device and the RTCU device is established through a serial port. In order to use the route-related functions, it is required that gpsFix is called repeatedly, e.g. from a separate thread, to feed the NMP device with GPS data.

To determine if the device is an NMP device and therefore supports the NMP functions as well as the navigation functions, call the nmpPresent function which returns "true" if an NMP is connected and "false" if an ordinary navigation device or no device is connected.
An attempt to use the NMP functionality on a Garmin device will fail with an error code specifying that the feature is unsupported.

It is possible for the user to define multiple buttons at the bottom of the NMP screen. These user-configurable buttons can be controlled with various attributes such as: size, color, text, visibility, and flashing. An event is generated when any of these buttons are pressed by the user and as such allows for a high degree of customization and user control.

Some of the functions are not supported on all NMP devices. To determine the type of device the application is running on, use nmpGetHardwareId.

The following functions are used to access the NMP, in addition to the navigation and MDT functions:

- nmpPresent — Queries whether an NMP device is present.
- nmpUpdate — Transfers an update to the NMP device and tries to install it.
- nmpUpdateProgressReceive — Reads an update transfer progress report.
- nmpPower — Powers the NMP on and off.
- nmpStopPopup — Makes a destination pop up when added.
- nmpStopClose — Closes a stop dialog.
- nmpPlaySound — Plays a sound on the NMP device.
- nmpShowDialog — Shows a dialog with buttons.
- nmpRemoveDialog — Removes a dialog without user interaction.
- nmpDialogClickReceive — Reads the clicked button in the dialog.
- nmpGetHardwareID — Reads the type of NMP device that is connected.
- nmpHideMenus — Hides/shows menu bars.
- nmpHomePos — Retrieves the coordinates of the home location.
- nmpSetVolume — Controls the volume of the device.
- nmpLCDBrightness — Sets the brightness of the backlight.
- nmpGetIdleTime — Retrieves the time since the last user interaction.
- nmpSetLED — Controls the status LEDs on the device.
- nmpHardwareButtonsEnable — Enables the use of the hardware buttons on the device.

- nmpHardwareButtonPressedReceive — Rreads the ID of the pressed hardware button.
- nmpCameraOpen — Opens the connection to a camera and shows what it sees.
- nmpCameraClose — Closes the connection to a camera.
- nmpRGBToDint — Converts 3 RGB color components into a DINT color.
- nmpCardID — Retrieves the serial number of the SD-CARD.
- nmpTouchscreenCal — Starts the GUI for calibrating the touchscreen.

Configurable screen buttons:

- nmpButtonsDefine — Creates a number of buttons.
- nmpButtonCreate — Initializes a button.
- nmpButtonEnable — Enables a button.
- nmpButtonVisible — Sets the visibility of a button.
- nmpButtonWidth — Sets the width of a button.
- nmpButtonColor — Sets the background color of a button.
- nmpButtonText — Sets the text of a button.
- nmpButtonFlash — Controls the flashing of the button.
- nmpButtonConfirm — Makes the button require confirmation when pressed.
- nmpButtonPressedReceive — Reads the ID of pressed buttons.

The NMP interface has the following limitations:

- navFix is not implemented as the NMP requires an RTCU device with a GPS receiver while navFix is for RTCU devices without a GPS receiver.
- The GPI functions are not supported as the NMP cannot parse GPI files. Instead, the POIs will have to be added manually by using the navWaypointSet function.
- Only API level 1 and 2 functions are supported.

4.2.33.2 nmpPresent (Function)

```
Architecture:       X32 / NX32 / NX32L
Device support:     MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-
                    200, NX-400, NX-900, LX2
Firmware version:   2.62 / 1.00.00
Nav. API level:     2
```

This function will determine if an NMP device is present.

Input:

None.

Returns: BOOL

TRUE: If an NMP device is present.
FALSE: If an NMP device is not present.

Declaration:

```
FUNCTION nmpPresent : BOOL;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NMPExample;
```

```
BEGIN
   // Check if navigation device is present
   IF nmpPresent() THEN
      DebugMsg(message := "NMP device present");
      ...
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.33.3 nmpUpdate (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-200, NX-400, NX-900, LX2
Firmware version: 2.62 / 1.00.00
Nav. API level: 2

This function is used to update the NMP device when new versions of the software are released. When called, it will start transferring an update file from the file system of the RTCU device to the NMP, and when the transfer is done (see navWaitEvent for event #10), the user is notified of the update. By using the "force" parameter, it can be controlled whether the update should be installed immediately or postponed until its next restart to avoid losing access to the NMP while driving.
During the update process, the RTCU device cannot communicate with the NMP device.
An "NMP Update Progress" event is raised when there are changes to the progress and/or status of the update. See also navWaitEvent for event # 10.

If the user declines an update or it fails, e.g. due to being shut down, the NMP navigation device will try to install it when the system is restarted.

The file used for the upgrade is usually supplied by Logic IO or its partners.

Input:

filename : STRING
The name and path of the file to transfer.

force : BOOL
Determines whether the user is asked if the update should be installed or if it should be forced.

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-3	- File not found.
-4	- The navigation device rejected the command.
-11	- This is not supported by the device (the device is not an NMP device).
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION nmpUpdate : INT;
VAR_INPUT
   filename : STRING;
   force    : BOOL := FALSE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NMPExample;
VAR
   rc : INT;
END_VAR;
BEGIN
   ...
   // Start transfer of update
   rc := nmpUpdate(filename := "A:\update.zip", force  := FALSE);
   IF rc <> 0 THEN
      DebugFmt(message := "Error: nmpUpdate=\1", v1 := rc);
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.33.4 nmpPower (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-200, NX-400, NX-900, LX2
Firmware version: 2.64 / 1.00.00
Nav. API level: 2

This function controls the power of a connected NMP navigation device. By using this function, the program can turn the NMP on and off to save power or reset the device.
This must be called after navOpen as navOpen determines if the serial port is in use.

Depending on the RTCU device in use, the power to the NMP navigation device is controlled by the following output signal:

RTCU type	Signal used.
RTCU MX2i pro/pro+	RS232 DTR signal.
RTCU MX2 turbo	RS232 DTR signal.
RTCU CX1 series	Digital output 2.
RTCU SX1	Digital output 2.

When using the nmpPower function on the RTCU CX1 series or on the RTCU SX1, the digital output 2 is reserved for this purpose and cannot be used from the VPL application.

Usually, the power is controlled by a dedicated interface box connected to the NMP device in question. Please install according to the hardware documentation.

Input:

power : BOOL (*Default: ON*)
ON: Turns power to NMP on.
OFF: Turns power to NMP off.

Returns: INT

0 - Success.
1 - Failure.

Declaration:

```
FUNCTION nmpPower : INT;
VAR_INPUT
   power : BOOL := ON;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NMPExample;
VAR
   rc : INT;
END_VAR;

   DebugMsg(message := "Initializing NMP...");
   navOpen();
   rc := nmpPower();
   IF rc <> 0 THEN
      DebugFmt(message := "Error: nmpPower=\1", v1 := rc);
   END_IF;

   ...

   rc := nmpPower(power := OFF);
   // Close the navigation interface
   navClose();
   ...
END_PROGRAM;
```

4.2.33.5 nmpUpdateProgressReceive (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-200, NX-400, NX-900, LX2
Firmware version: 2.62 / 1.00.00
Nav. API level: 2

This function block returns information about the status and progress of an NMP update as initiated with nmpUpdate.
Note that the progress is only valid after an "NMP Update Progress" event has been raised; the event will be removed when this function block is called.
The status must be 2 before the update can be considered complete, and the user can still cancel the update if he wants to.

Input:

None.

Output:

status : INT
The status of the transfer

- 2 - Update is started on NMP. If update is not forced, the user is allowed to cancel it.
- 1 - Transfer in progress.
- 0 - Transfer completed.
- -2 - Error communicating with navigation device.
- -3 - Error reading file.
- -4 - The navigation device rejected the transfer.

progress : SINT
The progress of the transfer in percent.

ready : BOOL
TRUE: If data is ready.
FALSE: If data is not ready.

Declaration:

```
FUNCTION_BLOCK nmpUpdateProgressReceive;
VAR_OUTPUT
   status    : INT;
   progress  : SINT;
   ready     : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   updRcv : nmpUpdateProgressReceive;
END_VAR;

FUNCTION ReadUpdProgress;
   updRcv();
   IF updRcv.ready THEN
      DebugMsg(message := "*** Update progress received ***");
      DebugFmt(message := "-status=\1", v1 := updRcv.status);
      DebugFmt(message := "-progress=\1", v1 := updRcv.progress);
   END_IF;
END_FUNCTION;

THREAD_BLOCK navMonitor;
VAR
   event : INT := 0;
END_VAR;

WHILE event <> -1 DO
   event := navWaitEvent(timeout := -1);
   CASE event OF
      ...
      10: ReadUpdProgress();
      ...
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;
```

4.2.33.6 nmpStopPopup (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-200, NX-400, NX-900, LX2
Firmware version: 2.80 / 1.00.00
Nav. API level: 2

This function causes a destination to pop up when it is set instead of just being added to the stop list in the background. This function must be called before navStopSet to prepare it. If the same ID is used multiple times, nmpStopPopup must be called before each call to navStopSet. If a destination is already shown, the new destination will be shown instead.
To close a stop shown this way, nmpStopClose can be used.

Input:

ID : INT
Unique ID to identify the destination when it is added.

popup : BOOL
TRUE: Destination will pop up.
FALSE: Destination will not pop up.

Returns: INT

0 - Success.
-1 - Navigation interface is not open.
-2 - Error communicating with navigation device.
-4 - The navigation device rejected the command.
-11 - This is not supported by the device (e.g. the device is not an NMP device).
-12 - Navigation interface is busy.

Declaration:

```
FUNCTION nmpStopPopup : INT;
VAR_INPUT
   id    : INT;
   popup : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   nmpStopPopup(id := 61, popup := TRUE);
   // Add a destination to the route
   navStopSet(id := 61, latitude := 666377598, longitude := 117525954, text :=
"Logic IO - main office");
   ...
END;
END_PROGRAM;
```

4.2.33.7 nmpStopClose (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-200, NX-400, NX-900, LX2
Firmware version: 2.84 / 1.00.00
Nav. API level: 2

This function causes a stop that is shown to be closed. If called after nmpStopPopup, but before calling navStopSet, the timeout will start when the stop is shown, otherwise it will start immediately.
Can close both stops that are opened with nmpStopPopup and stops that are manually opened.

Input:

ID : INT
Unique ID that identifies the destination to close.

timeout : INT (default: 0)
Time until the stop should be closed in seconds. 0 will close the stop immediately.

Returns: INT

0 - Success.

-1 - Navigation interface is not opened.
-2 - Error communicating with navigation device.
-4 - The navigation device rejected the command.
-8 - Illegal timeout value
-11 - This is not supported by the device (e.g. the device is not an NMP device).
-12 - Navigation interface is busy.

Declaration:

```
FUNCTION nmpStopClose : INT;
VAR_INPUT
   id      : INT;
   timeout : INT := 0;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NavigationExample;

BEGIN
   ...
   nmpStopPopup(id := 61, popup := TRUE);
   // Set stop to close after a minute
   nmpStopClose(id := 61, timeout := 60);
   // Add a destination to the route
   navStopSet(id := 61, latitude := 666377598, longitude := 117525954, text :=
"Logic IO - main office");
   ...
END;
END_PROGRAM;
```

4.2.33.8 nmpPlaySound (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-200, NX-400, NX-900, LX2
Firmware version: 2.80 / 1.00.00
Nav. API level: 2

This function will play a sound on the NMP.
The file to play must be a .wav file (Waveform Audio File Format) and must be located on the NMP device - e.g on the SD-CARD.
Only one sound can be played at the same time.

Input:

loop : BOOL (*Default: FALSE*)
TRUE: The sound will loop. To stop the loop, call nmpPlaySound with an empty file name.
FALSE: The sound will be played once.

filename : STRING
The path to the sound file to play. Use an empty string to stop playing a looping sound. The string must be less than 200 characters. If the file can not be found, the default sound will be played.

Returns: INT

0 - Success.
-1 - Navigation interface is not open.
-2 - Error communicating with navigation device.

-4 - The navigation device rejected the command.
-8 - File name too long.
-11 - This is not supported by the device (e.g. the device is not an NMP device).
-12 - Navigation interface is busy.

Declaration:

```
FUNCTION nmpPlaySound : INT;
VAR_INPUT
   loop     : BOOL := FALSE;
   filename : STRING := "";
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
BEGIN
   ...
   //Play sound for 5 seconds
   nmpPlaySound(loop := TRUE, filename := "\SD Card\NMP\msg.wav");
   sleep(delay := 5000);
   nmpPlaySound();
   ...
END;
END_PROGRAM;
```

4.2.33.9 nmpSetVolume (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-200, NX-400, NX-900, LX2
Firmware version: 2.84 / 1.00.00
Nav. API level: 2

This function will mute or un-mute the PNM device.

Input:

volume : SINT (-1..100)
0 - The volume will be muted
-1 - The volume level will be restored.
1..100 - Currently not used but reserved for setting volume level directly.

Returns: INT

0 - Success.
-1 - Navigation interface is not open.
-2 - Error communicating with navigation device.
-4 - Failed to set volume.
-8 - Volume out of range.
-11 - This is not supported by the device (e.g. the device is not an NMP device).
-12 - Navigation interface is busy.

Declaration:

```
FUNCTION nmpSetVolume : INT;
VAR_INPUT
   volume   : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
BEGIN
   ...
   //Mute sound
   nmpSetVolume(volume := 0);
   ...
END;
END_PROGRAM;
```

4.2.33.10 nmpLCDBrightness (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-200, NX-400, NX-900, LX2
Firmware version: 2.84 / 1.00.00
Nav. API level: 2

This function will set the brightness of the LCD of the NMP device.

Screen saver functionality

Starting with NMP v3.00, brightness level 0 can be used as a screen saver, to reduce the risk of burned in images.
If the hardware buttons are enabled, setting the brightness level to 0 will both turn off the backlight and blank the entire screen. Any press on the touchscreen in this mode will be sent as a hardware button press, to allow the user to wake the NMP device manually. When the brightness is set above 0, the screen is restored.
To determine when to turn off the screen, the nmpGetIdleTime function can be used, in combination with additional logic, to avoid turning off the screen while navigating.

Input:

brightness : SINT (0..100)
The brightness of the LCD backlight.
The backlight is turned off at level 0 and the brightest at level 100.
All devices do not necessarily support all levels but will be mapped to the most correct levels.

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-4	- Failed to set brightness.
-8	- Brightness out of range.
-11	- This is not supported by the device (e.g. the device is not an NMP device).
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION nmpLCDBrightness : INT;
VAR_INPUT
   brightness : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
BEGIN
   ...
   //Turn off LCD
   nmpLCDBrightness(brightness := 0);
   ...
END;
END_PROGRAM;
```

4.2.33.11 nmpGetIdleTime (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-200, NX-400, NX-900, LX2
Firmware version: 4.50 / 1.00.00
Nav. API level: 2

This function will return the number of minutes it has been since the last user interaction with the NMP device (Press on the touchscreen or on the hardware buttons).
This can be used e.g. to implement screen saver functionality using nmpLcdBrightness.

Input:
None.

Returns: INT
0..32	- The number of minutes since the last interaction.
767	
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-4	- The navigation device rejected the command.
-11	- This is not supported by the device (e.g. the device is not an NMP device).
-12	- Navigation interface is busy.

Declaration:
```
FUNCTION nmpGetIdleTime : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
BEGIN
   ...
   //Turn off LCD if it has been unused for 10 minutes
   IF nmpGetIdleTime() >= 10 THEN
      nmpLCDBrightness(brightness := 0);
   END_IF;

   ...
END;
END_PROGRAM;
```

4.2.33.12 nmpSetLED (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-200, NX-400, NX-900, LX2
Firmware version: 2.84 / 1.00.00
Nav. API level: 2

This function will set the state of an LED on the PNM device.

Input:

led : SINT(1..4)
The ID of the LED to set.
- 1 - First indicator.
- 2 - Second indicator.
- 3 - Third indicator.
- 10 - Button backlight.

state : SINT (0..4)
- 0 - Turns LED off
- 1 - Turns LED on
- 2..4 - Blinks LED by using a different pattern.

Returns: INT

- 0 - Success.
- -1 - Navigation interface is not open.
- -2 - Error communicating with navigation device.
- -4 - Failed to set LED.
- -8 - Wrong LED ID or state.
- -11 - This is not supported by the device (e.g. the device is not an NMP device).
- -12 - Navigation interface is busy.

Declaration:

```
FUNCTION nmpSetLED : INT;
VAR_INPUT
   led   : SINT;
   state : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
BEGIN
   ...
   //Turn on the first LED
   nmpSetLED(led := 1, state := 1);
   ...
END;
END_PROGRAM;
```

4.2.33.13 nmpShowDialog (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-200, NX-400, NX-900, LX2
Firmware version: 2.80 / 1.00.00
Nav. API level: 2

This function will show a dialog with a number of buttons on the NMP. It can be used as a supplement to navMessageSend, as these dialogs are not saved and are shown immediately. The response is returned in nmpDialogClickReceive.

Input:

type : SINT
Determines the type of dialog and buttons to show.

0 - OK
1 - OK, Cancel
2 - Abort, Retry, Ignore
3 - Yes, No, Cancel
4 - Yes, No
5 - Retry, Cancel

caption : STRING
The text to show at the top of the dialog. (Max. 39 characters)

message : STRING
The text to show in the dialog. (Max. 199 characters)

Returns: INT

0 - Success.
-1 - Navigation interface is not opened.
-2 - Error communicating with navigation device.
-4 - A dialog is already shown. Close the dialog with nmpRemoveDialog and try again.
-8 - Caption or message too long or buttons out of range.
-11 - This is not supported by the device (e.g. the device is not an NMP device).
-12 - Navigation interface is busy.

Declaration:

```
FUNCTION nmpShowDialog : INT;
VAR_INPUT
   type    : SINT;
   caption : STRING;
   message : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
BEGIN
   ...
   //Show dialog
   nmpShowDialog(type := 4, caption := "Drive?", message := "Do you want to
drive?");
   ...
```

```
END;
END_PROGRAM;
```

4.2.33.14 nmpRemoveDialog (Function)

```
Architecture:       X32 / NX32 / NX32L
Device support:     MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-
                    200, NX-400, NX-900, LX2
Firmware version:   2.80 / 1.00.00
Nav. API level:     2
```

This function will remove the current dialog from the NMP without triggering an event.

Input:
None.

Returns: **INT**

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-4	- Dialog could not be removed as no dialog is currently shown.
-11	- This is not supported by the device (e.g. the device is not an NMP device).
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION nmpRemoveDialog : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc : INT;
BEGIN
   ...
   // Remove any old dialogs
   rc := nmpRemoveDialog();
   IF rc = 0 OR rc = -4 THEN
      //Show dialog
      nmpShowDialog(type := 4, caption := "Drive?", message := "Do you want to
drive?");
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.33.15 nmpDialogClickReceive (Functionblock)

```
Architecture:       X32 / NX32 / NX32L
Device support:     MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-
                    200, NX-400, NX-900, LX2
Firmware version:   2.80 / 1.00.00
Nav. API level:     2
```

The function block returns the ID of a clicked button from a dialog on an NMP.

Note that the ID is only valid after an "NMP Dialog Click" event is raised; the event will be removed when this function block is called.

Input:

None.

Output:

id : INT

The ID of the clicked button.

2 - OK.
3 - Cancel.
4 - Abort.
5 - Retry.
6 - Ignore.
7 - Yes.
8 - No.

ready : BOOL

TRUE: If data is ready.
FALSE: If data is not ready.

Declaration:

```
FUNCTION_BLOCK nmpDialogClickReceive;
VAR_OUTPUT
   id    : INT;
   ready : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   dialogClick : nmpDialogClickReceive;
END_VAR;

FUNCTION DialogClicked;
   dialogClick();
   IF dialogClick.ready THEN
      DebugMsg(message := "*** Dialog Clicked ***");
      DebugFmt(message := "-button=\1", v1 := dialogClick.id);

   END_IF;
END_FUNCTION;

THREAD_BLOCK navMonitor;
VAR
   event : INT := 0;
END_VAR;

WHILE event <> -1 DO
   event := navWaitEvent(timeout := -1);
   CASE event OF
      ...
      12: DialogClicked();
      ...
   END_CASE;
```

```
END_WHILE;
END_THREAD_BLOCK;
```

4.2.33.116 nmpGetHardwareID (Function)

```
Architecture:       X32 / NX32 / NX32L
Device support:     MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-
                    200, NX-400, NX-900, LX2
Firmware version:   2.84 / 1.00.00
Nav. API level:     2
```

This function will return the type of device the NMP is running on.

Input:

None.

Returns: **INT**

1	PNM-100.
2	PNM-200 Series.
0x100 to 0x10F	The 16 values from 0x100 to 0x10F are reserved for custom devices. The exact value depends on the configuration of the device.
0	- Navigation device is not present or no hardware ID is available.
-1	- Navigation interface is not open.
-11	- This is not supported by the device (the device is not an NMP device).

Declaration:

```
FUNCTION nmpGetHardwareID : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM NMPExample;

BEGIN
   ...   // Read device type
   DebugFmt(message := "NMP device type: \1", v1 := nmpGetHardwareID());
   ...
END;
END_PROGRAM;
```

4.2.33.117 nmpHideMenus (Function)

```
Architecture:       X32 / NX32 / NX32L
Device support:     MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-
                    200, NX-400, NX-900, LX2
Firmware version:   2.84 / 1.00.00
Nav. API level:     2
```

This function will show or hide menus on the NMP.
This can be used to make the screen area available to the map larger.

Input:

menus : DINT
This is a mask that determines the visible menus. Each bit represents a menu, where 1 is visible, and 0 is hidden.
The following values can be used to control the menus:
0 - All menus hidden.
-1 - All menus visible.
-2 - Menu bar hidden.
-3 - Custom buttons hidden.
-4 - Menu and custom buttons hidden.

Returns: INT
0 - Success.
-1 - Navigation interface is not open.
-2 - Error communicating with navigation device.
-4 - NMP rejected command.
-11 - This is not supported by the device (e.g. the device is not an NMP device).
-12 - Navigation interface is busy.

Declaration:
```
FUNCTION nmpHideMenus : INT;
VAR_INPUT
   menus    : DINT;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;
BEGIN
   ...
   //Hide menu bar
   nmpHideMenus(menus := -2);
   ...
END;
END_PROGRAM;
```

4.2.33.18 nmpRGBToDint (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-200, NX-400, NX-900, LX2
Firmware version: 2.84 / 1.00.00
Nav. API level: 2

This function will convert the provided 3-color components to a single DINT that contains the resulting color.
The format of the resulting color as a hex value is 16#00_rr_gg_bb, where "rr", "gg", and "bb" are the hex representation of the three components.

Input:
r : INT (0..255)
The red component of the color.

g : INT (0..255)
The green component of the color.

b : INT (0..255)
The blue component of the color.

Returns: DINT
The color.

Declaration:
```
FUNCTION nmpRGBToDint : DINT;
VAR_INPUT
    r   : INT;
    g   : INT;
    b   : INT;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;
VAR
 color : DINT;
END_VAR;
BEGIN
   ...
   // Set color
    color := nmpRGBToDint(r := 8, g:= 146, b := 206);
   ...
END;
END_PROGRAM;
```

4.2.33.119 nmpHardwareButtonPressedReceive (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-200, NX-400, NX-900, LX2
Firmware version: 2.84 / 1.00.00
Nav. API level: 2

This function block returns the ID of a pressed hardware button on an NMP.
To enable this event, call nmpHardwareButtonsEnable.
Note that the ID is only valid after a "Hardware Button Pressed" event is raised; the event will be removed when this function block is called.

If hardware buttons are enabled, and the lcd backlight is set to level 0, any press on the touchscreen will be sent as button id 100, useful to e.g. wake the device from a screen saver mode.

Input:
None.

Output:
id : INT
The ID of the pressed button. Possible values:
- 1 - "1" button.
- 2 - "2" button.
- 3 - "3" button.

4 - "4" button.
10 - "SET" button.
11 - "Power" button.
100 - Touchscreen press (nmpLcdBrightness level 0)

ready : BOOL
TRUE: If data is ready.
FALSE: If data is not ready.

Declaration:

```
FUNCTION_BLOCK nmpHardwareButtonPressedReceive;
VAR_OUTPUT
   id    : INT;
   ready : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
VAR
   hwButtonPress : nmpHardwareButtonPressedReceive;
END_VAR;

FUNCTION HwButtonPressed;
   hwButtonPress();
   IF hwButtonPress.ready THEN
      DebugMsg(message := "*** Hardware Button Pressed ***");
      DebugFmt(message := "-button=\1", v1 := hwButtonPress.id);
   END_IF;
END_FUNCTION;

THREAD_BLOCK navMonitor;
VAR
   event : INT := 0;
END_VAR;

WHILE event <> -1 DO
   event := navWaitEvent(timeout := -1);
   CASE event OF
      ...
      13: HwButtonPressed();
      ...
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;
```

4.2.33.20 nmpHardwareButtonsEnable (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-200, NX-400, NX-900, LX2
Firmware version: 2.84 / 1.00.00
Nav. API level: 2

This function will enable or disable the hardware buttons on the NMP.

Input:

enable : BOOL (default : true)
This determines if the buttons are enabled and can raise the "Hardware Button Pressed" event.

led_state : SINT (default : -1)
Determines the state of the keyboard LED.
-1 - Follow-enabled state (turned on when keys are enabled, turned off when they are not).
0 - Turns LED off.
1 - Turns LED on.
2..4 - Blinks LED by using a different pattern.

Returns: INT

0 - Success.
-1 - Navigation interface is not open.
-2 - Error communicating with navigation device.
-4 - Failed to enable buttons.
-11 - This is not supported by the device (the device is not an NMP device).
-12 - Navigation interface is busy.

Declaration:

```
FUNCTION nmpHardwareButtonsEnable : INT;
VAR_INPUT
   enable    : BOOL := TRUE;
   led_state : SINT := -1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc : INT;
END_VAR;
BEGIN
   ...
   // Enable buttons and turn on LED
   rc := nmpHardwareButtonsEnable();
   IF rc <> 0 THEN
      DebugFmt(message := "Error: nmpHardwareButtonsEnable=\1", v1 := rc);
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.33.21 nmpCameraOpen (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-200, NX-400, NX-900, LX2
Firmware version: 2.84 / 1.00.00
Nav. API level: 2

This function will open the connection to connected cameras and show the output on the entire screen.
A blank screen will be shown if no output is available (e.g. camera is not connected).

Input:

id : SINT (default : 1)
ID of camera to show: 1 or 2.

Returns: **INT**

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-4	- Failed to open camera. It might already be open.
-8	- Unknown camera ID.
-11	- This is not supported by the device (e.g. the device is not an NMP device).
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION nmpCameraOpen : INT;
VAR_INPUT
   id    : SINT := 1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
BEGIN
   ...
   //Show camera
   nmpCameraOpen(id := 1);
   ...
END;
END_PROGRAM;
```

4.2.33.22 nmpCameraClose (Function)

```
Architecture:       X32 / NX32 / NX32L
Device support:     MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-
                    200, NX-400, NX-900, LX2
Firmware version:   2.84 / 1.00.00
Nav. API level:     2
```

This function will close the full screen camera output and close the connection to a camera.

Input:

id : SINT (default : 1)
ID of camera to close.

Returns: **INT**

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-4	- Failed to open camera. It might already be closed.
-8	- Unknown camera ID.
-11	- This is not supported by the device (e.g. the device is not an NMP device).
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION nmpCameraClose : INT;
VAR_INPUT
   id    : SINT := 1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
BEGIN
   ...
   //Close camera
   nmpCameraClose(id := 1);
   ...
END;
END_PROGRAM;
```

4.2.33.23 nmpHomePos (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-200, NX-400, NX-900, LX2
Firmware version: 2.80 / 1.00.00
Nav. API level: 2

This function will read the coordinates of the home position specified in the menu of Sygic. This can be used for driving to home without having to define the location on the RTCU.

Input:

None.

Output:

lat : DINT
Latitude of the home position in semicircle format.

lon : DINT
Longitude of the home position in semicircle format.

Returns: INT

0	- Success.
-1	- Navigation interface is not opened.
-2	- Error communicating with navigation device.
-4	- Failed to retrieve position.
-11	- This is not supported by the device (e.g. the device is not an NMP device).
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION nmpHomePos : INT;
VAR_INPUT
   lat   : ACCESS INT;
   lon   : ACCESS INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   lat : DINT;
   lon : DINT;
END_VAR;
BEGIN
```

```
   ...
   //Show dialog
   IF nmpHomePos(lat := lat, lon := lon) = 0 THEN
      navStopSet(id := 60, latitude := lat, longitude := lon, text := "Home");
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.33.24 nmpCardID (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-200, NX-400, NX-900, LX2
Firmware version: 3.15 / 1.00.00
Nav. API level: 2

This function will read the ID of the SD-CARD inserted in the PNM, which is used for activating map licenses.

Input:

None.

Output:

serial : STRING
The serial number of the SD card.

Returns: INT

0	- Success.
-1	- Navigation interface is not opened.
-2	- Error communicating with navigation device.
-4	- Failed to retrieve serial number.
-11	- This is not supported by the device (e.g. the device is not an NMP device).
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION nmpCardID : INT;
VAR_INPUT
   serial : ACCESS STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   str : STRING;
END_VAR;
BEGIN
   ...
   IF nmpCardID(serial := str) = 0 THEN
      DebugMsg(message := "ID: " + str);
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.33.25 nmpTouchscreenCal (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-200, NX-400, NX-900, LX2
Firmware version: 3.15 / 1.00.00
Nav. API level: 2

This function will show the GUI to calibrate the touch screen. To perform the calibration, follow the instructions on the screen by clicking the target.
If the target reappears in the center after calibration, it indicates that the values were too imprecise.
Note: The calibration GUI does not close until the calibration has been completed by the user.

Input:

None.

Returns: INT

0	- Success.
-1	- Navigation interface is not opened.
-2	- Error communicating with navigation device.
-4	- Calibration is already in progress.
-11	- This is not supported by the device (e.g. the device is not an NMP device).
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION nmpTouchscreenCal : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
BEGIN
   ...
   // Calibrate touchscreen
   nmpTouchscreenCal();
   ...
END;
END_PROGRAM;
```

4.2.33.26 nmpButtonsDefine (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-200, NX-400, NX-900, LX2
Firmware version: 2.64 / 1.00.00
Nav. API level: 2

This function will define a number of configurable buttons on the NMP with the IDs 1.. count. It is required to call this method before any other nmpButton calls.
Note that the buttons take up space that would otherwise be used for the navigation display.

Input:

count : INT (0..10)
The number of buttons to define. Using 0 will remove all the buttons and return the used space to the navigation display.

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-4	- The navigation device rejected the command.
-8	- Illegal value of count.
-11	- This is not supported by the device (the device is not an NMP device).
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION nmpButtonsDefine : INT;
VAR_INPUT
   count : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc : INT;
END_VAR;
BEGIN
   ...
   // Define 3 buttons
   rc := nmpButtonsDefine(count := 3);
   IF rc <> 0 THEN
      DebugFmt(message := "Error: nmpButtonsDefine=\1", v1 := rc);
   END_IF;
   ...
END;
END_PROGRAM;
```

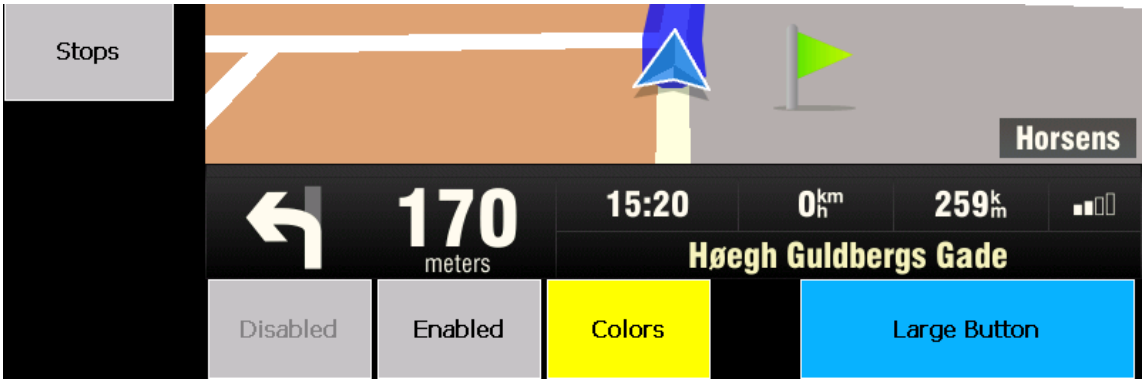
4.2.33.27 nmpButtonCreate (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-200, NX-400, NX-900, LX2
Firmware version: 2.64 / 1.00.00
Nav. API level: 2

This function will initialize a configurable button on the NMP.
It is not necessary to call this function as all the settings can be set with the functions for the individual properties, but this makes it possible to set all the visible properties at once. Using this function will avoid showing the user the intermediate states.

The following parameters was used to create the buttons on the following image.

ID	text	weight	visible	enabled	color
1	Disabled	2	True	False	16#C0_C0_C0
2	Enabled	2	True	True	16#C0_C0_C0
3	Colors	2	True	True	16#FF_FF_00
4	Small Button	1	False	False	16#C0_C0_C0
5	Large Button	4	True	True	16#0F_B3_FF



Input:

id : INT
The ID of the button - between 1 and the button count defined in nmpButtonsDefine.

weight : INT (default : 1)
Determines the width of the button relative to the total weight of all the buttons.
A button with weight 2 is twice as wide as a button with weight 1. If the total weight of the buttons is 4, a button with weight 2 will have a width 50% of this.

max_width : INT (0..100, default: 0)
Determines the maximum width of the button as an integer between 1 and 100, where 100 is the entire width of the panel.To have no maximum value, use 0.
This is used to prevent a button from becoming too wide when only a few buttons are defined by limiting the width to e.g. 50 % of the total width.

visible : BOOL (default: FALSE)
The visibility of the button.

enable : BOOL (default: FALSE)
Determines if the button is enabled and can raise the "NMP Button Pressed" event. See also navWaitEvent for event # 11.

color: DINT (-1, 16#00_00_00..16#FF_FF_FF, default: -1)
This determines the color of the button. If -1, the theme color will be used.
Instead of specifying the color directly, the nmpRGBToDint function can be used.

night_color: DINT (-1, 16#00_00_00..16#FF_FF_FF, default: -1)
This determines the color of the button to use when in the night theme. If -1, the day color will be used unless it is set to use the theme, in which case the color of the night theme will be used.
Instead of specifying the color directly, the nmpRGBToDint function can be used.

text : STRING
The text of the button. Max. 49 characters.

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-4	- The navigation device rejected the command.
-8	- Invalid parameters.
-11	- This is not supported by the device (the device is not an NMP device).
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION nmpButtonCreate : INT;
VAR_INPUT
   id          : INT;
   weight      : INT := 1;
   visible     : BOOL := FALSE;
   enable      : BOOL := FALSE;
   max_width   : INT := 0;
   text        : STRING;
   color       : SINT := -1;
   night_color : SINT := -1;

END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc : INT;
END_VAR;
BEGIN
   ...
   // Create button with text "Click Me" and a bright yellow backgound for the
day, and a dark yellow color for the night.
   rc := nmpButtonCreate(id := 1, text := "Click Me", visible := TRUE, enable
:= TRUE,
color:= 16#FF_FF_00, night_color:= 16#C8_C8_00);
   IF rc <> 0 THEN
      DebugFmt(message := "Error: nmpButtonCreate=\1", v1 := rc);
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.33.28 nmpButtonEnable (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-200, NX-400, NX-900, LX2
Firmware version: 2.64 / 1.00.00
Nav. API level: 2

This function will enable or disable a configurable button on the NMP.

Input:

id : INT
The ID of the button - between 1 and the button count defined in nmpButtonsDefine.

enable : BOOL
Determines if the button is enabled and can raise the "NMP Button Pressed" event.

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-4	- The navigation device rejected the command.

-8 - Illegal ID.
-11 - This is not supported by the device (the device is not an NMP device).
-12 - Navigation interface is busy.

Declaration:

```
FUNCTION nmpButtonEnable : INT;
VAR_INPUT
   id       : INT;
   enable   : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc : INT;
END_VAR;
BEGIN
   ...
   // Enables button 1.
   rc := nmpButtonEnable(id := 1, enable := TRUE);
   IF rc <> 0 THEN
      DebugFmt(message := "Error: nmpButtonEnable=\1", v1 := rc);
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.33.29 nmpButtonVisible (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-200, NX-400, NX-900, LX2
Firmware version: 2.64 / 1.00.00
Nav. API level: 2

This function will set the visibility of a configurable button on the NMP.

Input:

id : INT
The ID of the button - between 1 and the button count defined in nmpButtonsDefine.

visible : BOOL
Determines the visibility of the button.

Returns: INT

0 - Success.
-1 - Navigation interface is not open.
-2 - Error communicating with navigation device.
-4 - The navigation device rejected the command.
-8 - Illegal ID.
-11 - This is not supported by the device (the device is not an NMP device).
-12 - Navigation interface is busy.

Declaration:

```
FUNCTION nmpButtonVisible : INT;
VAR_INPUT
   id       : INT;
   visible  : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc : INT;
END_VAR;
BEGIN
   ...
   // Show button 1.
   rc := nmpButtonVisible(id := 1,visible:=TRUE);
   IF rc <> 0 THEN
      DebugFmt(message := "Error: nmpButtonVisible=\1", v1 := rc);
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.33.30 nmpButtonWidth (Function)

Architecture:	**X32 / NX32 / NX32L**
Device support:	**MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-200, NX-400, NX-900, LX2**
Firmware version:	**2.64 / 1.00.00**
Nav. API level:	**2**

This function will set the width of a configurable button on the NMP.
To prevent the buttons from jumping when updated, it is recommended to only change the width of multiple buttons when they all are invisible.

Input:

id : INT
The ID of the button - between 1 and the button count defined in nmpButtonsDefine.

weight : INT
Determines the width of the button relative to the total weight of all the buttons.
A button with weight 2 is twice as wide as a button with weight 1. If the total weight of the buttons is 4, a button with weight 2 will a width 50% of this.

max_width : INT (0..100)
Determines the maximum width of the button as an integer between 1 and 100, where 100 is the entire width of the panel.To have no maximum value, use 0.
This is used to prevent a button from becoming too wide when only a few buttons are defined by limiting the width to e.g. 50 % of the total width.

Returns: **INT**

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-4	- The navigation device rejected the command.
-8	- Invalid parameters.

-11 - This is not supported by the device (the device is not an NMP device).
-12 - Navigation interface is busy.

Declaration:

```
FUNCTION nmpButtonWidth : INT;
VAR_INPUT
   id        : INT;
   weight    : INT := 1;
   max_width : INT := 0;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Sets the width of four buttons: The first one is small, the next one
twice as wide.
   // The next one takes up the same space as the second, but is only as wide
as the first one.
   // The fourth button is the same width as all the others combined.
   nmpButtonWidth(id:=1, weight:=1);
   nmpButtonWidth(id:=2, weight:=2);
   nmpButtonWidth(id:=3, weight:=2, max_width:=10);
   nmpButtonWidth(id:=4, weight:=5);
   ...
END;
END_PROGRAM;
```

4.2.33.31 nmpButtonColor (Function)

Architecture:	**X32 / NX32 / NX32L**
Device support:	**MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-200, NX-400, NX-900, LX2**
Firmware version:	**2.64 / 1.00.00**
Nav. API level:	**2**

This function will set the background colors for a configurable button on the NMP.

Input:

id : INT
The ID of the button - between 1 and the button count defined in nmpButtonsDefine.

color: DINT (-1, 16#00_00_00..16#FF_FF_FF, default: -1)
Determines the color of the button. If -1, the theme color will be used.
Instead of specifying the color directly, the nmpRGBToDint function can be used.

night_color: DINT (-1, 16#00_00_00..16#FF_FF_FF, default: -1)
Determines the color of the button to use when in the night theme. If -1, the day color will be used - unless it is set to use the theme, in which case the color of the night theme will be used.
Instead of specifying the color directly, the nmpRGBToDint function can be used.

Returns: INT

0 - Success.
-1 - Navigation interface is not open.
-2 - Error communicating with navigation device.
-4 - The navigation device rejected the command.
-8 - Illegal ID.
-11 - This is not supported by the device (the device is not an NMP device).
-12 - Navigation interface is busy.

Declaration:

```
FUNCTION nmpButtonColor : INT;
VAR_INPUT
   id          : INT;
   color       : DINT := -1;
   night_color : DINT := -1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc : INT;
END_VAR;
BEGIN
   ...
   // Set backgound color for button 1 to a bright yellow, and the night color
to a dark yellow
   rc := nmpButtonColor(id := 1, color := 16#FF_FF_00, night_color :=
16#C8_C8_00);
   IF rc <> 0 THEN
      DebugFmt(message := "Error: nmpButtonColor=\1", v1 := rc);
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.33.32 nmpButtonText (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-200, NX-400, NX-900, LX2
Firmware version: 2.64 / 1.00.00
Nav. API level: 2

This function will set the text of a configurable button on the NMP.

Input:

id : INT
The ID of the button - between 1 and the button count defined in nmpButtonsDefine.

text : STRING
The text of the button. Max. 49 characters.

Returns: INT

0 - Success.
-1 - Navigation interface is not open.

-2 - Error communicating with navigation device.
-4 - The navigation device rejected the command.
-8 - Illegal ID or text length.
-11 - This is not supported by the device (the device is not an NMP device).
-12 - Navigation interface is busy.

Declaration:

```
FUNCTION nmpButtonText : INT;
VAR_INPUT
   id       : INT;
   text     : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc : INT;
END_VAR;
BEGIN
   ...
   // Set the text of the button
   rc := nmpButtonText(id := 1, text := "Click Me");
   IF rc <> 0 THEN
      DebugFmt(message := "Error: nmpButtonText=\1", v1 := rc);
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.33.33 nmpButtonFlash (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-200, NX-400, NX-900, LX2
Firmware version: 2.64 / 1.00.00
Nav. API level: 2

This function will make the background of a configurable button on the NMP begin or stop alternating between its normal background color and its specified color. This feature can be used to bring attention to the device.

Input:

id : INT
The ID of the button - between 1 and the button count defined in nmpButtonsDefine.

flash : BOOL
This determines if the button should flash or not. If false, the remaining parameters are ignored.

duration : INT (0..32767, default : 0)
The number of seconds to flash for. Use 0 to keep flashing until stopped with a call to nmpButtonFlash.

period : INT (100..32767, default : 1000)
The number of milliseconds between each color change. This can be used to change the flashing frequency and is rounded up to the next 100 milliseconds.

color: DINT (16#00_00_00..16#FF_FF_FF, default: 16#00_FF_00)
Determines the color that the button should flash to.
Instead of specifying the color directly, the nmpRGBToDint function can be used.

night_color: DINT (-1, 16#00_00_00..16#FF_FF_FF, default: -1)
Determines the color of the button to use when flashing in the night theme. If -1, the normal color will be used.
Instead of specifying the color directly, the nmpRGBToDint function can be used.

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-4	- The navigation device rejected the command.
-8	- Invalid parameter.
-11	- This is not supported by the device (the device is not an NMP device).
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION nmpButtonFlash : INT;
VAR_INPUT
   id       : INT;
   flash    : BOOL;
   duration : INT := 0;
   period   : INT := 1000;
   color    : DINT := 16#00FF00;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc : INT;
END_VAR;
BEGIN
   ...
   // Make button 1 flash blue for 5 seconds
   rc := nmpButtonFlash(id := 1, flash := TRUE, duration := 5, period := 500,
color := 16#00_00_FF);
   IF rc <> 0 THEN
      DebugFmt(message := "Error: nmpButtonFlash=\1", v1 := rc);
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.33.34 nmpButtonConfirm (Function)

Architecture:	**X32 / NX32 / NX32L**
Device support:	**MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-200, NX-400, NX-900, LX2**
Firmware version:	**2.80 / 1.00.00**
Nav. API level:	**2**

This function will make the button require confirmation before the "Button Pressed" event will be created.

Input:

id : INT
The ID of the button - between 1 and the button count defined in nmpButtonsDefine.

confirm : BOOL (*Default: FALSE*)
Determines if the button requires confirmation.

message : STRING
Determines the text in the confirmation dialog. Max. 199 characters.

Returns: INT

0	- Success.
-1	- Navigation interface is not open.
-2	- Error communicating with navigation device.
-4	- The navigation device rejected the command.
-8	- Invalid parameter.
-11	- This is not supported by the device (e.g. the device is not an NMP device).
-12	- Navigation interface is busy.

Declaration:

```
FUNCTION nmpButtonConfirm : INT;
VAR_INPUT
   ID       : INT;
   confirm  : BOOL;
   message  : STRING := "";
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc : INT;
END_VAR;
BEGIN
   ...
   // Make button 1 require confirmation
   rc := nmpButtonConfirm(id := 1, confirm := TRUE, message := "Are you
sure?");
   IF rc <> 0 THEN
      DebugFmt(message := "Error: nmpButtonConfirm=\1", v1 := rc);
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.33.35 nmpButtonPressedReceive (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: MX2i pro, CX1 pro/flex/warp, SX1, MX2 turbo, NX-200, NX-400, NX-900, LX2
Firmware version: 2.64 / 1.00.00
Nav. API level: 2

This function block returns the ID of a pressed configurable button on an NMP.

Note that the ID is only valid after an "NMP Button Pressed" event is raised; the event will be removed when this function block is called.

Input:

None.

Output:

id : INT
The ID of the pressed button.

ready : BOOL
TRUE: If data is ready.
FALSE: If data is not ready.

Declaration:

```
FUNCTION_BLOCK nmpButtonPressedReceive;
VAR_OUTPUT
   id    : INT;
   ready : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   buttonPress : nmpButtonPressedReceive;
END_VAR;

FUNCTION ButtonPressed;
   buttonPress();
   IF buttonPress.ready THEN
      DebugMsg(message := "*** Button Pressed ***");
      DebugFmt(message := "-button=\1", v1 := buttonPress.id);

   END_IF;
END_FUNCTION;

THREAD_BLOCK navMonitor;
VAR
   event : INT := 0;
END_VAR;

WHILE event <> -1 DO
   event := navWaitEvent(timeout := -1);
   CASE event OF
      ...
      11: ButtonPressed();
      ...
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;
```

4.2.34 ow: OneWire functions

4.2.34.1 OneWire: Functions to interact with OneWire devices

This group of functions allows connections of Maxim 1-Wire devices to the RTCU device. 1-Wire is a simple and very easy to implement serial bus. It consists of only two wires (Signal and GND) and makes it very easy to implement remote sensing in various applications.

Currently, direct support for iButton and temperature sensors is implemented, and support for additional devices can be added by using the generic OneWire functions.

- owSearch — Searches for OneWire devices.
- owTempGetID — Gets the ID of a specific temperature sensor.
- owTempGet — Gets the temperature from a specific device.
- owiButtonGetID — Queries iButton for its ID.
- owiButtonReadData — Reads data from a memory iButton.
- owiButtonWriteData — Writes data to a memory iButton.
- owiButtonEnableLED — Enables or disables the use of the iButton reader LED.
- owiButtonSetLed — Controls the iButton LED.

Generic OneWire functions:

- owSearchX — Advanced search function for OneWire devices.
- owQuery — Queries the number of devices in a family.
- owGetID — Queries the ID of a device.
- owGetFamily — Returns the family ID of the device.
- owAccess — Selects a device and locks the OneWire net.
- owRelease — Releases the lock on the OneWire net.
- owReadData — Reads data from the OneWire net.
- owWriteData — Writes data to the OneWire net.
- owRead — Reads a byte from the OneWire net.
- owReadBit — Reads a bit from the OneWire net.
- owWrite — Writes a byte to the OneWite net.
- owWriteBit — Writes a bit to the OneWire net.
- owAlarmSearch — Searches for alerted devices.
- owAlarmQuery — Queries if there are alerted devices.
- owAlarmGetID — Queries the ID of the alerted device.
- owAlarmGetFamily — Reads the family of the alerted device.
- owAlarmAccess — Selects the alerted device and locks the OneWire net.
- owAlarmRelease — Releases the lock on the OneWire net.

The implementation of OneWire has the following limitations:

- It is only possible to read/write to the last iButton discovered with owiButtonGetID.

4.2.34.2 owSearch (Function)

```
Architecture:       X32 / NX32 / NX32L
Device support:     All devices with 1-Wire
Firmware version:   1.00 / 1.00.00
```

Each 1-wire device has a unique ROM-number like the iButton. This number is used to point out one specific device of many devices on the bus.

Before any devices can be accessed, the ROM-number has to be discovered. By using the 1-Wire search function, all devices on the 1-Wire bus are discovered and the ROM-numbers are stored in a temporary list placed inside the RTCU device. When the device is powered off, the list is erased and a new search has to be performed on power on. The devices are sorted in the order they are discovered on the 1-Wire net. If a device is removed from the bus, all devices after this one will move one position down the list when a new search is carried out. As the search function is generally intended for 1-Wire use, a family number has to be supplied when making the function call. For 1-Wire temperature devices, the family number is 1.

owSearch returns the number of family-specified devices found on the 1-Wire bus. This function is essential in order to access the 1-Wire devices on the 1-Wire bus (except the iButton). Call the function at least at device power-on.

Please consult the Logic IO 1-Wire basics application note for information on 1-Wire devices.

Input:

family : SINT
The family of 1-Wire devices to search for.
1 = Temperature sensors (current only family code 16#28 as maxim's `DS18B20`).

Returns: INT

>= 0	Number of devices found within family range.
-1	General error or not supported on target.

Declaration:

```
FUNCTION owSearch : INT;
VAR_INPUT
   family : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   Temp : DINT;
   tmp  : DINT;
   str  : STRING;
END_VAR;

PROGRAM test;

   // Search for temperature sensors
   IF owSearch(family:=1) < 1 THEN
      DebugMsg(message:="No temperature sensor!");
   END_IF;

BEGIN
   ...
   // Get temperature of first device
   Temp := owTempGet(device:=1);
   str  := strConcat(str1:=owTempGetID(Device:=1),str2:=" = ");
   str  := strConcat(str1:=str,str2:=dintToStr(v:=(Temp/100)));
   str  := strConcat(str1:=str,str2:=".");
   tmp  := (Temp MOD 100) / 10;
   IF tmp < 0 THEN tmp := -tmp; END_IF;
   str  := strConcat(str1:=str,str2:=dintToStr(v:=tmp));
   DebugFmt(message:=str);
   ...
END;
```

```
END_PROGRAM;
```

4.2.34.3 owTempGetID (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with 1-Wire
Firmware version: 1.00 / 1.00.00

---

owTempGetID returns the ID of a specific 1-Wire temperature sensor. After the owSearch function has been carried out, the ID of each device on the bus can be read. Place the number (between 1 and the number returned by the owSearch function) specifying which temperature sensor is to be read.

Please consult the Logic IO 1-Wire basics application note for information on 1-Wire devices.

Input:

Device : SINT

Returns: STRING

The ID of the temperature sensor.

Declaration:

```
FUNCTION owTempGetID : STRING;
VAR_INPUT
   device : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   Temp : DINT;
   tmp  : DINT;
   str  : STRING;
END_VAR;

PROGRAM test;

   // Search for temperature sensors
   IF owSearch(family:=1) < 1 THEN
      DebugMsg(message:="No temperature sensor!");
   END_IF;

BEGIN
   ...
   // Get temperature
   Temp := owTempGet(device:=1);
   str  := strConcat(str1:=owTempGetID(Device:=1),str2:=" = ");
   str  := strConcat(str1:=str,str2:=dintToStr(v:=(Temp/100)));
   str  := strConcat(str1:=str,str2:=".");
   tmp  := (Temp MOD 100) / 10;
   IF tmp < 0 THEN tmp := -tmp; END_IF;
   str  := strConcat(str1:=str,str2:=dintToStr(v:=tmp));
   DebugFmt(message:=str);
   ...
END;

END_PROGRAM;
```

4.2.34.4 owTempGet (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with 1-Wire
Firmware version: 1.00 / 1.00.00

---

owTempGet returns the temperature of a specific 1-Wire temperature sensor. After the owSearch function has been carried out, the temperature of each device on the bus can be read. Place the number (between 1 and the number returned by the owSearch function) specifying which temperature sensor is to be read.

The function reads the temperature of the device and returns the temperature in 0.01°C multiplied with 100 - this means that 22.50 degrees will be returned as 2250 and -17.00 degrees will be returned as –1700.

The temperature sensors retrieve its power directly from the 1-Wire bus. This is called parasitic power and no external power is needed for remote temperature sensing. A temperature conversion takes approximately 0.8 -> 1.0 seconds in this mode. This is due to the power needed for a temperature conversion when operating in parasitic power mode.

Input:

Device : SINT

Returns: DINT

The current temperature - coded as Degrees * 100 + Decimal degrees. Example: a temperature of 22.76 Degrees is returned as 2276.
-9999 - General error.

Declaration:

```
FUNCTION owTempGet : DINT;
VAR_INPUT
   device : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   Temp : DINT;
   tmp  : DINT;
   str  : STRING;
END_VAR;

PROGRAM test;

   // Search for temperature sensors
   IF owSearch(family:=1) < 1 THEN
      DebugMsg(message:="No temperature sensor!");
   END_IF;

BEGIN
   ...
   // Get temperature
   Temp := owTempGet(device:=1);
   str  := strConcat(str1:=owTempGetID(Device:=1),str2:=" = ");
   str  := strConcat(str1:=str,str2:=dintToStr(v:=(Temp/100)));
   str  := strConcat(str1:=str,str2:=".");
```

```
    tmp  := (Temp MOD 100) / 10;
    IF tmp < 0 THEN tmp := -tmp; END_IF;
    str  := strConcat(str1:=str,str2:=dintToStr(v:=tmp));
    DebugFmt(message:=str);
    ...
END;

END_PROGRAM;
```

4.2.34.5 owiButtonGetID (Function)

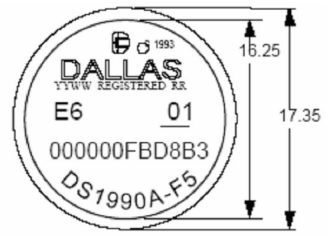
Architecture: X32 / NX32 / NX32L
Device support: All devices with 1-Wire
Firmware version: 1.00 / 1.00.00

---

owiButtonGetID returns a string with the unique serial number of a Maxim 1-Wire iButton. The iButton is a unique, factory-lasered and tested 48-bit registration number that assures absolute trace ability because not two parts are alike. It comes in a small metal MicroCan that is very robust even in harsh environments. In order to read the iButton, a special iButton reader must be used. The reader is attached to the 1-Wire bus and connects the iButton to the 1-Wire bus. A small keyring (iButton Fob) adapter is available so that the iButton is easier to carry and place in the reader.

Please consult the Logic IO 1-Wire Basics application note for information on 1-Wire devices.

In order to use the function it is necessary to know the ROM-number of the iButton. The ROM-number can be read on the top of the iButton. Please see the figure below. All twelve digits have to be read.



The serial number from the above figure is "000000FBD8B3". When the ROM-number is known, a string compare can be done to clarify if the iButton has a valid serial number or not.

It cannot be determined when a user places an iButton in the reader, and it is therefore important to call the owiButtonGetID function very often so that the user will not experience a delay before the iButton is read. Make sure not to have any more time consuming tasks in the main program loop than necessary alternatively implement the owiButtonGetID function as a thread (please see the chapter about multithreading for more information).

The iButton reader is supplied with an LED that can easily be used to inform the user whether the iButton placed in the reader is valid or not. Example: fast blinking = access, slow blinking = no access. The LED is controlled with the owiButtonEnableLED and owiButtonSetLed functions.

Please note that it is only possible to have a single iButton connected to the 1-Wire bus at a time.

Input:
None.

Returns: STRING
A string containing the ID of the iButton if present or an empty string if not.

Declaration:

```
FUNCTION owiButtonGetID : STRING;
```

Example:

```
INCLUDE rtcu.inc

VAR
   iButtonID : STRING;
END_VAR;

PROGRAM test;

   // Enable use of I-Button Led
   owiButtonEnableLED( enable := ON );
BEGIN
   ...
   // Get I-Button ID
   iButtonID := owiButtonGetID();
   // Was a I-Button present?
   IF strLen(str:=iButtonID) = 12 THEN
      // Valid I-Button ID is 12 characters long (000008F5E179)
      // Turn I-Button LED on to show that I-Button ID has been read
      owiButtonSetLED( state := ON );
      ...
      // Turn I-Button LED off
      owiButtonSetLED( state := OFF );
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.34.6 owiButtonReadData (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with 1-Wire
Firmware version: 1.50 / 1.00.00

owiButtonReadData will read a block of data from a memory iButton previously written by the owiButtonWriteData function.

Before a read can be done, an owiButtonGetID must be completed.

When a read request is executed, the index and data size is verified against the following possible values:

1-Wire family	*Type (Dallas/Maxim)*	*Memory*	*Max index*	*Max data size*
08	DS1992	1024 bits	4	29 Bytes
06	DS1993	4096 bits	16	29 Bytes

If the iButton found by owiButtonGetID is not present at the time of reading, an "iButton not present" (-2) will be returned.

On a successful read operation, the number of bytes actually read from the iButton will be returned, which may be less than the requested size.

A "no data" (-5) will be returned if there is no data or the data found was not previously written by owiButtonWriteData function.

Note: partial data may be read in case of an unsuccessful read operation.

Input:

index : INT(1..Max)
Location number the data should be loaded from.

data : ADR
Address of the memory block that receives the loaded data.

size : INT(0..Max)
Maximum number of bytes to load into "ptr".

Returns: INT

>=0	Number of bytes read.
-1	General read error.
-2	iButton not present.
-3	Illegal index value.
-4	Illegal data size value.
-5	No data.

Declaration:

```
FUNCTION owiButtonReadData : INT;
VAR_INPUT
   index    : INT;
   data     : PTR;
   size     : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   id    : STRING;
   data  : ARRAY[1..29] OF SINT;
   rc    : INT;
   str   : STRING;
   i     : INT;
END_VAR;
BEGIN
   // Retrieve ID of I-Button
   id := owiButtonGetID();

   // iButton detected.
   IF strLen(str := id) <> 0 THEN
      // clear data area
      FOR i := 1 TO SIZEOF(data) DO
         data[i] := 0;
      END_FOR;

      rc := owiButtonReadData(index := 1, data := ADDR(data), size :=
SIZEOF(data));
      IF rc < 0 THEN
         // Write a string to the debug window
         DebugFmt(message := "Failed to read iButton error code \1", v1 :=
rc);
      ELSE
         DebugFmt(message := "Read \1 bytes from index 1 on " + id + ".", v1
:= rc);
         // Copy contents from memory into a string
         str := strFromMemory(src := ADDR(data), len := SIZEOF(data));
         DebugMsg(message := "Content : " + str);
      END_IF;
```

```
   END_IF;
END;

END_PROGRAM;
```

4.2.34.7 owiButtonWriteData (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with 1-Wire
Firmware version: 1.50 / 1.00.00

---

owiButtonWriteData will write a block of data to a memory iButton for later to be read by owiButtonReadData.

Before a write can be done, an owiButtonGetID must be completed.

When a write request is executed, the index and data size is verified against the following possible values:

1-Wire family	*Type (Dallas/Maxim)*	*Memory*	*Max index*	*Max data size*
08	DS1992	1024 bits	4	29 Bytes
06	DS1993	4096 bits	16	29 Bytes

If the iButton found by owiButtonGetID is not present at the time of writing, an "iButton not present" (-2) will be returned.

Input:

index : INT(1..Max)
Location number the data should be stored in.

data : ADR
Address of the memory block that holds the data to be stored.

size : INT(0..Max)
Maximum number of bytes to store from "ptr".

Returns: INT

0	Data written successful.
-1	General read error.
-2	iButton not present.
-3	Illegal index value.
-4	Illegal data size value.

Declaration:

```
FUNCTION owiButtonWriteData : INT;
VAR_INPUT
   index    : INT;
   data     : PTR;
   size     : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   id    : STRING;
   data  : ARRAY[1..29] OF SINT;
   rc    : INT;
   str   : STRING;
```

```
   i     : INT;
END_VAR;
BEGIN
   // Retrieve ID of I-Button
   id := owiButtonGetID();

   // iButton detected.
   IF strLen(str := id) <> 0 THEN
      // clear data area
      FOR i := 1 TO SIZEOF(data) DO
         data[i] := 0;
      END_FOR;

      str := strFormat(format := "ID: " + id);
      // Copy contents of a string to memory
      strToMemory(dst := ADDR(data), str := str, len := strLen(str := str));

      rc := owiButtonWriteData(index := 1, data := ADDR(data), size :=
SIZEOF(data));
      IF rc <> 0 THEN
         // Write a string to the debug window
         DebugFmt(message := "Write to iButton failed error code \1", v1 :=
rc);
      ELSE
         DebugMsg(message := "Data written to index 1 on " + id);
         DebugMsg(message := "Content : " + str);
      END_IF;
   END_IF;
END;

END_PROGRAM;
```

4.2.34.8 owiButtonEnableLED (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with 1-Wire except CX1
Firmware version: 1.00 / 1.00.00

---

owiButtonEnableLED enables use of the LED placed inside the iButton reader. The LED is turned on and off with the owiButtonSetLED function. Before the state can be controlled, owiButtonEnableLED must be set to "on".

Input:

enable : BOOL
This enables or disables the use of the LED on the iButton reader.

Returns: INT

0 - Successful.
-1 - 1-Wire not supported on target.

Declaration:

```
FUNCTION owiButtonEnableLED : INT;
VAR_INPUT
   enable : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   iButtonID : STRING;
END_VAR;

PROGRAM test;

   // Enable use of I-Button Led
   owiButtonEnableLED( enable := ON );
BEGIN
   ...
   // Get I-Button ID
   iButtonID := owiButtonGetID();
   // Was a I-Button present?
   IF strLen(str:=iButtonID) = 12 THEN
      // Valid I-Button ID is 12 characters long (000008F5E179)
      // Turn I-Button LED on to show that I-Button ID has been read
      owiButtonSetLED( state := ON );
      ...
      // Turn I-Button LED off
      owiButtonSetLED( state := OFF );
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.34.9 owiButtonSetLED (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with 1-Wire except CX1
Firmware version: 1.00 / 1.00.00

The iButton reader is supplied with an LED that can easily be used to inform the user if the iButton placed in the reader is valid or not. Example: fast blinking = access, slow blinking = no access. The LED is controlled with the owiButtonEnableLED and owiButtonSetLed functions.

The owiButtonSetLED controls the state of the LED placed inside the iButton reader. Before the state can be controlled, owiButtonEnableLED must be set to "on".

Please notice:
When the RTCU device has been connected to the RTCU IDE, the user may experience a delay before the LED is ready for use. This is due to a timeout when disconnecting from the RTCU-IDE.
A delay may also be experienced when the owiButtonEnableLED is set to "on" and the user connects the RTCU device to the RTCU IDE.

Input:

state : BOOL
The new state of the LED in the iButton reader.

Returns:

None.

Declaration:

```
FUNCTION owiButtonSetLed;
VAR_INPUT
```

```
   state : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   iButtonID : STRING;
END_VAR;

PROGRAM test;

   // Enable use of I-Button Led
   owiButtonEnableLED( enable := ON );
BEGIN
   ...
   // Get I-Button ID
   iButtonID := owiButtonGetID();
   // Was a I-Button present?
   IF strLen(str:=iButtonID) = 12 THEN
      // Valid I-Button ID is 12 characters long (000008F5E179)
      // Turn I-Button LED on to show that I-Button ID has been read
      owiButtonSetLED( state := ON );
      ...
      // Turn I-Button LED off
      owiButtonSetLED( state := OFF );
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.34.10 owSearchX (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with 1-Wire
Firmware version: 2.80 / 1.00.00

An advanced version of the owSearch function that allows detection of all 1-Wire devices on the bus which belong to the specified families.

Each 1-Wire device has a unique ROM-number like the iButton. This number is used to point out one specific device of many devices on the bus.
Before any devices can be accessed, the ROM-number has to be discovered. By using the 1-Wire search function, all devices on the 1-Wire bus are discovered and the ROM-numbers are stored in a temporary list placed inside the RTCU device.

The list of known devices will be update when the following functions are called:

- owSearchX
- owSearch
- owiButtonGetID

The devices are sorted in the order they are discovered on the 1-Wire net. If a device is removed from the bus, all devices after this will move one position down the list when a new search is carried out As the search function is intended for general 1-Wire use, a family number can be supplied when making the function call to optimize the search.

Current official list of Maxim 1-Wire family codes:
16#01 DS1990A, DS1990R, DS2401, DS2411.

16#02 DS1991.
16#04 DS1994, DS2404.
16#05 DS2405.
16#06 DS1993.
16#08 DS1992.
16#09 DS1982, DS2502.
16#0A DS1995.
16#0B DS1985, DS2505.
16#0C DS1996.
16#0F DS1986, DS2506.
16#10 DS1920.
16#12 DS2406, DS2407.
16#14 DS1971, DS2430A.
16#1A DS1963L.
16#1C DS28E04-100.
16#1D DS2423.
16#1F DS2409.
16#20 DS2450.
16#21 DS1921G, DS1921H, DS1921Z.
16#23 DS1973, DS2433.
16#24 DS1904, DS2415.
16#27 DS2417.
16#28 DS18B20.
16#29 DS2408.
16#2C DS2890.
16#2D DS1972.
16#37 DS1977.
16#3A DS2413.
16#41 DS1922L, DS1922T, DS1923, DS2422.
16#42 DS28EA00.
16#43 DS28EC20.

To optimize the search time, only conduct a single search for all devices.

Input:

first : INT [0..255] (*default* 0)
Skips devices with lower family codes.

last : INT [0..255] (*default* 255)
Does not search for devices with higher family codes than this.

reset : BOOL [TRUE, FALSE] (*default* TRUE)
Deletes all known devices before search.

Returns: INT

>= 0 Number of devices found within family range.
-1 General error or not supported on target.

Declaration:

```
FUNCTION owSearchX : INT;
VAR_INPUT
   first : INT := 16#00;
   last  : INT := 16#FF;
   reset : BOOL := TRUE;
END_VAR;
```

Example:

Please see the "Examples - Generic OneWire Example"

4.2.34.11 owQuery (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with 1-Wire
Firmware version: 2.80 / 1.00.00

This function will query the number of known devices within a single or all families found with owSearchX, owSearch, or owiButtonGetID.

Input:

family : INT [0..255] (*default* 0)
Queries number of known devices within this family code.
0 = All known devices

Returns: INT

>=0 Number of devices found within given family.
-1 1-Wire not supported on target.
-4 Illegal family code.

Declaration:

```
FUNCTION owQuery : INT;
VAR_INPUT
   family    : INT := 16#00;
END_VAR;
```

Example:

Please see the "Examples - Generic OneWire Example"

4.2.34.12 owGetID (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with 1-Wire
Firmware version: 2.80 / 1.00.00

This function will return the ROM ID number of a known device previously found with owSearchX, owSearch, or owiButtonGetID.

Input:

family : INT [0..255] (*default* 0)
Family code of the device.
0 = Any family code. The device count will ignore family code and will represent the raw order of found devices with the lowest family first.

device : INT [1..50] (*default* 1)
Device number within the applied family.
1 = This will give the first device within family if present. When <family> is 0, the device count continues across family codes.

Returns: STRING

A string containing the ID of the device if present or an empty string if not.

Declaration:

```
FUNCTION owGetID : STRING;
VAR_INPUT
   family   : INT := 16#00;
   device   : INT := 1;
END_VAR;
```

Example:

Please see the "Examples - Generic OneWire Example"

4.2.34.13 owGetFamily (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with 1-Wire
Firmware version: 2.80 / 1.00.00

This function returns the family code of a known device previously found with owSearchX, owSearch, or owiButtonGetID.

Input:

device : INT [1..50] (*default* 1)
Device number of a found device across all family codes.

Returns: INT

> 0	1-Wire family code.
-1	1-Wire not supported on target.
-2	Device not found.
-3	Illegal device number.

Declaration:

```
FUNCTION owGetFamily : INT;
VAR_INPUT
   device    : INT := 1;
END_VAR;
```

Example:

Please see the "Examples - Generic OneWire Example"

4.2.34.14 owAccess (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with 1-Wire
Firmware version: 2.80 / 1.00.00

This function will select a 1-Wire device and prepare it for device-specific commands.
The device must first have been discovered with owSearchX.

This will also acquire the 1-Wire lock - thus allowing no other threads access to the 1-Wire net until it has been released by calling owRelease. Failing to do so will result in an fault 148 (see Fault codes for further details).

The lock can be refreshed with another call to owAccess but owRelease must be called the same number of times before all locks are released.

Input:

family : INT [0..255] (*default* 0)
Family code of the device to access/select.
0 = Any family code. The device count will ignore family code and will represent the raw order of found devices with the lowest family first.

device : INT [1..50] (*default* 1)
Device number within the applied family.
1 = This will give the first device within family if present. When <family> is 0, the device count continues across family codes.

Returns: INT

0	Successful
-1	General error or not supported on target.
-2	Device not present.
-3	Illegal device number.
-4	Illegal family code.

Declaration:

```
FUNCTION owAccess : INT;
VAR_INPUT
   family   : INT := 16#00;
   device   : INT := 1;
END_VAR;
```

Example:

Please see the "Examples - Generic OneWire Example"

4.2.34.15 owRelease (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with 1-Wire
Firmware version: 2.80 / 1.00.00

This releases the lock to the 1-Wire net previously obtained with owAccess.

Input:

None.

Returns: INT

0	Successful.
-1	1-Wire not supported on target.
-6	Not locked.

Declaration:

```
FUNCTION owRelease : INT;
```

Example:

Please see the "Examples - Generic OneWire Example"

4.2.34.16 owReadData (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with 1-Wire
Firmware version: 2.80 / 1.00.00

This function will read data from the 1-Wire net. A device must first have been selected with owAccess. If the lock has not been acquired, all data will be read as if the device has been removed from the 1-Wire net.
See also owRead and owReadBit.

Input:

data : PTR
Address of the memory block that holds the data to be stored.

size : INT [1..32767] (*default 0*)
Number of bytes/bits to read.

bitmode : BOOL [TRUE, FALSE] (default *FALSE*)
If true, each bit received is stored as a byte.

Returns: INT

0	Successful.
-1	1-Wire not supported on target.
-5	Missing pointer to data.

Declaration:

```
FUNCTION owReadData : INT;
VAR_INPUT
   data     : PTR;
   size     : INT  := 0;
   bitmode  : BOOL := FALSE;
END_VAR
```

Example:

Please see the "Examples - Generic OneWire Example"

4.2.34.17 owWriteData (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with 1-Wire
Firmware version: 2.80 / 1.00.00

This function will write data to the 1-Wire net, a device must first have been selected with owAccess. If the lock has not been acquired, no data will be transmitted.
See also owWrite and owWriteBit.

Input:

data : PTR
Address of the memory block that holds the data to be written.

size : INT [1..32767] (*default 0)*
Number of bytes/bits to read.

bitmode : BOOL [TRUE, FALSE] (default *FALSE*)
If true, then each byte is sent as a single bit, where the value "0" is sent as a "0" and any other value is sent as a "1".

Returns: INT

0 Successful.
-1 1-Wire not supported on target.
-5 Missing pointer to data.

Declaration:

```
FUNCTION owReadData : INT;
VAR_INPUT
   data     : PTR;
   size     : INT  := 0;
   bitmode  : BOOL := FALSE;
END_VAR
```

Example:

Please see the "Examples - Generic OneWire Example"

4.2.34.18 owRead (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with 1-Wire
Firmware version: 2.80 / 1.00.00

This function will read a single byte from the 1-Wire net. A device must first have been selected with owAccess. If not, then data will be received as 16#FF as if connection to the device has been lost.

Input:

None.

Output:

data : SINT
Data received.

Returns: INT

0 Successful.
-1 1-Wire not supported on target.

Declaration:

```
FUNCTION owRead : INT;
VAR_INPUT
   data  : ACCESS SINT;
END_VAR
```

Example:

Please see the "Examples - Generic OneWire Example"

4.2.34.19 owReadBit (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with 1-Wire
Firmware version: 2.80 / 1.00.00

This function will read a single bit from the 1-Wire net. A device must first have been selected with owAccess. If not, then data will be received as false as if connection to the device has been lost.

Input:
None.

Output:
data : BOOL
Data received.

Returns: INT
0 Successful
-1 1-Wire not supported on target.

Declaration:
```
FUNCTION owReadBit : INT;
VAR_INPUT
   data  : ACCESS BOOL;
END_VAR
```

Example:

Please see the "Examples - Generic OneWire Example"

4.2.34.20 owWrite (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with 1-Wire
Firmware version: 2.80 / 1.00.00

This function will write a single byte to the 1-Wire net. A device must first have been selected with owAccess. If the lock has not been acquired, no data will be transmitted.

Input:
data : SINT [16#00 .. 16#FF]
Data to send.

Returns: INT
0 Successful.
-1 1-Wire not supported on target.

Declaration:
```
FUNCTION owWrite : INT;
VAR_INPUT
```

```
   data  : SINT;
END_VAR
```

Example:

Please see the "Examples - Generic OneWire Example"

4.2.34.21 owWriteBit (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with 1-Wire
Firmware version: 2.80 / 1.00.00

This function will write a single bit to the 1-Wire net. A device must first have been selected with owAccess. If the lock has not been acquired, no data will be transmitted.

Input:

data : BOOL
Value to send.

Returns: INT

0	Successful.
-1	1-Wire not supported on target.

Declaration:

```
FUNCTION owWriteBit : INT;
VAR_INPUT
   data  : BOOL;
END_VAR
```

Example:

Please see the "Examples - Generic OneWire Example"

4.2.34.22 owAlarmSearch (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with 1-Wire
Firmware version: 2.80 / 1.00.00

This function will search for the first 1-Wire device responding to a conditional search.
See owSearchX to perform a normal search.

To make a device respond to owAlarmSearch, please consult the data sheet for the specific device.

After the device has been discovered, it can be accessed with owAlarmAccess.

Input:

first : INT [0..255] (*default* 0)
Skip devices with lower family codes.

last : INT [0..255] (*default* 255)

Do not search for devices with higher family codes than this.

Returns: INT

1 Device found.
0 No device found.
-1 General error or not supported on target.

Declaration:

```
FUNCTION owAlarmSearch : INT;
VAR_INPUT
   first : INT := 16#00;
   last  : INT := 16#FF;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Search for a DS2408 or a temperature sensor that is alerting.
   IF owAlarmSearch(first := 16#28, last := 16#29) < 1 THEN
      DebugMsg(message:="No alarm!");
   ELSE
      // Get ID of device
      DebugMsg(message:="Alarm on device: " + owAlarmGetID());
      // Get Family of device
      DebugMsg(message:="Device family: " + owAlarmGetFamily());
      ...
   END_IF;
END;

END_PROGRAM;
```

4.2.34.23 owAlarmQuery (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with 1-Wire
Firmware version: 2.80 / 1.00.00

This function will query for alerted devices found with owAlarmSearch.

Input:

None.

Returns: INT

1 A device was found.
0 No device found.
-1 1-Wire not supported on target.

Declaration:

```
FUNCTION owAlarmQuery : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   // Search for One-Wire device with alarm condition
   owAlarmSearch();
   // Query for a device that is alerting.
   IF owAlarmQuery() = 0 THEN
      DebugMsg(message:="No alarm!");
   ELSE
      // Get ID of device
      DebugMsg(message:="Alarm on device: " + owAlarmGetID());
      ...
   END_IF;
END;

END_PROGRAM;
```

4.2.34.24 owAlarmGetID (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with 1-Wire
Firmware version: 2.80 / 1.00.00

This function will return the ROM ID number of the alerted device previously found with owAlarmSearch.

Input:
None.

Returns: STRING
A string containing the ID of the alerted device if present or an empty string if not.

Declaration:
```
FUNCTION owAlarmGetID : STRING;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Query for a device that is alerting.
   IF owAlarmSearch() < 1 THEN
      DebugMsg(message:="No alarm!");
   ELSE
      // Get ID of device
      DebugMsg(message:="Alarm on device: " + owAlarmGetID());
      // Get Family of device
      DebugMsg(message:="Device family: " + owAlarmGetFamily());
      ...
   END_IF;
END;

END_PROGRAM;
```

4.2.34.25 owAlarmGetFamily (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with 1-Wire
Firmware version: 2.80 / 1.00.00

This function returns the family code of the device previously found with owAlarmSearch.

Input:
None.

Returns: INT
> 0 1-Wire family code.
-1 1-Wire not supported on target.
-2 Device not found.

Declaration:
```
FUNCTION owAlarmGetFamily : INT;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Query for a device that is alerting.
   IF owAlarmSearch() < 1 THEN
      DebugMsg(message:="No alarm!");
   ELSE
      // Get ID of device
      DebugMsg(message:="Alarm on device: " + owAlarmGetID());
      // Get Family of device
      DebugMsg(message:="Device family: " + owAlarmGetFamily());
      ...
   END_IF;
END;

END_PROGRAM;
```

4.2.34.26 owAlarmAccess (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with 1-Wire
Firmware version: 2.80 / 1.00.00

This function will select the alerted 1-Wire device and prepare it for device specific commands. The device must first have been discovered with owAlarmSearch.

This will also acquire the 1-Wire lock which allows no other threads access to the 1-Wire net until it has been released by calling owAlarmRelease. Failing to do so will result in a fault 148 (see Fault codes for further details).

The lock can be refreshed with another call to owAlarmAccess but owAlarmRelease must be called the same number of times before all locks are released.

Input:
None.

Returns: INT
0 Successful.
-1 General error or not supported on target.
-2 Device not present.

Declaration:
```
FUNCTION owAlarmAccess : INT;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Query for a device that is alerting.
   IF owAlarmSearch() < 1 THEN
      DebugMsg(message:="No alarm!");
   ELSE
      // Get ID of device
      DebugMsg(message:="Alarm on device: " + owAlarmGetID());
      // Get Family of device
      DebugMsg(message:="Device family: " + owAlarmGetFamily());
      IF owAlarmAccess() = 0 THEN
         // Communicate with device
         ...
         // Release device
         owAlarmRelease();
      END_IF;
   END_IF;
END;

END_PROGRAM;
```

4.2.34.27 owAlarmRelease (Function)

Architecture: X32 / NX32 / NX32L
Device support: All devices with 1-Wire
Firmware version: 2.80 / 1.00.00

Releases the lock to the 1-Wire net previously obtained with owAlarmAccess.

Input:
None.

Returns: INT
0 Successful.
-1 1-Wire not supported on target.
-6 Not locked.

Declaration:

```
FUNCTION owRelease : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Query for a device that is alerting.
   IF owAlarmSearch() < 1 THEN
      DebugMsg(message:="No alarm!");
   ELSE
      // Get ID of device
      DebugMsg(message:="Alarm on device: " + owAlarmGetID());
      // Get Family of device
      DebugMsg(message:="Device family: " + owAlarmGetFamily());
      IF owAlarmAccess() = 0 THEN
         // Communicate with device
         ...
         // Release device
         owAlarmRelease();
      END_IF;
   END_IF;
END;

END_PROGRAM;
```

4.2.35 pm: Power management

4.2.35.1 pm: Power management

As demands for flexible power management has increased for realizing more battery efficient solutions, a new group of power management functions are available which implements the possibilities to control the power usage of the RTCU. The new features offer the application to stop executing and wait for an event to occur before proceeding operation from the location where the program was stopped. Another feature is control of the RTCU execution speed and how the device should behave when an external power fail occurs.

Power management functions for control of the power usage.

- pmPowerDown — Power down the device and reset when an event happens.
- pmExternalPower — Enable/disable the external power to run on battery.
- pmDeepSleep — Makes device sleep for a number milliseconds.
- pmPowerFail — Sets device behavior when external power fails.
- pmWaitEvent — Makes device sleep until an event happens by entering a power saving sleep mode.
- pmSetVibrationsensitivity — Sets the sensitivity of the Vibration sensor.
- pmVibration — Query whether vibration has been detected since last call.
- pmSetSpeed — Control the speed of execution.
- pmSuspend — Makes device sleep until an event happens by entering a power saving sleep mode.
- pmGetWakeSource — Converts the return code from pmSuspend into usable values.
- pmGetWakeSourceString — Converts the return code from pmSuspend into a string for debugging.

4.2.35.2 pmPowerDown (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This powers down the device – similar to PowerDown. The device will wake up with a full reset when the ignition input (please refer to the technical manual of the RTCU) is activated and optionally after a specific time period defined by the time input variable.
A manual reset of the RTCU or restoring the external power will also wake up the RTCU.

Input:

time : DINT
Time to power down in seconds. Leave empty if not used.

Returns: INT

0 - Success.
1 - Ignition is activated.
2 - Error.

Declaration:

```
FUNCTION pmPowerDown : INT;
VAR_INPUT
   time      : DINT := -1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Power down and reset after 60 seconds
   pmPowerDown(time := 60);
   ...
END;

END_PROGRAM;
```

4.2.35.3 pmExternalPower (Function)

Architecture: NX32 / NX32L
Device support: MX2 turbo/encore/warp, AX9 turbo/encore, NX-200, NX-400, NX-900, NX-910, LX2
Firmware version: 4.00 / 1.00.00

Turns the external power on and off. This makes it possible to run on battery even though external power is provided.
If battery backup is not enabled or if BatPowerLevel() is at the lowest level, the external power can not be disabled.

Input:

enable: BOOL
Enables/disables the external power.

Returns:

0 - Success.
1 - Battery backup not enabled.
2 - Battery power too low.
3 - Not supported.

Declaration:

```
FUNCTION pmExternalPower : INT;
VAR_INPUT
   enable   : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Turn off external power
   pmExternalPower(enable := FALSE);
   ...
END;

END_PROGRAM;
```

4.2.35.4 pmDeepSleep (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.02 / 1.00.00

---

pmDeepSleep will stop the processor of the RTCU device for a number of milliseconds - thereby lowering the power consumption considerably.

Note:
During DeepSleep mode, the timers (TP, TON, TOF, etc) will not be updated, and the time spent in DeepSleep mode will not be counted.
The real-time clock functions (clockNow, clockGet, etc.) will be updated correctly.

The DeepSleep() function will degrade to a Sleep() operation if full execution speed is required by other operations such as the battery charger, voice messaging, or the insertion of a programming cable.

Input:

delay : INT (0..32767)
Number of milliseconds to sleep. The minimum sleep period is 200 ms.

Returns:

None.

Declaration:

```
FUNCTION pmDeepSleep;
VAR_INPUT
   delay : INT := -1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Go in low-power mode for 5000 mSecs
   pmDeepSleep(delay := 5000);
   ...
END;

END_PROGRAM;
```

4.2.35.5 pmPowerFail (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This sets whether the device will continue to operate on internal battery or enter pmPowerDown (forever) when external power is lost (power fail). Please also refer to the technical manual for the specific RTCU device for further information and support.
If external power is disabled, operating on internal battery can not be disabled.

Input:

bat : BOOL
If true, the device will operate on internal battery when a power fail occur. If false, the device will enter a power-down mode.

Returns: INT

0 - Success.
1 - External power is disabled.

Declaration:

```
FUNCTION pmPowerFail : INT;
VAR_INPUT
   bat : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   ext_pwr : BOOL;
END_VAR;

gsmPower(power:=ON);
gprsOpen();
DebugFmt(message:="pmPowerFail=\1",v1:=pmPowerFail(bat:=TRUE));

BEGIN
   ...
   IF boardSupplyType() = 1 AND ext_pwr THEN
      ext_pwr := FALSE;
      pmSetSpeed(speed:=2);
   ELSIF boardSupplyType() = 2 AND NOT ext_pwr THEN
      ext_pwr := TRUE;
      pmSetSpeed(speed:=4);
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.35.6 pmGetPowerFail (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.60.00

This function tells if the device will continue to operate on internal battery or enter pmPowerDown (forever) when external power is lost (power fail).
To change behavior see pmPowerFail.

Returns: BOOL

FALSE - Battery backup is disabled.
TRUE - Battery backup is enabled.

Declaration:

```
FUNCTION pmGetPowerFail : BOOL;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM getPF;
VAR
   bat : BOOL;
END_VAR;

bat := pmGetPowerFail();
IF bat THEN
   DebugMsg(message:="Battery backup enabled");
ELSE
   DebugMsg(message:="Battery backup NOT enabled");
END_IF;
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.35.7 pmSetSpeed (Function)

Architecture: X32 / NX32
Device support: All
Firmware version: 1.02

This function is used to set the CPU execution speed to reduce the power consumption. The speed is selected in five steps (0..4) where 0 is slowest. However, when changing to another speed, a number of limitations exist.

Limitations:

- If another function, which must operate at a higher speed than the one requested, is enabled, an error will be returned. Please see the return codes.
- A voice call will temporarily switch to full speed (4) as long as the call is active.
- When the on-board battery is being charged, the device will operate at full speed (4) until charging has terminated.
- Connecting a programming cable will temporary switch to full speed (4) to enable communication with the RTCU IDE. The selected speed is resumed when the cable is removed.
- Using gsmPower is only allowed at speed 3 or 4. Use gsmPowerLP for all other speeds.
- When the speed is changed, all serial communication is interrupted, and the RTCU cannot receive any data that is transmitted to it during the change. This includes CAN, 1-Wire, RS232, and RS485.
- Operating CAN at 1 Mbit is only allowed at speed 1 or higher.
- 1-Wire is only allowed at speed 4.
- On-board Ethernet is only allowed at speed 4.
- It is not possible for RS232 and RS485 to obtain all baud rates at all speeds. Below is an overview of which baud rates that can be used at the specific CPU execution speed.

speed/baudrate	1.2k	2.4k	4.8k	9.6k	19.2k	38.4k	57.6k	115.2k
0								
1								
2								
3								
4								

Input:

speed : INT
0 = Very slow (2 MHz clock).
1 = Slow (6 MHz clock).
2 = Medium (12 MHz clock).
3 = Fast (24 MHz clock).
4 = Very fast (48 MHz clock) (default at boot-up).

Returns: **INT**
0 - Success.
1 - Invalid speed.
2 - CAN communication speed is too high for the requested speed.
3 - GSM is not in low power mode. See gsmPowerLP.
4 - Baud rate selected on serial port 0 does not support the requested speed.
5 - Baud rate selected on serial port 1 does not support the requested speed.
6 - Baud rate selected on serial port 2 does not support the requested speed.
7 - Bluetooth does not support the requested speed.
8 - GPS update frequency does not support the requested speed.
9 - On-board Ethernet does not support the requested speed.

Declaration:
```
FUNCTION pmSetSpeed : INT;
VAR_INPUT
   speed : INT;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   ext_pwr : BOOL;
END_VAR;

gsmPowerLP(power:=ON);
gprsOpen();
DebugFmt(message:="pmPowerFail=\1",v1:=pmPowerFail(bat:=TRUE));

BEGIN
   ...
   IF boardSupplyType() = 1 AND ext_pwr THEN
      ext_pwr := FALSE;
      pmSetSpeed(speed:=2);
   ELSIF boardSupplyType() = 2 AND NOT ext_pwr THEN
      ext_pwr := TRUE;
      pmSetSpeed(speed:=4);
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.35.8 pmSetVibrationSensivity (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, CX1, SX1, MX2 turbo/encore/warp, NX-200, NX-900, LX2, LX5
Firmware version: 1.00 / 1.40.00

This sets the vibration sensor sensitivity (please see the RTCU technical manual for support of the vibration sensor). With this function, the sensitivity of the vibration sensor can be set. The RTCU does not remember the value over a reset or PowerDown, and it must at least be set during program initialization.

Input:

sen : SINT (0..100) (default: 50)
The level of sensitivity. 100 is most sensitive. 0 (zero) disables vibration detection.

Returns: INT

0 - Success.
1 - Invalid value.

Declaration:

```
FUNCTION pmSetVibrationSensivity : INT;
VAR_INPUT
   sen : SINT := 50;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   sw : BOOL;
END_VAR;

PROGRAM test;
VAR
   sms    : gsmIncomingSMS;
   writer : logWrite;
   tmp    : INT;
   sen    : INT := 50;
END_VAR;

pmSetVibrationSensivity(sen:=100);
gsmPower(power:=ON);
gprsOpen();

IF NOT logIsInitialized(key:=4711) THEN
   logInitialize(key:=4711, numlogvalues:=2, numlogrecords:=3000);
END_IF;
DebugMsg(message:="Running...");

BEGIN
   IF sw THEN
      sms();
      IF sms.status > 0 THEN
         tmp := strToInt(str:=sms.message);
         IF tmp > 0 AND tmp < 101 THEN
            IF pmSetVibrationSensivity(sen:=SINT(tmp)) = 0 THEN
               sen := tmp;
               DebugFmt(message:="Vibration sensivity: \1",v1:=sen);
            END_IF;
         END_IF;
      END_IF;
   ELSE
      tmp := pmWaitEvent(Vibration:=TRUE,time:=10);
      IF tmp <> 8 THEN
         writer(tag:=1, value[1]:=sen, value[2]:=tmp);
```

```
      END_IF;
   END_IF;
END;

END_PROGRAM;
```

4.2.35.9 pmVibration (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, CX1, SX1, MX2 turbo/encore/warp, NX-200, NX-900, LX2, LX5
Firmware version: 1.00 / 1.40.00

Query whether vibration has been detected by the on-board vibration sensor.
The pmVibration() function will return whether vibration has been detected since the last call. It is the responsibility of application to call pmVibration() frequently so that the information processed is up to date.
The sensivity of the vibration sensor can be adjusted by calling the function pmSetVibrationSensivity().

Note: pmVibration will return FALSE while accelerometer or motion sensor is open.

Input:
None.

Returns: BOOL
TRUE: - Vibration has been detected since last call.
FALSE: - No vibration has been detected since last call.

Declaration:
```
FUNCTION pmVibration : BOOL;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM test;

pmSetVibrationSensivity(sen:=25);

DebugMsg(message:="Running...");
BEGIN
   IF NOT pmVibration() THEN
      DebugMsg(message:="No movement.. Sleep");
      pmWaitEvent(Vibration:=TRUE);
      DebugMsg(message:="Wakeup");
   END_IF;
END;

END_PROGRAM;
```

4.2.35.10 pmWaitEvent (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This function enters power-saving mode and effectively freezes program execution and waits for one or several of the following events:

- Change on digital input 1..4.
- Change on ignition (please refer to the technical manual of the RTCU).
- External power fail.
- External power apply.
- Vibration sensor.
- Timer.
- CAN reception.
- Serial 0 activity.
- Serial 1 activity.
- GSM activity.
- GPS 2D/3D fix (RTCU SX1 series only).
- Intrusion detection (RTCU AX9 turbo only).
- Key pressed (RTCU DX4 series, RTCU NX-400 and RTCU LX4 only)
- RF activity.

When the event(s) happens, the program will continue operation from the location it was called. The polarity (rising or falling edge) of a digital input to wake up from can be set individually for each input. If the vibration sensor is activated, a vibration larger than the threshold set by the pmSetVibrationSensivity will also wake up the RTCU.

Like the pmPowerDown function, a timer can be set to wake up the RTCU - the main difference between the pmPowerDown and the pmWaitEvent is that a power down will restart program execution from the start (as a reset) and the WaitForEvent will proceed program execution from where it was called.

Please note that the pmWaitEvent does not affect the digital outputs. The user must, if desired, deactivate the outputs before calling pmWaitEvent to reduce the power consumed by the outputs.

If the ignition input is selected for wake-up and the input is logical low, a power apply will also wake the RTCU up even if power apply is not selected for wake-up. The return code will be the ignition wake-up code.

If the serial port baud rate is higher than 9,600, the RTCU will not be able to receive the transmitted data. However, the RTCU will wake up if there is any activity on the connection, and as soon as the function has returned, the serial functions are operational again.

It is not possible to use pmWaitEvent while the on-board battery is fast-charging, and therefore it may be necessary to disable the charger by using batChargerEnable.

The pmWaitEvent function will exclude any selected events that are not supported by the RTCU device. If all the selected events are discounted, pmWaitEvent will fail with the error "no events selected". An example could be waiting for a vibration event on a RTCU-DX4 device which does not have vibration detection.

On NX32L devices it is also possible to use the pmSuspend function, which supports additional events and low power modes.

Note when using GSM event:
Using pmWaitEvent with the GSM event enabled implies that the GSM module must be operating in low power mode. See gsmPowerLP.
When the GSM module is on, the GSM event must be selected when calling pmWaitEvent.
pmWaitEvent with GSM event enabled cannot be used when the GSM module is busy with an active voice or CSD session.

Any GSM event, including minor events such as base station changes, will trigger the wake-up, and it is therefore the responsibility of the application to detect and handle any relevant event at wake-up time.
After handling of the event, or if no relevant event was found, pmWaitEvent can be called again while awaiting the next GSM event.
There is an interval of 15 seconds before pmWaitEvent can be called again while awaiting the next GSM event.

Input:

Ignition : BOOL (default: FALSE)
The device waits until the ignition input is activated.

The actual input depends on the device in question. For example on the RTCU MX2i series it is input 5, and on the RTCU CX1 series it is input 1. Please refer to the technical manual of the RTCU for details.

Also note that waiting on an input which is also used for ignition will return the input event and not the ignition event as these are the same.

IgnitionFalling : BOOL (default: FALSE)
If TRUE the ignition is activated with a falling edge. If FALSE the ignition is activated with a rising edge.

DI1 : BOOL (default: FALSE)
The device waits until digital input 1 is activated.

DI1Falling : BOOL (default: FALSE)
If TRUE digital input 1 is activated with a falling edge. If FALSE digital input 1 is activated with a rising edge.

DI2 : BOOL (default: FALSE)
The device waits until digital input 2 is activated.

DI2Falling : BOOL (default: FALSE)
If TRUE digital input 2 is activated with a falling edge. If FALSE digital input 2 is activated with a rising edge.

DI3 : BOOL (default: FALSE)
The device waits until digital input 3 is activated.

DI3Falling : BOOL (default: FALSE)
If TRUE digital input 3 is activated with a falling edge. If FALSE digital input 3 is activated with a rising edge.

DI4 : BOOL (default: FALSE)
The device waits until digital input 4 is activated.
Note: On the MX2 turbo either DI4 or SER1 must be selected. Both at the same time is unsupported.

DI4Falling : BOOL (default: FALSE)
If TRUE digital input 4 is activated with a falling edge. If FALSE digital input 4 is activated with a rising edge.

Vibration : BOOL (default: FALSE)
The device waits until vibration is detected (see pmSetVibrationSensivity for adjusting sensitivity).

PowerFail : BOOL (default: FALSE)
The device waits until external power is removed.

PowerApply : BOOL (default: FALSE)
The device waits until external power is applied.

GSM : BOOL (default: FALSE)
The device waits until activity is detected on the GSM modem.
Note: Not supported on the RTCU NX-400.

CAN : BOOL (default: FALSE)
The device waits until a message has arrived on the first CAN bus.
The reception of a CAN message is subject to the filtering mechanism. See canFilterCreate().

CAN2 : BOOL (default: FALSE)
The device waits until a message has arrived on the second CAN bus.
The reception of a CAN message is subject to the filtering mechanism. See canFilterCreate().

Note:
Operating at a CAN speed of 1000 Kbps is not supported on X32/NX32. Failing to observe this will result in a return-code of -3.
NX-400 only supports waiting for CAN messages at a CAN speed of 500 Kbps or 1000 Kbps.

SER0 : BOOL (default: FALSE)
The device waits until a byte is received on serial port 0.

SER1 : BOOL (default: FALSE)
The device waits until a byte is received on serial port 1.
Note: On the MX2 turbo either DI4 or SER1 must be selected. Both at the same time is unsupported.

Time : DINT (default: -1)
The device waits a number of seconds.
If Time is -1 or 0, the event will be disabled.

GPS : BOOL (default: FALSE)
The device waits until a 2D or 3D fix has been reported by the GPS module.
This is only supported by the RTCU SX1 series.

Intrusion : BOOL (default: FALSE)
The device waits until an intrusion has been detected.
This is only supported by the RTCU AX9 turbo.

Key : BOOL (default: FALSE)
The device waits until a key has been pressed on the keypad or touchscreen.
This is only supported by the RTCU DX4, RTCU LX4 and the RTCU NX-400.

RF : BOOL (default: FALSE)
The device waits until an RF packet has been received.
This is only supported by devices that has on-board ISM RF.

Returns: **INT**

18	CAN activity on the second CAN bus.
17	RF activity.
16	Key press detected.
15	Intrusion detected.
14	GPS 2D/3D fix.
13	PowerApply.
12	Serial 1 activity.
11	Serial 0 activity.

10	CAN activity on the first CAN bus.
9	GSM activity.
8	Timeout.
7	PowerFail.
6	Vibration detected.
5	Ignition detected.
4	DI4 detected.
3	DI3 detected.
2	DI2 detected.
1	DI1 detected.
0	Error (no event enabled).
-1	Unspecified error.
-2	Specified event is not available due to the feature not being enabled.
-3	Event not legal due to illegal CAN communication speed.
-4	GSM module is OFF but the GSM event has been selected.
-5	GSM module is ON but the GSM event has not been selected.
-6	GSM module is not operating in low power mode (see gsmPowerLP) .
-7	GSM module is busy with an active Voice or CSD session.
-8	GSM module is busy from last call to pmWaitEvent.
-9	The on-board battery is fast-charging
-10	The RF module is open (see rfClose).
-11	GPS module is OFF but the GPS event has been selected.
-12	Serial port 1 and digital input 4 is both selected (MX2 turbo)
-13	External power is disabled. Can not wait for PowerApply or PowerFail.
-14	Ethernet network interface is open.
-15	RF is not open, and RF event is selected.
-16	WiFi network interface is open.
-17	S0 is enabled but there is no external power.
-18	Bluetooth is ON.
-19	Waiting for GSM activity is not supported
-20	CAN is ON but CAN event has not been selected.

Declaration:

```
FUNCTION pmWaitEvent : INT;
VAR_INPUT
   Ignition        : BOOL := FALSE;
   IgnitionFalling : BOOL := FALSE; // FALSE => Rising edge, TRUE => Falling
edge
   DI1             : BOOL := FALSE;
   DI1Falling      : BOOL := FALSE;
   DI2             : BOOL := FALSE;
   DI2Falling      : BOOL := FALSE;
   DI3             : BOOL := FALSE;
   DI3Falling      : BOOL := FALSE;
   DI4             : BOOL := FALSE;
   DI4Falling      : BOOL := FALSE;
   Vibration       : BOOL := FALSE;
   PowerFail       : BOOL := FALSE;
   GSM             : BOOL := FALSE;
   CAN             : BOOL := FALSE;
   CAN2            : BOOL := FALSE;
   SER0            : BOOL := FALSE;
   SER1            : BOOL := FALSE;
   TIME            : DINT := -1;
   PowerApply      : BOOL := FALSE;
   GPS             : BOOL := FALSE;
   Intrusion       : BOOL := FALSE;
   Key             : BOOL := FALSE;
   RF              : BOOL := FALSE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   sw : BOOL;
END_VAR;

PROGRAM test;
VAR
   sms    : gsmIncomingSMS;
   writer : logWrite;
   tmp    : INT;
   sen    : INT := 50;
END_VAR;

pmSetVibrationSensivity(sen:=100);

IF NOT logIsInitialized(key:=4711) THEN
   logInitialize(key:=4711, numlogvalues:=2, numlogrecords:=3000);
END_IF;
DebugMsg(message:="Running...");

BEGIN
   IF sw THEN
      sms();
      IF sms.status > 0 THEN
         tmp := strToInt(str:=sms.message);
         IF tmp > 0 AND tmp < 101 THEN
            IF pmSetVibrationSensivity(sen:=SINT(tmp)) = 0 THEN
               sen := tmp;
               DebugFmt(message:="Vibration sensivity changed to:
\1",v1:=sen);
            END_IF;
         END_IF;
      END_IF;
   ELSE
      // stop execution and wait for one of the following events - All events
except timeout is logged:
      //Ignition rising edge,
      //Digital Input 1 falling edge,
      //Vibration
      //or timeout on 10 seconds.
      tmp := pmWaitEvent(Ignition:=TRUE, DI1:=TRUE, DI1Falling:=TRUE,
Vibration:=TRUE, time:=10);
      IF tmp <> 8 THEN
         writer(tag:=1, value[1]:=sen, value[2]:=tmp);
      END_IF;
   END_IF;
END;

END_PROGRAM;
```

4.2.35.11 pmSuspend (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.36.00

This function enters power-saving mode and effectively freezes program execution and waits for one or several of the enabled wake-up events.

It has support for more advanced wake-up sources and low power modes than pmWaitEvent.

To enable a wake-up event, please use the following functions:
- accVibrationSetWakeup
- boardDinSetWakeup
- boardSupplySetWakeup
- canSetWakeup
- clockSetWakeup
- displayKeySetWakeup
- gsmSetWakeup
- msVibrationSetWakeup
- serSetWakeup

Additionally, it is possible to wake after a specified number of seconds.

When the event(s) happens, the program will continue operation from the location it was called, if the low power mode supports it.

Please note that pmSuspend does not affect the digital outputs. The user must, if desired, deactivate the outputs before calling pmSuspend to reduce the power consumed by the outputs.

It is not possible to use pmSuspend while the on-board battery is fast-charging, and therefore it may be necessary to disable the charger by using batChargerEnable.

pmSuspend return codes for the time wake-up source:

Error	Type	ID	Value	Description
False	1	2	1	Wake-up on time.

Input:

Time : DINT (default: -1)
The device waits a number of seconds.
If Time is -1 or 0, the event will be disabled.
If wake on RTC is enabled, wake on time can not also be enabled.

Mode : SINT (default: 0)
Selects the low power mode to use.

mode	Description	Blinks every ~10s	Continues after wake-up	Resets on wake-up
0	Automatically selects the lowest possible mode that is able to resume.	(X)	X	
1	Similar to pmWaitEvent.	X	X	
2	Lower power consumption than pmWaitEvent.		X	
3	Similar to pmPowerDown.			X

Returns: DINT

>0 The device was suspended and has woken. Use pmGetWakeSource to get details.
0 This function is not supported.
<0 Failed to suspend. Use pmGetWakeSource to get details.

Detailed error codes from pmGetWakeSource for the common errors:

Error	Type	ID	Value	Description
True	0	0	-1	No wake source was selected.
True	0	0	-2	The wake sources do not share a common low power mode.
True	0	0	-3	The mode is not supported on this device.

Declaration:

```
FUNCTION pmSuspend : DINT;
VAR_INPUT
   time            : DINT := -1;
   mode            : SINT := 0;
END_VAR;
```

Example:

```
//------------------------------------------------------------------------------
-
// Example of how to use pmSuspend.
//
//------------------------------------------------------------------------------
-
INCLUDE rtcu.inc

//  Input variables that can be configured via the configuration dialog (These
are global)
VAR_INPUT
   suspend : BOOL; | Set this switch to enable suspend
   absolute_time : BOOL; | Set to true to sleep until the next hour starts,
false will sleep for 10 minutes
END_VAR;

PROGRAM test;
// These are the local variables of the program block
VAR
   rc    : INT;
   wk    : DINT;
   type, id, val : INT;
   err   : BOOL;
END_VAR;
// The next code will only be executed once after the program starts
   // Disable charger as it prevents suspend.
   batChargerEnable(enable := FALSE);

   // Enable wake on DIN 1 rising edge.
   rc := boardDinSetWakeup(pin := 1, rising := TRUE);
   DebugFmt(message:="boardDinSetWakeup: \1", v1 := rc);

   // Enable wake on power apply.
   rc := boardSupplySetWakeup(apply := TRUE);
   DebugFmt(message:="boardSupplySetWakeup: \1", v1 := rc);

BEGIN
// Code from this point until END will be executed repeatedly

   IF suspend THEN
      IF absolute_time THEN
         // Enable wake-up when minute and second are 0, i.e. when the hour
starts.
```

```
            rc := clockSetWakeup(minute := 0, second := 0);
            DebugFmt(message:="clockSetWakeup: \1", v1 := rc);

            // Suspend until an event occurs or until the next hour starts.
            DebugFmt(message:="Calling pmSuspend");
            wk := pmSuspend(time:=-1, mode := 0);
         ELSE
            // Suspend for 10 minutes or until an event occurs.
            DebugFmt(message:="Calling pmSuspend");
            wk := pmSuspend(time:=600, mode := 0);
         END_IF;

         IF wk < 0 THEN
            DebugFmt(message:="pmSuspend failed");
         ELSE
            DebugFmt(message:="pmSuspend succeeded");
         END_IF;

         rc := pmGetWakeSource(source:=wk, error:=err,
type:=type,id:=id,value:=val);
         DebugFmt(message:="$ttype: \1", v1:=type);
         DebugFmt(message:="$tid: \1", v1:=id);
         DebugFmt(message:="$tval: \1", v1:=val);
         DebugFmt(message:=pmGetWakeSourceString(source := wk));
         IF suspend THEN
            // Wait for a few seconds on error
            Sleep(delay:=5000);
         END_IF;
      END_IF;
   END;
END_PROGRAM;
```

4.2.35.12 pmGetWakeSource (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.36.00

This function is used to parse the return code from pmSuspend.
For details about the values, please see the descriptions for the individual wake-up sources.

Input:

Source : DINT
The return code from pmSuspend.

Output:

Error : BOOL
If TRUE the return code represents an error. If FALSE, the return code represents a successful wake-up source.

Type : INT
The module that caused the return code. Some modules supports wake-up while others are only used when they prevent suspend.

Type	Module	Details
0	System	Errors not related to a module

1	Board	ID 1 = RTC
		ID 2 = Timeout (time parameter on pmSuspend)
		ID 3 = Supply change
2	Digital input	ID = input number
3	Serial port	ID = port number
4	Motion sensor	ID 1 = Vibration
5	Cellular	
8	Key press	
9	CAN	ID = Bus number

Additionally, some modules may prevent the system from being suspended and must be powered off first:

Type	Module	Details
1	Board	ID 4 = Battery
6	Network	
7	Bluetooth	

ID : INT
The ID of the wake source. For some modules, this may represent a port or a pin while it in other cases may represent entirely different wake-up sources.

Value : INT
Detailed return code for the module.
Some values are common for all modules and can be found below, the rest can be found in the documentation for the individual wake-up functions.

-1 Unknown
-2 The wake-up source does not support the requested low power mode. There may be other wake sources that require incompatible low power modes.
-3 The module must be powered off before the low power mode can be entered.

Returns: INT

1 The return code was parsed
0 the function is not supported.

Declaration:

```
FUNCTION pmGetWakeSource : INT;
VAR_INPUT
   source          : DINT;
   error           : ACCESS BOOL;
   type            : ACCESS INT;
   id              : ACCESS INT;
   value           : ACCESS INT;
END_VAR;
```

Example:

```
   ...
      wk := pmSuspend(time:=600, mode := 0);
      // Parse return code
      rc := pmGetWakeSource(source:=wk, error:=err,
type:=type,id:=id,value:=val);
      IF err THEN
         DebugFmt(message:="pmSuspend failed");
      ELSE
         DebugFmt(message:="pmSuspend succeeded");
      END_IF;
```

```
      DebugFmt(message:="$ttype: \1", v1:=type);
      DebugFmt(message:="$tid: \1", v1:=id);
      DebugFmt(message:="$tval: \1", v1:=val);
   ...
```

4.2.35.13 pmGetWakeSourceString (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.36.00

This function is used to get a textural description of the return code from pmSuspend.

This is meant to use for debugging and the text may change without notice.
A typical use case would be to pass the string on to debugMsg, to log especially unhandled return codes.

Input:

Source : DINT
The return code from pmSuspend.

Returns: STRING

A string with a description of the wake-up source or the error.

Declaration:

```
FUNCTION pmGetWakeSourceString : STRING;
VAR_INPUT
   source : DINT;
END_VAR;
```

Example:

```
   ...
      wk := pmSuspend(time:=600, mode := 0);
      // Parse return code
      DebugFmt(message:=pmGetWakeSourceString(source := wk));
   ...
```

4.2.36 rest: REST and HTTP functions

4.2.36.1 rest : REST and HTTP functions

The REST and HTTP functions provides a framework for building and using REST APIs as well as for performing generic HTTP requests and running a simple HTTP server.
The REST and HTTP functions are only available in NX32L compilation mode.

The API uses three different types of SYSHANDLE: Server, Request and Response.
Depending on what the needed functionality is, a different set of functions will be needed:

Requesting data from remote server:
A request is created using restReqCreate and the contents of it are set using the different Request functions.
The function restClientRequest then sends the request to a server and receives the response.
The response must then be read using the Response functions.

Making data available to clients:
The server is created using restServerCreate and endpoints are added using restServerEndpointAdd.
The server is started using restServerStart and will now listen for incoming requests on the configured port.
When it receives a request, the callback for the endpoint will be called and the necessary data must be read from the request using the Request functions.
The response must be filled out using the Response functions and it will be sent back to the client.

The following REST and HTTP functions are available:

Server functions:
The server functions are used to manage the REST/HTTP server.

- restServerCreate — Create a REST server.
- restServerFree — Release a REST server.
- restServerEndpointAdd — Create an endpoint on the REST server.
- restServerStart — Starts the REST server.
- restServerStop — Stops the REST server.

Request functions:
The request functions are used to create a request or to extract data from an incoming request.

- restReqCreate — Create a REST request.
- restReqFree — Free a REST request.
- restReqHeaderKey — Read the name of a header field in a request.
- restReqHeaderGet — Read a header field from the REST request.
- restReqHeaderSet — Set the value of a header field in a REST request.
- restReqQueryKey — Read the name of a query field in a request.
- restReqQueryGet — Read a query parameter from the REST request.

- restReqQuerySet Set the value of a query parameter on a REST request.
- restReqPostKey Read the name of a post field in a request.
- restReqPostGet Read a post parameter from the REST request.
- restReqPostSet Set the value of a post parameter on a REST request.
- restReqBasicAuthGet Read the basic authentication provided from the client.
- restReqBasicAuthSet Set the basic authentication to use.
- restReqBodyGetString Get the contents of the body of the REST request as a string.
- restReqBodySetString Set the body content of the REST request to the provided string.
- restReqBodyGetJSON Get the contents of the body of the REST request as a JSON structure.
- restReqBodySetJSON Set the body content of the REST request to the provided JSON structure.
- restReqBodySize Return the size of the body content.
- restReqBodyGetRaw Copy part of the body content into a buffer.
- restReqClientAddressGet Get the IP address of the client that sent the request.
- restReqUrlGet Get the URL and method of the request.

Request Client certificate functions:
The client certificate functions are used to set and examine the client certificate used to authenticate a client with a server.

- restReqClientCertSet Set the client certificate of a request, it will be sent to the server
- restReqClientCertPresent Check if the incoming request contains a client certificate
- restReqClientCertSubjectGet Get the subject of the client certificate
- restReqClientCertSubjectCNGet Get the subject common name(CN) from the client certificate
- restReqClientCertIssuerGet Get the issuer of the client certificate
- restReqClientCertVersionGet Get the version of the client certificate format
- restReqClientCertValidFrom Get the start of the valid date range of the client certificate
- restReqClientCertValidTo Get the end of the valid date range of the client certificate
- restReqClientCertSerialGet Get the serial number of the client certificate
- restReqClientCertFingerprintGet Get the fingerprint of the client certificate
- restReqClientCertSANGet Get a Subject Alternative Name (SAN) for the client certificate.
- restReqClientCertCheckHostname Check if the subject of the certificate matches the provided hostname.
- restReqClientCertCheckEmail Check if the certificate matches the provided email address.

Response functions:

The response functions are used to extract data from a response to a request or to build the response to send for a request.

- restRespFree Free a REST response.
- restRespBodySetString Set the body content of the REST response to the provided string.
- restRespBodyGetString Get the contents of the body of the REST response as a string.
- restRespBodySetJSON Set the body content of the REST response to the provided JSON structure.
- restRespBodyGetJSON Get the contents of the body of the REST response as a JSON structure.
- restRespBodySize Return the size of the body content.
- restRespBodyGetRaw Copy part of the body content into a buffer.
- restRespHeaderKey Read the name of a header field in a response.
- restRespHeaderGet Read a header field from the REST response.
- restRespHeaderSet Set the value of a header field in a REST response.
- restRespBasicAuthSet Make the response request basic authentication on status 401.

Client functions:

- restClientRequest Execute a REST client request and wait for the response.

4.2.36.2 Server functions

4.2.36.2.1 restServerCreate (Function)

```
Architecture:       NX32L
Device support:     ALL
Firmware version:   2.10.00
```

This function will create a new REST server.
To free the server again, call restServerFree.
A maximum of two servers can be created at the same time.

Input:

port : DINT
The port to listen on.

iface : SINT *(default 0)*
The network interface to listen on. 0 = Any network, 1 = Mobile network, 2 = LAN network, etc. (See Network).
When listening to the WLAN interface, it may be necessary to wait until the network is connected before starting the server with restServerStart.

cert : STRING
The name of the server certificate to use with TLS. Leave empty to not use TLS.
See Certificates for more details about how to work with the certificates.

key : STRING
Private key for server certificate. Leave empty if not used or if it is included in the server certificate.

key_pass : STRING
Password for the private key.

accept_ca: STRING
The name of a root certificate. If it is specified, the clients are asked to provide a client certificate signed by it. The client certificate will be made available as part of the request, see restReqClientCertPresent.

Output:
handle : SYSHANDLE
A handle to the server.

Returns: INT
1 - Success
0 - Not supported
-2 - No more servers can be created before one is freed.
-3 - Failed to create server.
-6 - Can not read certificate or key.

Declaration:
```
FUNCTION restServerCreate : INT;
VAR_INPUT
   handle    : ACCESS SYSHANDLE;
   port      : DINT;
   iface     : SINT;
   cert      : STRING;
   key       : STRING;
   accept_ca : STRING;
   key_pass  : STRING;
END_VAR;
```

Example:
Please see the "Examples - REST Example"

4.2.36.2.2 restServerFree (Function)

```
Architecture:       NX32L
Device support:     ALL
Firmware version:   2.10.00
```

This function will release a REST server.
If the server is currently running it will be stopped.

Input:
handle : SYSHANDLE
A handle to the server.

Returns: INT

1 - Success
0 - Not supported
-1 - Invalid handle

Declaration:

```
FUNCTION restServerFree : INT;
VAR_INPUT
   handle    : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   rc       : INT;
   server   : SYSHANDLE;
END_VAR
BEGIN

   rc := restServerCreate(handle := server, port := 8080);
   ...
   rc := restServerFree(handle := server);
END;
END_PROGRAM;
```

4.2.36.2.3 restServerEndpointAdd (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

Create an endpoint on the REST server.

Endpoints are used to provide services to the clients.
Endpoints specifies which URLs should be mapped to different callbacks.
It is possible to add endpoints to a running server, but the server must have at least one endpoint to start.
A server can have up to 128 endpoints.

Note that the request provided to the callback can only be read and may not be freed or used outside the callback.

Incoming requests are limited to a maximum of 20 kB per post parameter and a max body size of 1 MB.

Multiple prioritized endpoints for similar URLs:

If there is shared code that should be run for multiple requests, e.g. for validate the user, it is possible to configure it so the request calls multiple endpoints in a sequence.
The url_prefix and url_format parameters are used to control which endpoints will be called depending on the URL. The variables specified in url_format will be extracted from the URL and made available as part of the query parameters.
The prio parameter is used to control the priority of the endpoint, which controls the order the matching endpoints are called in.
When a endpoint callback does not use 0 as return code, no more callbacks will be called for the request, skipping the endpoints with lower priority.

An example uses a server is configured to use the following endpoints:

ID	url_prefix	url_format	prio	Description
1	/test	/*	0	Matches everything, called as the first because prio=0. Suitable for access control.
2	/test	/*	5	Same as #1, called last as prio=5. Suitable for a default response.
3	/test	/:a	2	Matches URL with 1 segment after the prefix. Variable "a" will be set to the value of it.
4	/test	/:a/:b	2	Matches URL with 2 segments after the prefix. Variables "a" and "b" will be set to the values of them.
5	/test	/:a/:b/*	2	Matches URL with 2 or more segments after the prefix. Variables "a" and "b" will be set to the value of the first two.
6	/test	/:a/4	1	Matches URL with 2 segments after the prefix, the second one must be "4". Variable "a" will be set to the value of the first segment. Higher priority so it is called before the more generic endpoints.

The callbacks all return 0, so all the matching endpoints will be called.
Accessing the following URLs will trigger the endpoints in the shown order:

URL	Endpoints
/test	1 2
/test/1	1 3 2
/test/1/2	1 4 5 2
/test/1/2/3	1 5 2
/test/1/4	1 4 5 6 2
/test/1/4/5	1 5 2

As can be seen, endpoint 1 and 2 are called for all the requests, while e.g. endpoint 6 is only called when there are two segments after the prefix and the last segment is "4".

Input:

handle : SYSHANDLE
A handle to the server.

method : STRING (default "*")
The HTTP method to handle with this endpoint. Typical methods are "GET", "POST "and "PUT". Use "*" to handle any method.

url_prefix : STRING
This is an optional fixed part of the URL that must match the start of the request URL.

url_format : STRING
The format of the endpoint URL, which can include variables that will be extracted.
Separate words with "/". Prefix variables with "@" or ":".
End url_format with "*" to not test the rest of the URL.

The variables can be read using restReqQueryGet.

prio : INT
The priority of the endpoint. 0 is the highest priority.
This is needed when multiple endpoints might match the same URL and to call multiple callbacks for the same request, to e.g. authenticate the user in one callback and then generate the data in another.

cb_func : CALLBACK
Callback to call when the endpoint is called.

It is called with a handle to the request from the client, a handle to an empty response to fill out and with the value of the user argument.
A positive return value is used as the HTTP status code, e.g. 200 (OK) for success, 404 (Not Found) if something is not found.
A return value of 0 will cause it to continue to the next callback for the same endpoint.
A return value of -1 will cause it to complete the transaction and send status code 401 (Unauthorized) to the client, requesting basic authentication from the client, see restReqGetBasicAuthenthication.
A return value of -2 will cause it to complete the transaction and send status code 500 (Internal Server Error) to the client.
As the response can not be sent before this callback has returned, it should try to return as quickly as possible.

cb_arg : DINT
User argument to pass on to the callback.

Returns: INT

1	- Success
0	- Not supported
-1	- Invalid server.
-2	- The maximum number of endpoints is already in use.
-3	- Failed to create endpoint.

Declaration:

```
FUNCTION restServerEndpointAdd : INT;
VAR_INPUT
   handle     : SYSHANDLE;
   method     : STRING := "*";
   url_prefix : STRING;
   url_format : STRING;
   prio       : INT;
   cb_func    : CALLBACK;
   cb_arg     : DINT;
END_VAR;
```

Callback declaration:

```
FUNCTION CALLBACK restServerCallback : INT;
VAR_INPUT
   req     : SYSHANDLE;
   resp    : SYSHANDLE;
   arg     : DINT;
END_VAR;
```

Example:

```
//------------------------------------------------------------------------------
-
//
// Starts a server with multiple prioritized endpoints.
//
//------------------------------------------------------------------------------
-
INCLUDE rtcu.inc

//  These are the global variables of the program
VAR
   serv: SYSHANDLE;
END_VAR;
```

```
FUNCTION CALLBACK devicesGetCallback: INT;
VAR_INPUT
   req         : SYSHANDLE; // Request
   resp        : SYSHANDLE; // Response
   arg         : DINT;      // User argument
END_VAR;
VAR
   str, name, url:STRING;
   body  : STRING := "";
   i     : INT;
   rc    : INT;
END_VAR;
   // Show client address
   restReqClientAddressGet (req:=req, address:=str);
   DebugFmt(message:="Request type \4 from "+str, v4:=arg);

   // Read body content from previous callback, if it is present
   restRespBodyGetString(resp:=resp, str:=body);

   // Add header for this callback to body
   body := body+" -- EP "+dintToStr(v:=arg)+" -- $N";

   // Get request URL
   rc := restReqUrlGet(req:=req, method:=str, url:=url);
   IF rc > 0 THEN
      DebugFmt(message:=" "+str+" "+url);
      body:=body+str+" "+url+"$N";
   ELSE
      DebugFmt(message:="restReqUrlGet: \1", v1:=rc);
   END_IF;


   // Read query parameters
   DebugFmt(message:=" Query:");
   body:=body+"Query:$N";
   i:=0;
   rc := restReqQueryKey(req:=req, idx:=i, name:=name);
   WHILE rc > 0 DO
      restReqQueryGet(req:=req, name:=name, value:=str);
      body:=body+name+"="+str+"$N";
      DebugMsg(message:=" "+name+"="+str);
      i:=i+1;
      rc := restReqQueryKey(req:=req, idx:=i, name:=name);
   END_WHILE;

    body:=body+"$N";

   // Set response body
   restRespBodySetString(resp:=resp, str := body);
   // Set content-type
   restRespHeaderSet(resp:=resp, name:="Content-Type", value:="text/plain");

   // Return 0 to continue with the next callback.
   // Return status code to stop and return the response.
   devicesGetCallback := 0;

END_FUNCTION;



PROGRAM ep_ex;
// These are the local variables of the program block
VAR
   iface : SINT := 2;
```

```
   port  : DINT := 80;
   rc    : INT;
END_VAR;
// The next code will only be executed once after the program starts
   netOpen(iface:=iface);

   rc := restServerCreate(handle:=serv, port:=port);
   DebugFmt(message:="restServerCreate: \1", v1:=rc);

   rc := restServerEndpointAdd(handle:=serv, method:="*",
cb_func:=@devicesGetCallback,
   url_prefix:="/test",url_format:= "/*", cb_arg:=1, prio:=0);
   DebugFmt(message:="restServerEndpointAdd: \1", v1:=rc);

   rc := restServerEndpointAdd(handle:=serv, method:="*",
cb_func:=@devicesGetCallback,
   url_prefix:="/test",url_format:= "/*", cb_arg:=2, prio:=5);
   DebugFmt(message:="restServerEndpointAdd: \1", v1:=rc);

   rc := restServerEndpointAdd(handle:=serv, method:="*",
cb_func:=@devicesGetCallback,
   url_prefix:="/test",url_format:= "/:a", cb_arg:=3, prio:=2);
   DebugFmt(message:="restServerEndpointAdd: \1", v1:=rc);

   rc := restServerEndpointAdd(handle:=serv, method:="*",
cb_func:=@devicesGetCallback,
   url_prefix:="/test",url_format:= "/:a/:b", cb_arg:=4, prio:=2);
   DebugFmt(message:="restServerEndpointAdd: \1", v1:=rc);

   rc := restServerEndpointAdd(handle:=serv, method:="*",
cb_func:=@devicesGetCallback,
   url_prefix:="/test",url_format:= "/:a/:b/*", cb_arg:=5, prio:=2);
   DebugFmt(message:="restServerEndpointAdd: \1", v1:=rc);

   rc := restServerEndpointAdd(handle:=serv, method:="*",
cb_func:=@devicesGetCallback,
   url_prefix:="/test",url_format:= "/:a/4", cb_arg:=6, prio:=1);
   DebugFmt(message:="restServerEndpointAdd: \1", v1:=rc);


   WHILE NOT netConnected(iface:=iface) DO
      Sleep(delay:=1000);

   END_WHILE;

   rc := restServerStart(handle:=serv);
   DebugFmt(message:="restServerStart: \1", v1:=rc);

   DebugFmt(message:="http://"+sockIPToName(ip :=
sockGetLocalIP(iface:=iface))+":\4/test", v4:=port);



BEGIN
// Code from this point until END will be executed repeatedly

END;

END_PROGRAM;
```

4.2.36.2.4 restServerStart (Function)

```
Architecture:       NX32L
Device support:     ALL
Firmware version:   2.10.00
```

This function will start the REST server.
To stop the server, call restServerStop.

Input:

handle : SYSHANDLE
A handle to the server to start.

Returns: INT

1	- Success
0	- Not supported
-1	- Invalid server.
-3	- Failed to start server.
-4	- Invalid parameter, certificate or key might not work.
-12	- Failed to bind to interface, interface might not be ready. When binding to WLAN, it might be necessary to wait for the network to be connected before starting the server.
-26	- Failed to start server, port might already be in use.
-27	- Server is already running.

Declaration:

```
FUNCTION restServerStart : INT;
VAR_INPUT
   handle    : SYSHANDLE;
END_VAR;
```

Example:

Please see the "Examples - REST Example"

4.2.36.2.5 restServerStop (Function)

```
Architecture:       NX32L
Device support:     ALL
Firmware version:   2.10.00
```

This function will stop the REST server.

Input:

handle : SYSHANDLE
A handle to the server to stop.

Returns: INT

1	- Success
0	- Not supported
-1	- Invalid server.
-3	- Failed to stop server.
-27	- Server is not running

Declaration:

```
FUNCTION restServerStop : INT;
VAR_INPUT
   handle    : SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   rc       : INT;
   server   : SYSHANDLE;
END_VAR
BEGIN

   rc := restServerCreate(handle := server, port := 8080);
   ...
   rc := restServerStart(handle := server);
   ...
   rc := restServerStop(handle := server);
   rc := restServerFree(handle := server);
END;
END_PROGRAM;
```

4.2.36.3 Request functions

4.2.36.3.1 restReqCreate (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

This function will create a new request that can be executed by calling restClientRequest.
To free the request when it is no longer needed, call restReqFree.
This request can both be written and read.
There can maximum be 20 requests at the same time. This includes requests used for the server endpoint callbacks, see restServerEndpointAdd.

Input:

method : STRING *(default "GET")*
The HTTP method to use for the request.

url : STRING
The URL of the request, including the protocol ("http://" or "https://").

check_server: BOOL *(default TRUE)*
Set to FALSE to not validate the server certificate when using TLS.

check_hostname : BOOL
Set to FALSE to not validate the hostname of the server when using TLS. This is useful if the server uses a dynamic IP address so the server certificate is not able to match the hostname.

follow_redirect: BOOL *(default FALSE)*
Set to TRUE to follow HTTP redirects, which could cause it to receive content that does not match the expected content. By leaving it at FALSE, any redirect response will be returned and can be manually handled.

timeout: INT ***(default 20)***
The timeout in seconds before canceling the request.

Output:
req : SYSHANDLE
A handle to the new request.

Returns: INT
1 - Success
0 - Not supported
-2 - The max number of requests have already been created.
-3 - Failed to create request.

Declaration:
```
FUNCTION restReqCreate : INT;
VAR_INPUT
   req            : ACCESS SYSHANDLE;
   method         : STRING := "GET";
   url            : STRING;
   check_server   : BOOL := TRUE;
   check_hostname : BOOL := TRUE;
   timeout        : INT := 20;
   follow_redirect: BOOL;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   rc    : INT;
   req   : SYSHANDLE;
END_VAR
BEGIN

   rc := restReqCreate(req := req, url := "https://www.example.com");
   ...
   rc := restReqFree(req := req);
END;
END_PROGRAM;
```

4.2.36.3.2 restReqFree (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

This function will release a REST request.

Input:
req : SYSHANDLE
A handle to the request.

Returns: INT

1 - Success
0 - Not supported
-1 - Invalid request

Declaration:

```
FUNCTION restReqFree : INT;
VAR_INPUT
   req    : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   rc    : INT;
   req   : SYSHANDLE;
END_VAR
BEGIN

   rc := restReqCreate(req := req, url := "https://www.example.com");
   ...
   rc := restReqFree(req := req);
END;
END_PROGRAM;
```

4.2.36.3.3 restReqHeaderKey (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

This function will read the name of a header field in a request.
By calling this multiple times with incrementing index, it is possible to list all the header fields in the request.

Input:

req : SYSHANDLE
A handle to the request.

idx : INT *(Default 0)*
The index of the header field to read the name of.

Output:

name : STRING
The name of the field.

Returns: INT

1 - Success
0 - Not supported
-1 - Invalid request
-2 - The header field was not found

Declaration:

```
FUNCTION restReqHeaderKey : INT;
VAR_INPUT
```

```
   req            : SYSHANDLE;
   idx            : INT;
   name           : ACCESS STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   rc        : INT;
   str, name : STRING;
   i         : INT;
   req       : SYSHANDLE;
END_VAR
BEGIN
   ...

   i:=0;
   // Get name of header
   rc := restReqHeaderKey(req:=req, idx:=i, name:=name);
   WHILE rc > 0 DO
      // Get value of header
      restReqHeaderGet(req:=req, name:=name, value:=str);
      DebugMsg(message:=" "+name+"="+str);
      i:=i+1;
      // Get name of next header
      rc := restReqHeaderKey(req:=req, idx:=i, name:=name);
   END_WHILE;
   ...
END;
END_PROGRAM;
```

4.2.36.3.4 restReqHeaderGet (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

This function will read a header field from a request.

Input:

req : SYSHANDLE
A handle to the request.

name : STRING
The name of the header field to read.
Common fields include "Content-Type", "Accept" and "User-Agent".

check_case : BOOL *(Default FALSE)*
If set to TRUE, the name will use case sensitive comparison.
If left at FALSE, the name will use case insensitive comparison.
Note that if the same name is present with different case, only the value for the first matching name will be returned.

Output:

value : STRING
The value of the field.
If a value is specified multiple times, all the values will be provided, separated by commas.

Returns: INT

1 - Success
0 - Not supported
-1 - Invalid request
-2 - The header was not found
-7 - The value was too large for a string.

Declaration:

```
FUNCTION restReqHeaderGet : INT;
VAR_INPUT
   req            : SYSHANDLE;
   name           : STRING;
   check_case     : BOOL := FALSE;
   value          : ACCESS STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   rc        : INT;
   str, name : STRING;
   i         : INT;
   req       : SYSHANDLE;
END_VAR
BEGIN
   ...

   i:=0;
   // Get name of header
   rc := restReqHeaderKey(req:=req, idx:=i, name:=name);
   WHILE rc > 0 DO
      // Get value of header
      restReqHeaderGet(req:=req, name:=name, value:=str);
      DebugMsg(message:=" "+name+"="+str);
      i:=i+1;
      // Get name of next header
      rc := restReqHeaderKey(req:=req, idx:=i, name:=name);
   END_WHILE;
   ...
END;
END_PROGRAM;
```

4.2.36.3.5 restReqHeaderSet (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

This function will set the value of a header field in a request.

Input:

req : SYSHANDLE
A handle to the request.

name : STRING
The name of the header field to set.
Common fields include "Content-Type", "Accept" and "User-Agent".

value : STRING
The new value of the field.

Returns: INT

1	- Success
0	- Not supported
-1	- Invalid request
-3	- Failed to set header
-5	- Request is read only.

Declaration:

```
FUNCTION restReqHeaderSet: INT;
VAR_INPUT
   req            : SYSHANDLE;
   name           : STRING;
   value          : STRING;
END_VAR;
```

Example:

Please see the "Examples - REST Example"

4.2.36.3.6 restReqQueryKey (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

This function will read the name of a query field in a request.
By calling this multiple times with incrementing index, it is possible to list all the query fields in the request.

Input:

req : SYSHANDLE
A handle to the request.

idx : INT *(Default 0)*
The index of the query field to read the name of.

Output:

name : STRING
The name of the field.

Returns: INT

1	- Success

0 - Not supported
-1 - Invalid request
-2 - The query field was not found

Declaration:

```
FUNCTION restReqQueryKey : INT;
VAR_INPUT
   req            : SYSHANDLE;
   idx            : INT;
   name           : ACCESS STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   rc        : INT;
   str, name : STRING;
   i         : INT;
   req       : SYSHANDLE;
END_VAR
BEGIN
   ...

   i:=0;
   // Get name of query
   rc := restReqQueryKey(req:=req, idx:=i, name:=name);
   WHILE rc > 0 DO
      // Get value of query
      restReqQueryGet(req:=req, name:=name, value:=str);
      DebugMsg(message:=" "+name+"="+str);
      i:=i+1;
      // Get name of next query
      rc := restReqQueryKey(req:=req, idx:=i, name:=name);
   END_WHILE;
   ...
END;
END_PROGRAM;
```

4.2.36.3.7 restReqQueryGet (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

This function will read a query field from a request.
This function is both used to read values specified in the query string, i.e. `"/index?id=xx&value=yy"` and to read variables extracted from the URL according to the format specified in restServerEndpointAdd.
If a value is specified multiple times, all the values will be provided, separated by commas.

Input:

req : SYSHANDLE
A handle to the request.

name : STRING
The name of the query field to read.

check_case : BOOL *(Default FALSE)*
If set to TRUE, the name will use case sensitive comparison.
If left at FALSE, the name will use case insensitive comparison.
Note that if the same name is present with different case, only the value for the first matching name will be returned.

Output:

value : STRING
The value of the field.

Returns: INT

1	- Success
0	- Not supported
-1	- Invalid request
-2	- The query was not found
-7	- The value was too large for a string.

Declaration:

```
FUNCTION restReqQueryGet : INT;
VAR_INPUT
   req            : SYSHANDLE;
   name           : STRING;
   check_case     : BOOL := FALSE;
   value          : ACCESS STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   rc         : INT;
   str, name  : STRING;
   i          : INT;
   req        : SYSHANDLE;
END_VAR
BEGIN
   ...

   i:=0;
   // Get name of query
   rc := restReqQueryKey(req:=req, idx:=i, name:=name);
   WHILE rc > 0 DO
      // Get value of query
      restReqQueryGet(req:=req, name:=name, value:=str);
      DebugMsg(message:=" "+name+"="+str);
      i:=i+1;
      // Get name of next query
      rc := restReqQueryKey(req:=req, idx:=i, name:=name);
   END_WHILE;
   ...
END;
END_PROGRAM;
```

4.2.36.3.8 restReqQuerySet (Function)

```
Architecture:       NX32L
Device support:     ALL
Firmware version:   2.10.00
```

This function will set the value of a query field in a request.

Input:

req : SYSHANDLE
A handle to the request.

name : STRING
The name of the query field to set.

value : STRING
The new value of the field.

Returns: INT

1	- Success
0	- Not supported
-1	- Invalid request
-3	- Failed to set query
-5	- Request is read only.

Declaration:

```
FUNCTION restReqQuerySet: INT;
VAR_INPUT
   req             : SYSHANDLE;
   name            : STRING;
   value           : STRING;
END_VAR;
```

Example:

Please see the "Examples - REST Example"

4.2.36.3.9 restReqPostKey (Function)

```
Architecture:       NX32L
Device support:     ALL
Firmware version:   2.10.00
```

This function will read the name of a post parameter in a request.
By calling this multiple times with incrementing index, it is possible to list all the post parameters in the request.
Note: Transferring files as post parameters is not currently supported.

Input:

req : SYSHANDLE
A handle to the request.

idx : INT *(Default 0)*
The index of the post parameter to read the name of.

Output:

name : STRING
The name of the parameter.

Returns: INT

1 - Success
0 - Not supported
-1 - Invalid request
-2 - The post parameter was not found

Declaration:

```
FUNCTION restReqPostKey : INT;
VAR_INPUT
   req            : SYSHANDLE;
   idx            : INT;
   name           : ACCESS STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   rc        : INT;
   str, name : STRING;
   i         : INT;
   req       : SYSHANDLE;
END_VAR
BEGIN
   ...

   i:=0;
   // Get name of post parameter
   rc := restReqPostKey(req:=req, idx:=i, name:=name);
   WHILE rc > 0 DO
      // Get value of post parameter
      restReqPostGet(req:=req, name:=name, value:=str);
      DebugMsg(message:=" "+name+"="+str);
      i:=i+1;
      // Get name of next post parameter
      rc := restReqPostKey(req:=req, idx:=i, name:=name);
   END_WHILE;
   ...
END;
END_PROGRAM;
```

4.2.36.3.10 restReqPostGet (Function)

```
Architecture:       NX32L
Device support:     ALL
Firmware version:   2.10.00
```

This function will read a post parameter from a request.
If a value is specified multiple times, all the values will be provided, separated by commas.

Input:

req : SYSHANDLE
A handle to the request.

name : STRING
The name of the post parameter to read.

check_case : BOOL *(Default FALSE)*
If set to TRUE, the name will use case sensitive comparison.
If left at FALSE, the name will use case insensitive comparison.
Note that if the same name is present with different case, only the value for the first matching name will be returned.

Output:

value : STRING
The value of the parameter.

Returns: INT

1	- Success
0	- Not supported
-1	- Invalid request
-2	- The query was not found
-7	- The value was too large for a string.

Declaration:

```
FUNCTION restReqPostGet : INT;
VAR_INPUT
   req            : SYSHANDLE;
   name           : STRING;
   check_case     : BOOL := FALSE;
   value          : ACCESS STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   rc        : INT;
   str, name : STRING;
   i         : INT;
   req       : SYSHANDLE;
END_VAR
BEGIN
   ...

   i:=0;
   // Get name of post parameter
   rc := restReqPostKey(req:=req, idx:=i, name:=name);
   WHILE rc > 0 DO
      // Get value of post parameter
      restReqPostGet(req:=req, name:=name, value:=str);
      DebugMsg(message:=" "+name+"="+str);
      i:=i+1;
      // Get name of next post parameter
      rc := restReqPostKey(req:=req, idx:=i, name:=name);
   END_WHILE;
   ...
```

```
END;
END_PROGRAM;
```

4.2.36.3.11 restReqPostSet (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

This function will set the value of a post parameter in a request.
If the request contains post parameters, it can not also contain a normal body, as set with restReqBodySetJSON or restReqBodySetString.

Input:

req : SYSHANDLE
A handle to the request.

name : STRING
The name of the post parameter to set.

value : STRING
The new value of the parameter.

Returns: INT

1 - Success
0 - Not supported
-1 - Invalid request
-3 - Failed to set query
-5 - Request is read only.

Declaration:

```
FUNCTION restReqPostSet: INT;
VAR_INPUT
   req             : SYSHANDLE;
   name            : STRING;
   value           : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   rc         : INT;
   req        : SYSHANDLE;
END_VAR
BEGIN
   ...

   // Set post parameter "post1" to "Value"
   rc := restReqPostSet(req:=req, name:="post1", value:="Value");
   DebugFmt(message:="restReqPostSet: \1", v1:=rc);
   ...
END;
END_PROGRAM;
```

4.2.36.3.12 restReqBasicAuthGet (Function)

```
Architecture:       NX32L
Device support:     ALL
Firmware version:   2.10.00
```

This function will read the basic authentication of a request.
To tell the client that it must send basic authentication, use restRespBasicAuthSet and status code 401.

Input:

req : SYSHANDLE
A handle to the request.

Output:

user : STRING
The user name. Empty if no user name was provided.

pass : STRING
The password. Empty if no password was provided.

Returns: INT

1 - Success
0 - Not supported
-1 - Invalid request

Declaration:

```
FUNCTION restReqBasicAuthGet : INT;
VAR_INPUT
   req            : SYSHANDLE;
   user           : ACCESS STRING;
   pass           : ACCESS STRING;
END_VAR;
```

Example:

Please see the "Examples - REST Basic Authentication Example"

4.2.36.3.13 restReqBasicAuthSet (Function)

```
Architecture:       NX32L
Device support:     ALL
Firmware version:   2.10.00
```

This function will set the basic authentication to use for a request.

Input:

req : SYSHANDLE
A handle to the request.

user : STRING
The user name to use.

pass : STRING
The password to use.

Returns: INT

1 - Success
0 - Not supported
-1 - Invalid request
-3 - Failed to set authentication
-5 - Request is read only.

Declaration:

```
FUNCTION restReqBasicAuthSet: INT;
VAR_INPUT
   req            : SYSHANDLE;
   user           : STRING;
   pass           : STRING;
END_VAR;
```

Example:

Please see the "Examples - REST Basic Authentication Example"

4.2.36.3.14 restReqBodyGetString (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

This function will read a the body contents of a request as a string.

Input:

req : SYSHANDLE
A handle to the request.

Output:

str : STRING
The body content of the request as a string.

Returns: INT

1 - Success
0 - Not supported
-1 - Invalid request
-2 - The body is too large to fit in a string.
-20 - The body was larger than the maximum body size.

Declaration:

```
FUNCTION restReqBodyGetString : INT;
VAR_INPUT
   req            : SYSHANDLE;
   str            : ACCESS STRING;
END_VAR;
```

Example:

Please see the "Examples - REST Example"

4.2.36.3.15 restReqBodySetString (Function)

```
Architecture:       NX32L
Device support:     ALL
Firmware version:   2.10.00
```

This function will set the body content of a request to the provided string.
When the body needs to contain mostly fixed data, one option is to use strTemplateCreate to add the dynamic data to a fixed template string, which can then be used as the body, without needing to manually create the entire string.

Input:

req : SYSHANDLE
A handle to the request.

str : STRING
The string to use for the request body.

Returns: INT

1 - Success
0 - Not supported
-1 - Invalid request
-3 - Failed to set query
-5 - Request is read only.

Declaration:

```
FUNCTION restReqBodySetString: INT;
VAR_INPUT
   req            : SYSHANDLE;
   str            : STRING;
END_VAR;
```

Example:

Please see the "Examples - REST Basic Authentication Example"

4.2.36.3.16 restReqBodyGetJSON (Function)

```
Architecture:       NX32L
Device support:     ALL
Firmware version:   2.10.00
```

This function will read the contents of the body of the request as a JSON structure.

Input:

req : SYSHANDLE
A handle to the request.

Output:

json : SYSHANDLE
A handle to the JSON structure created from the body content.
When the JSON structure is no longer needed, it must be released using jsonFree.

Returns: INT

1 - Success
0 - Not supported
-1 - Invalid request
-2 - The max number of JSON structures have already been created.
-3 - Failed to create JSON structure. Body content might not be valid JSON.

Declaration:

```
FUNCTION restReqBodyGetJSON : INT;
VAR_INPUT
   req             : SYSHANDLE;
   json            : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
FUNCTION CALLBACK devicesGetCallback: INT;
VAR_INPUT
   req         : SYSHANDLE; // Request
   resp        : SYSHANDLE; // Response
   arg         : DINT;      // User argument
END_VAR;
VAR
   json  : SYSHANDLE;
   rc    : INT;
END_VAR;
   ...
   // Read body content from previous callback, if it is present
   rc := restRespBodyGetJSON(resp:=resp, json:=json);

   ...

   // Release json
   rc := jsonFree(o:=json);
   ...

END_FUNCTION;
```

4.2.36.3.17 restReqBodySetJSON (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

This function will set the body content of the request to the provided JSON structure.

Input:

req : SYSHANDLE
A handle to the request.

json : SYSHANDLE
The JSON structure to use for the body content.

Returns: INT

1	- Success
0	- Not supported
-1	- Invalid request
-3	- Failed to convert JSON
-4	- Invalid JSON handle.
-5	- Request is read only.
-20	- Failed to set body.

Declaration:

```
FUNCTION restReqBodySetJSON: INT;
VAR_INPUT
   req            : SYSHANDLE;
   json           : SYSHANDLE;
END_VAR;
```

Example:

Please see the "Examples - REST Example"

4.2.36.3.18 restReqBodySize (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

This function will return the size of the body content in the request.

Input:

req : SYSHANDLE
A handle to the request.

Returns: DINT

>=0	- Size of request body in bytes
0	- Not supported
-1	- Invalid response
-2	- The request body is too large to report.

Declaration:

```
FUNCTION restReqBodySize : DINT;
VAR_INPUT
   req            : SYSHANDLE;
END_VAR;
```

Example:

Please see the "Examples - REST Basic Authentication Example"

4.2.36.3.19 restReqBodyGetRaw (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

This function will copy a part the body contents from the request into the provided buffer.

Input:

req : SYSHANDLE
A handle to the request.

dst : PTR
A pointer to the buffer to place the body content in.

start : DINT *(default 0)*
The offset to start copying from, starting at 0.

size : DINT
The maximum length to read, i.e. the size of the buffer.

Returns: DINT

>0	- Success, number of bytes copied.
0	- Not supported
-1	- Invalid request
-2	- Start is past the end of the body.
-4	- Size, start or the destination is invalid.
-20	- Body was too large to keep, it is not available.

Declaration:

```
FUNCTION restReqBodyGetRaw : DINT;
VAR_INPUT
   req             : SYSHANDLE;
   dst             : PTR;
   start           : DINT;
   size            : DINT;
END_VAR;
```

Example:

Please see the "Examples - REST Basic Authentication Example"

4.2.36.3.20 restReqClientAddressGet (Function)

```
Architecture:       NX32L
Device support:     ALL
Firmware version:   2.10.00
```

This function will get the address of the client that created the request.

Input:

req : SYSHANDLE
A handle to the request.

Output:

address : STRING
The socket address of the client connection in the same format as used by the advanced socket API.

Returns: INT

1 - Success
0 - Not supported
-1 - Invalid request
-2 - Request does not have a client address.
-21 - Unsupported address.

Declaration:

```
FUNCTION restReqClientAddressGet : INT;
VAR_INPUT
   req             : SYSHANDLE;
   address         : ACCESS STRING;
END_VAR;
```

Example:

Please see the "Examples - REST Example"

4.2.36.3.21 restReqUrlGet (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

This function will read the URL and method of a request.
For outgoing requests, this is the URL and method specified in restReqCreate.
For incoming requests, this is the relative URL and method used by the client, including any query parameters.

Input:

req : SYSHANDLE
A handle to the request.

Output:

method : STRING
The HTTP method of the request, e.g. "GET" or "POST".

url : STRING
The URL of the request.

Returns: INT

1 - Success
0 - Not supported
-1 - Invalid request
-2 - Request does not contain an URL

Declaration:

```
FUNCTION restReqUrlGet : INT;
VAR_INPUT
   req             : SYSHANDLE;
   method          : ACCESS STRING;
   url             : ACCESS STRING;
END_VAR;
```

Example:

```
FUNCTION CALLBACK devicesGetCallback: INT;
VAR_INPUT
   req         : SYSHANDLE; // Request
   resp        : SYSHANDLE; // Response
   arg         : DINT;      // User argument
END_VAR;
VAR
   str, name, url:STRING;
   body  : STRING := "";
   rc    : INT;
END_VAR;
   // Show client address
   restReqClientAddressGet (req:=req, address:=str);
   DebugFmt(message:="Request type \4 from "+str, v4:=arg);

   // Read body content from previous callback, if it is present
   restRespBodyGetString(resp:=resp, str:=body);

   // Add header for this callback to body
   body := body+" -- EP "+dintToStr(v:=arg)+" -- $N";

   // Get request URL
   rc := restReqUrlGet(req:=req, method:=str, url:=url);
   IF rc > 0 THEN
      DebugFmt(message:=" "+str+" "+url);
      body:=body+str+" "+url+"$N";
   ELSE
      DebugFmt(message:="restReqUrlGet: \1", v1:=rc);
   END_IF;

   // Set response body
   restRespBodySetString(resp:=resp, str := body);
   // Set content-type
   restRespHeaderSet(resp:=resp, name:="Content-Type", value:="text/plain");

   // Return 0 to continue with the next callback.
   // Return status code to stop and return the response.
   devicesGetCallback := 0;

END_FUNCTION;
```

4.2.36.4 Certificate functions

4.2.36.4.1 restReqClientCertSet (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

This function sets the client certificate on a request that will be sent to the server when calling restClientRequest.
Note that the certificate set with this function can not be read using the other restReqClientCert functions, as they only work on incoming requests.

Input:

req : SYSHANDLE
A handle to the request.

cert : STRING
The name of the client certificate. See Certificates for more details about how to work with the certificates.

key : STRING
The name of the private key for the client certificate. If not provided, the key must be included in the client certificate.

pass : STRING
The password for the private key, if needed.

Returns: INT

1 - Success
0 - Not supported
-1 - Invalid request
-5 - Request is read only.
-6 - Failed to find certificate or key.

Declaration:

```
FUNCTION restReqClientCertSet: INT;
VAR_INPUT
   req            : SYSHANDLE;
   cert           : STRING;
   key            : STRING;
   pass           : STRING;
END_VAR;
```

Example:

Please see the "Examples - REST Example"

4.2.36.4.2 restReqClientCertPresent (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

This function will tell if the request contains a client certificate.
This function is meant to be called when an incoming request causes the callback registered with restServerEndpointAdd to be called.
If a certificate is present, it is already known that it is signed by the CA certificate provided as accept_ca in the call to restServerCreate.

Input:

req : SYSHANDLE
A handle to the request.

Returns: INT

1 - Request contains a client certificate
0 - Not supported
-1 - Invalid request
-5 - Not an incoming request
-6 - Request does not contain a client certificate.

Declaration:

```
FUNCTION restReqClientCertPresent : INT;
VAR_INPUT
   req            : SYSHANDLE;
END_VAR;
```

Example:

```
FUNCTION dumpCert;
VAR_INPUT
   req : SYSHANDLE;
   rip : DINT;
END_VAR;
VAR
   rc  : INT;
   str : STRING;
   d   : DINT;
   i   : INT;
   ip  : DINT;
END_VAR;
   rc := restReqClientCertPresent(req:=req);
   IF rc = 1 THEN
      DebugFmt(message:="Client cert present");
      rc := restReqClientCertSubjectGet(req := req, str := str);
      DebugFmt(message:=" Subject: "+str+": \1", v1 := rc);
      rc := restReqClientCertSubjectCNGet(req := req, str := str);
      DebugFmt(message:=" CN: "+str+": \1", v1 := rc);
      rc := restReqClientCertIssuerGet(req := req, str := str);
      DebugFmt(message:=" Issuer: "+str+": \1", v1 := rc);
      rc := restReqClientCertVersionGet(req := req);
      DebugFmt(message:=" Version: \1", v1 := rc);
      d := restReqClientCertValidFrom(req := req);
      DebugFmt(message:=" Valid from: \4, "+linsecToStr(linsec := d), v4 :=
d);
      d := restReqClientCertValidTo(req := req);
      DebugFmt(message:=" Valid to: \4, "+linsecToStr(linsec := d), v4 := d);
      rc := restReqClientCertSerialGet(req := req, str := str);
      DebugFmt(message:=" Serial: "+str+": \1", v1 := rc);
      rc := restReqClientCertFingerprintGet(req := req, type := 0, str :=
str);
      DebugFmt(message:=" SHA1  : "+str+": \1", v1 := rc);
      rc := restReqClientCertFingerprintGet(req := req, type := 1, str :=
str);
      DebugFmt(message:=" MD51  : "+str+": \1", v1 := rc);

      rc := restReqClientCertCheckHostname(req := req, hostname :=
"localhost");
      DebugFmt(message:=" Match localhost(\2): \1", v1 := rc, v2 := i);

      rc := restReqClientCertCheckEmail(req := req, email :=
"test@example.com");
      DebugFmt(message:=" Match test@example.com: \1", v1 := rc);


      i := 0;
      REPEAT
         rc := restReqClientCertSANGet(req := req, idx := i, san := str);
         DebugFmt(message:= "  SAN[\1]: \2: "+str, v1 := i, v2 := rc);
         IF rc = 4 THEN
            ip := soAddrToIP(address := str);
            IF ip = rip THEN
```

```
                DebugMsg(message:="   Matching IP found: "+str);
            END_IF;
         END_IF;
         i := i + 1;
      UNTIL rc = -21
      END_REPEAT;

   ELSE
      DebugFmt(message:="No client cert present: \1", v1 := rc);
   END_IF;
END_FUNCTION;
```

4.2.36.4.3 restReqClientCertSubjectGet (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

This function will get the subject of the client certificate as a string using the format "C=xxxx,O=yyyy,CN=zzzz" as described in RFC2253.
To only get the Common Name, use restReqClientCertSubjectCNGet.

Input:

req : SYSHANDLE
A handle to the request.

Output:

str : STRING
The subject of the certificate.

Returns: INT

1	- Success
0	- Not supported
-1	- Invalid request
-5	- Not an incoming request
-6	- Request does not contain a client certificate.
-21	- Subject was not found.

Declaration:

```
FUNCTION restReqClientCertSubjectGet : INT;
VAR_INPUT
   req            : SYSHANDLE;
   str            : ACCESS STRING;
END_VAR;
```

Example:

```
FUNCTION dumpCert;
VAR_INPUT
   req : SYSHANDLE;
   rip : DINT;
END_VAR;
VAR
   rc  : INT;
   str : STRING;
   d   : DINT;
   i   : INT;
   ip  : DINT;
```

```
END_VAR;
   rc := restReqClientCertPresent(req:=req);
   IF rc = 1 THEN
      DebugFmt(message:="Client cert present");
      rc := restReqClientCertSubjectGet(req := req, str := str);
      DebugFmt(message:=" Subject: "+str+": \1", v1 := rc);
      rc := restReqClientCertSubjectCNGet(req := req, str := str);
      DebugFmt(message:=" CN: "+str+": \1", v1 := rc);
      rc := restReqClientCertIssuerGet(req := req, str := str);
      DebugFmt(message:=" Issuer: "+str+": \1", v1 := rc);
      rc := restReqClientCertVersionGet(req := req);
      DebugFmt(message:=" Version: \1", v1 := rc);
      d := restReqClientCertValidFrom(req := req);
      DebugFmt(message:=" Valid from: \4, "+linsecToStr(linsec := d), v4 :=
d);
      d := restReqClientCertValidTo(req := req);
      DebugFmt(message:=" Valid to: \4, "+linsecToStr(linsec := d), v4 := d);
      rc := restReqClientCertSerialGet(req := req, str := str);
      DebugFmt(message:=" Serial: "+str+": \1", v1 := rc);
      rc := restReqClientCertFingerprintGet(req := req, type := 0, str :=
str);
      DebugFmt(message:=" SHA1  : "+str+": \1", v1 := rc);
      rc := restReqClientCertFingerprintGet(req := req, type := 1, str :=
str);
      DebugFmt(message:=" MD51  : "+str+": \1", v1 := rc);

      rc := restReqClientCertCheckHostname(req := req, hostname :=
"localhost");
      DebugFmt(message:=" Match localhost(\2): \1", v1 := rc, v2 := i);

      rc := restReqClientCertCheckEmail(req := req, email :=
"test@example.com");
      DebugFmt(message:=" Match test@example.com: \1", v1 := rc);


      i := 0;
      REPEAT
         rc := restReqClientCertSANGet(req := req, idx := i, san := str);
         DebugFmt(message:= "  SAN[\1]: \2: "+str, v1 := i, v2 := rc);
         IF rc = 4 THEN
            ip := soAddrToIP(address := str);
            IF ip = rip THEN
               DebugMsg(message:="   Matching IP found: "+str);
            END_IF;
         END_IF;
         i := i + 1;
      UNTIL rc = -21
      END_REPEAT;

   ELSE
      DebugFmt(message:="No client cert present: \1", v1 := rc);
   END_IF;
END_FUNCTION;
```

4.2.36.4.4 restReqClientCertSubjectCNGet (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

This function will get the Common Name (CN) from the Subject of the client certificate.
To get the entire subject, use restReqClientCertSubjectGet.

Input:
req : SYSHANDLE
A handle to the request.

Output:
str : STRING
The Common Name.

Returns: INT
1 - Success
0 - Not supported
-1 - Invalid request
-5 - Not an incoming request
-6 - Request does not contain a client certificate.
-21 - Common Name was not found.

Declaration:
```
FUNCTION restReqClientCertSubjectCNGet : INT;
VAR_INPUT
   req            : SYSHANDLE;
   str            : ACCESS STRING;
END_VAR;
```

Example:

```
FUNCTION dumpCert;
VAR_INPUT
   req : SYSHANDLE;
   rip : DINT;
END_VAR;
VAR
   rc  : INT;
   str : STRING;
   d   : DINT;
   i   : INT;
   ip  : DINT;
END_VAR;
   rc := restReqClientCertPresent(req:=req);
   IF rc = 1 THEN
      DebugFmt(message:="Client cert present");
      rc := restReqClientCertSubjectGet(req := req, str := str);
      DebugFmt(message:=" Subject: "+str+": \1", v1 := rc);
      rc := restReqClientCertSubjectCNGet(req := req, str := str);
      DebugFmt(message:=" CN: "+str+": \1", v1 := rc);
      rc := restReqClientCertIssuerGet(req := req, str := str);
      DebugFmt(message:=" Issuer: "+str+": \1", v1 := rc);
      rc := restReqClientCertVersionGet(req := req);
      DebugFmt(message:=" Version: \1", v1 := rc);
      d := restReqClientCertValidFrom(req := req);
      DebugFmt(message:=" Valid from: \4, "+linsecToStr(linsec := d), v4 :=
d);
      d := restReqClientCertValidTo(req := req);
      DebugFmt(message:=" Valid to: \4, "+linsecToStr(linsec := d), v4 := d);
      rc := restReqClientCertSerialGet(req := req, str := str);
      DebugFmt(message:=" Serial: "+str+": \1", v1 := rc);
      rc := restReqClientCertFingerprintGet(req := req, type := 0, str :=
str);
      DebugFmt(message:=" SHA1  : "+str+": \1", v1 := rc);
```

```
      rc := restReqClientCertFingerprintGet(req := req, type := 1, str :=
str);
      DebugFmt(message:=" MD51  : "+str+": \1", v1 := rc);

      rc := restReqClientCertCheckHostname(req := req, hostname :=
"localhost");
      DebugFmt(message:=" Match localhost(\2): \1", v1 := rc, v2 := i);

      rc := restReqClientCertCheckEmail(req := req, email :=
"test@example.com");
      DebugFmt(message:=" Match test@example.com: \1", v1 := rc);


      i := 0;
      REPEAT
         rc := restReqClientCertSANGet(req := req, idx := i, san := str);
         DebugFmt(message:= "  SAN[\1]: \2: "+str, v1 := i, v2 := rc);
         IF rc = 4 THEN
            ip := soAddrToIP(address := str);
            IF ip = rip THEN
               DebugMsg(message:="   Matching IP found: "+str);
            END_IF;
         END_IF;
         i := i + 1;
      UNTIL rc = -21
      END_REPEAT;

   ELSE
      DebugFmt(message:="No client cert present: \1", v1 := rc);
   END_IF;
END_FUNCTION;
```

4.2.36.4.5 restReqClientCertIssuerGet (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

This function will get the issuer of the client certificate as a string using the format "C=xxxx,O=yyyy,CN=zzzz" as described in RFC2253.

Input:

req : SYSHANDLE
A handle to the request.

Output:

str : STRING
The issuer of the certificate.

Returns: INT

1	- Success
0	- Not supported
-1	- Invalid request
-5	- Not an incoming request
-6	- Request does not contain a client certificate.
-21	- Issuer was not found.

Declaration:

```
FUNCTION restReqClientCertIssuerGet : INT;
VAR_INPUT
   req            : SYSHANDLE;
   str            : ACCESS STRING;
END_VAR;
```

Example:

```
FUNCTION dumpCert;
VAR_INPUT
   req : SYSHANDLE;
   rip : DINT;
END_VAR;
VAR
   rc  : INT;
   str : STRING;
   d   : DINT;
   i   : INT;
   ip  : DINT;
END_VAR;
   rc := restReqClientCertPresent(req:=req);
   IF rc = 1 THEN
      DebugFmt(message:="Client cert present");
      rc := restReqClientCertSubjectGet(req := req, str := str);
      DebugFmt(message:=" Subject: "+str+": \1", v1 := rc);
      rc := restReqClientCertSubjectCNGet(req := req, str := str);
      DebugFmt(message:=" CN: "+str+": \1", v1 := rc);
      rc := restReqClientCertIssuerGet(req := req, str := str);
      DebugFmt(message:=" Issuer: "+str+": \1", v1 := rc);
      rc := restReqClientCertVersionGet(req := req);
      DebugFmt(message:=" Version: \1", v1 := rc);
      d := restReqClientCertValidFrom(req := req);
      DebugFmt(message:=" Valid from: \4, "+linsecToStr(linsec := d), v4 :=
d);
      d := restReqClientCertValidTo(req := req);
      DebugFmt(message:=" Valid to: \4, "+linsecToStr(linsec := d), v4 := d);
      rc := restReqClientCertSerialGet(req := req, str := str);
      DebugFmt(message:=" Serial: "+str+": \1", v1 := rc);
      rc := restReqClientCertFingerprintGet(req := req, type := 0, str :=
str);
      DebugFmt(message:=" SHA1  : "+str+": \1", v1 := rc);
      rc := restReqClientCertFingerprintGet(req := req, type := 1, str :=
str);
      DebugFmt(message:=" MD51  : "+str+": \1", v1 := rc);

      rc := restReqClientCertCheckHostname(req := req, hostname :=
"localhost");
      DebugFmt(message:=" Match localhost(\2): \1", v1 := rc, v2 := i);

      rc := restReqClientCertCheckEmail(req := req, email :=
"test@example.com");
      DebugFmt(message:=" Match test@example.com: \1", v1 := rc);


      i := 0;
      REPEAT
         rc := restReqClientCertSANGet(req := req, idx := i, san := str);
         DebugFmt(message:= "  SAN[\1]: \2: "+str, v1 := i, v2 := rc);
         IF rc = 4 THEN
            ip := soAddrToIP(address := str);
            IF ip = rip THEN
               DebugMsg(message:="   Matching IP found: "+str);
            END_IF;
```

```
         END_IF;
         i := i + 1;
      UNTIL rc = -21
      END_REPEAT;

   ELSE
      DebugFmt(message:="No client cert present: \1", v1 := rc);
   END_IF;
END_FUNCTION;
```

4.2.36.4.6 restReqClientCertVersionGet (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

This function returns the version number of the format of the client certificate.
The version number specifies which features are supported, e.g. certificate extensions requires version 3.

Input:

req : SYSHANDLE
A handle to the request.

Returns: INT

>0	- Certificate version number
0	- Not supported
-1	- Invalid request
-5	- Not an incoming request
-6	- Request does not contain a client certificate.

Declaration:

```
FUNCTION restReqClientCertVersionGet : INT;
VAR_INPUT
   req            : SYSHANDLE;
END_VAR;
```

Example:

```
FUNCTION dumpCert;
VAR_INPUT
   req : SYSHANDLE;
   rip : DINT;
END_VAR;
VAR
   rc  : INT;
   str : STRING;
   d   : DINT;
   i   : INT;
   ip  : DINT;
END_VAR;
   rc := restReqClientCertPresent(req:=req);
   IF rc = 1 THEN
      DebugFmt(message:="Client cert present");
      rc := restReqClientCertSubjectGet(req := req, str := str);
      DebugFmt(message:=" Subject: "+str+": \1", v1 := rc);
```

```
      rc := restReqClientCertSubjectCNGet(req := req, str := str);
      DebugFmt(message:=" CN: "+str+": \1", v1 := rc);
      rc := restReqClientCertIssuerGet(req := req, str := str);
      DebugFmt(message:=" Issuer: "+str+": \1", v1 := rc);
      rc := restReqClientCertVersionGet(req := req);
      DebugFmt(message:=" Version: \1", v1 := rc);
      d := restReqClientCertValidFrom(req := req);
      DebugFmt(message:=" Valid from: \4, "+linsecToStr(linsec := d), v4 :=
d);
      d := restReqClientCertValidTo(req := req);
      DebugFmt(message:=" Valid to: \4, "+linsecToStr(linsec := d), v4 := d);
      rc := restReqClientCertSerialGet(req := req, str := str);
      DebugFmt(message:=" Serial: "+str+": \1", v1 := rc);
      rc := restReqClientCertFingerprintGet(req := req, type := 0, str :=
str);
      DebugFmt(message:=" SHA1  : "+str+": \1", v1 := rc);
      rc := restReqClientCertFingerprintGet(req := req, type := 1, str :=
str);
      DebugFmt(message:=" MD51  : "+str+": \1", v1 := rc);

      rc := restReqClientCertCheckHostname(req := req, hostname :=
"localhost");
      DebugFmt(message:=" Match localhost(\2): \1", v1 := rc, v2 := i);

      rc := restReqClientCertCheckEmail(req := req, email :=
"test@example.com");
      DebugFmt(message:=" Match test@example.com: \1", v1 := rc);


      i := 0;
      REPEAT
         rc := restReqClientCertSANGet(req := req, idx := i, san := str);
         DebugFmt(message:= "  SAN[\1]: \2: "+str, v1 := i, v2 := rc);
         IF rc = 4 THEN
            ip := soAddrToIP(address := str);
            IF ip = rip THEN
               DebugMsg(message:="   Matching IP found: "+str);
            END_IF;
         END_IF;
         i := i + 1;
      UNTIL rc = -21
      END_REPEAT;

   ELSE
      DebugFmt(message:="No client cert present: \1", v1 := rc);
   END_IF;
END_FUNCTION;
```

4.2.36.4.7 restReqClientCertValidFrom (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

This function will return the start of the valid date range of the client certificate.

Input:

req : SYSHANDLE
A handle to the request.

Returns: DINT

- The start of the valid rage as a linsec
0 - Not supported
-1 - Invalid request
-5 - Not an incoming request
-6 - Request does not contain a client certificate.

Declaration:

```
FUNCTION restReqClientCertValidFrom : DINT;
VAR_INPUT
   req            : SYSHANDLE;
END_VAR;
```

Example:

```
FUNCTION dumpCert;
VAR_INPUT
   req : SYSHANDLE;
   rip : DINT;
END_VAR;
VAR
   rc  : INT;
   str : STRING;
   d   : DINT;
   i   : INT;
   ip  : DINT;
END_VAR;
   rc := restReqClientCertPresent(req:=req);
   IF rc = 1 THEN
      DebugFmt(message:="Client cert present");
      rc := restReqClientCertSubjectGet(req := req, str := str);
      DebugFmt(message:=" Subject: "+str+": \1", v1 := rc);
      rc := restReqClientCertSubjectCNGet(req := req, str := str);
      DebugFmt(message:=" CN: "+str+": \1", v1 := rc);
      rc := restReqClientCertIssuerGet(req := req, str := str);
      DebugFmt(message:=" Issuer: "+str+": \1", v1 := rc);
      rc := restReqClientCertVersionGet(req := req);
      DebugFmt(message:=" Version: \1", v1 := rc);
      d := restReqClientCertValidFrom(req := req);
      DebugFmt(message:=" Valid from: \4, "+linsecToStr(linsec := d), v4 :=
d);
      d := restReqClientCertValidTo(req := req);
      DebugFmt(message:=" Valid to: \4, "+linsecToStr(linsec := d), v4 := d);
      rc := restReqClientCertSerialGet(req := req, str := str);
      DebugFmt(message:=" Serial: "+str+": \1", v1 := rc);
      rc := restReqClientCertFingerprintGet(req := req, type := 0, str :=
str);
      DebugFmt(message:=" SHA1  : "+str+": \1", v1 := rc);
      rc := restReqClientCertFingerprintGet(req := req, type := 1, str :=
str);
      DebugFmt(message:=" MD51  : "+str+": \1", v1 := rc);

      rc := restReqClientCertCheckHostname(req := req, hostname :=
"localhost");
      DebugFmt(message:=" Match localhost(\2): \1", v1 := rc, v2 := i);

      rc := restReqClientCertCheckEmail(req := req, email :=
"test@example.com");
```

```
      DebugFmt(message:=" Match test@example.com: \1", v1 := rc);


      i := 0;
      REPEAT
         rc := restReqClientCertSANGet(req := req, idx := i, san := str);
         DebugFmt(message:= "  SAN[\1]: \2: "+str, v1 := i, v2 := rc);
         IF rc = 4 THEN
            ip := soAddrToIP(address := str);
            IF ip = rip THEN
               DebugMsg(message:="   Matching IP found: "+str);
            END_IF;
         END_IF;
         i := i + 1;
      UNTIL rc = -21
      END_REPEAT;

   ELSE
      DebugFmt(message:="No client cert present: \1", v1 := rc);
   END_IF;
END_FUNCTION;
```

4.2.36.4.8 restReqClientCertValidTo (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

This function will return the end of the valid date range of the client certificate.
Note: If the end date is in 2038 or later, it will be returned as 2145914603 (2037-12-31 23:23:23).

Input:

req : SYSHANDLE
A handle to the request.

Returns: DINT

- The end of the valid rage as a linsec
0 - Not supported
-1 - Invalid request
-5 - Not an incoming request
-6 - Request does not contain a client certificate.

Declaration:

```
FUNCTION restReqClientCertValidTo : DINT;
VAR_INPUT
   req            : SYSHANDLE;
END_VAR;
```

Example:

```
FUNCTION dumpCert;
VAR_INPUT
   req : SYSHANDLE;
   rip : DINT;
END_VAR;
VAR
```

```
   rc  : INT;
   str : STRING;
   d   : DINT;
   i   : INT;
   ip  : DINT;
END_VAR;
   rc := restReqClientCertPresent(req:=req);
   IF rc = 1 THEN
      DebugFmt(message:="Client cert present");
      rc := restReqClientCertSubjectGet(req := req, str := str);
      DebugFmt(message:=" Subject: "+str+": \1", v1 := rc);
      rc := restReqClientCertSubjectCNGet(req := req, str := str);
      DebugFmt(message:=" CN: "+str+": \1", v1 := rc);
      rc := restReqClientCertIssuerGet(req := req, str := str);
      DebugFmt(message:=" Issuer: "+str+": \1", v1 := rc);
      rc := restReqClientCertVersionGet(req := req);
      DebugFmt(message:=" Version: \1", v1 := rc);
      d := restReqClientCertValidFrom(req := req);
      DebugFmt(message:=" Valid from: \4, "+linsecToStr(linsec := d), v4 :=
d);
      d := restReqClientCertValidTo(req := req);
      DebugFmt(message:=" Valid to: \4, "+linsecToStr(linsec := d), v4 := d);
      rc := restReqClientCertSerialGet(req := req, str := str);
      DebugFmt(message:=" Serial: "+str+": \1", v1 := rc);
      rc := restReqClientCertFingerprintGet(req := req, type := 0, str :=
str);
      DebugFmt(message:=" SHA1  : "+str+": \1", v1 := rc);
      rc := restReqClientCertFingerprintGet(req := req, type := 1, str :=
str);
      DebugFmt(message:=" MD51  : "+str+": \1", v1 := rc);

      rc := restReqClientCertCheckHostname(req := req, hostname :=
"localhost");
      DebugFmt(message:=" Match localhost(\2): \1", v1 := rc, v2 := i);

      rc := restReqClientCertCheckEmail(req := req, email :=
"test@example.com");
      DebugFmt(message:=" Match test@example.com: \1", v1 := rc);


      i := 0;
      REPEAT
         rc := restReqClientCertSANGet(req := req, idx := i, san := str);
         DebugFmt(message:= "  SAN[\1]: \2: "+str, v1 := i, v2 := rc);
         IF rc = 4 THEN
            ip := soAddrToIP(address := str);
            IF ip = rip THEN
               DebugMsg(message:="   Matching IP found: "+str);
            END_IF;
         END_IF;
         i := i + 1;
      UNTIL rc = -21
      END_REPEAT;

   ELSE
      DebugFmt(message:="No client cert present: \1", v1 := rc);
   END_IF;
END_FUNCTION;
```

4.2.36.4.9 restReqClientCertSerialGet (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

This function will get the serial number of the client certificate.

Input:

req : SYSHANDLE
A handle to the request.

Output:

str : STRING
The serial number of the certificate.

Returns: INT

1 - Success
0 - Not supported
-1 - Invalid request
-5 - Not an incoming request
-6 - Request does not contain a client certificate.
-21 - Serial number was not found.

Declaration:

```
FUNCTION restReqClientCertSerialGet : INT;
VAR_INPUT
   req            : SYSHANDLE;
   str            : ACCESS STRING;
END_VAR;
```

Example:

```
FUNCTION dumpCert;
VAR_INPUT
   req : SYSHANDLE;
   rip : DINT;
END_VAR;
VAR
   rc  : INT;
   str : STRING;
   d   : DINT;
   i   : INT;
   ip  : DINT;
END_VAR;
   rc := restReqClientCertPresent(req:=req);
   IF rc = 1 THEN
      DebugFmt(message:="Client cert present");
      rc := restReqClientCertSubjectGet(req := req, str := str);
      DebugFmt(message:=" Subject: "+str+": \1", v1 := rc);
      rc := restReqClientCertSubjectCNGet(req := req, str := str);
      DebugFmt(message:=" CN: "+str+": \1", v1 := rc);
      rc := restReqClientCertIssuerGet(req := req, str := str);
      DebugFmt(message:=" Issuer: "+str+": \1", v1 := rc);
      rc := restReqClientCertVersionGet(req := req);
      DebugFmt(message:=" Version: \1", v1 := rc);
      d := restReqClientCertValidFrom(req := req);
      DebugFmt(message:=" Valid from: \4, "+linsecToStr(linsec := d), v4 :=
```

```
d);
      d := restReqClientCertValidTo(req := req);
      DebugFmt(message:=" Valid to: \4, "+linsecToStr(linsec := d), v4 := d);
      rc := restReqClientCertSerialGet(req := req, str := str);
      DebugFmt(message:=" Serial: "+str+": \1", v1 := rc);
      rc := restReqClientCertFingerprintGet(req := req, type := 0, str :=
str);
      DebugFmt(message:=" SHA1  : "+str+": \1", v1 := rc);
      rc := restReqClientCertFingerprintGet(req := req, type := 1, str :=
str);
      DebugFmt(message:=" MD51  : "+str+": \1", v1 := rc);

      rc := restReqClientCertCheckHostname(req := req, hostname :=
"localhost");
      DebugFmt(message:=" Match localhost(\2): \1", v1 := rc, v2 := i);

      rc := restReqClientCertCheckEmail(req := req, email :=
"test@example.com");
      DebugFmt(message:=" Match test@example.com: \1", v1 := rc);


      i := 0;
      REPEAT
         rc := restReqClientCertSANGet(req := req, idx := i, san := str);
         DebugFmt(message:= "  SAN[\1]: \2: "+str, v1 := i, v2 := rc);
         IF rc = 4 THEN
            ip := soAddrToIP(address := str);
            IF ip = rip THEN
               DebugMsg(message:="   Matching IP found: "+str);
            END_IF;
         END_IF;
         i := i + 1;
      UNTIL rc = -21
      END_REPEAT;

   ELSE
      DebugFmt(message:="No client cert present: \1", v1 := rc);
   END_IF;
END_FUNCTION;
```

4.2.36.4.10 restReqClientCertFingerprintGet (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

This function will calculate the fingerprint of the client certificate using the specified algorithm.

Input:

req : SYSHANDLE
A handle to the request.

type : INT *(Default 0)*
The algorithm to use:
0 - SHA-1
1 - MD5

Output:

str : STRING
The fingerprint of the certificate.

Returns: INT

1 - Success
0 - Not supported
-1 - Invalid request
-3 - Failed to calculate fingerprint
-4 - Invalid algorithm.
-5 - Not an incoming request
-6 - Request does not contain a client certificate.

Declaration:

```
FUNCTION restReqClientCertFingerprintGet : INT;
VAR_INPUT
   req            : SYSHANDLE;
   type           : INT;
   str            : ACCESS STRING;
END_VAR;
```

Example:

```
FUNCTION dumpCert;
VAR_INPUT
   req : SYSHANDLE;
   rip : DINT;
END_VAR;
VAR
   rc  : INT;
   str : STRING;
   d   : DINT;
   i   : INT;
   ip  : DINT;
END_VAR;
   rc := restReqClientCertPresent(req:=req);
   IF rc = 1 THEN
      DebugFmt(message:="Client cert present");
      rc := restReqClientCertSubjectGet(req := req, str := str);
      DebugFmt(message:=" Subject: "+str+": \1", v1 := rc);
      rc := restReqClientCertSubjectCNGet(req := req, str := str);
      DebugFmt(message:=" CN: "+str+": \1", v1 := rc);
      rc := restReqClientCertIssuerGet(req := req, str := str);
      DebugFmt(message:=" Issuer: "+str+": \1", v1 := rc);
      rc := restReqClientCertVersionGet(req := req);
      DebugFmt(message:=" Version: \1", v1 := rc);
      d := restReqClientCertValidFrom(req := req);
      DebugFmt(message:=" Valid from: \4, "+linsecToStr(linsec := d), v4 :=
d);
      d := restReqClientCertValidTo(req := req);
      DebugFmt(message:=" Valid to: \4, "+linsecToStr(linsec := d), v4 := d);
      rc := restReqClientCertSerialGet(req := req, str := str);
      DebugFmt(message:=" Serial: "+str+": \1", v1 := rc);
      rc := restReqClientCertFingerprintGet(req := req, type := 0, str :=
str);
      DebugFmt(message:=" SHA1  : "+str+": \1", v1 := rc);
      rc := restReqClientCertFingerprintGet(req := req, type := 1, str :=
str);
      DebugFmt(message:=" MD51  : "+str+": \1", v1 := rc);

      rc := restReqClientCertCheckHostname(req := req, hostname :=
"localhost");
      DebugFmt(message:=" Match localhost(\2): \1", v1 := rc, v2 := i);
```

```
      rc := restReqClientCertCheckEmail(req := req, email :=
"test@example.com");
      DebugFmt(message:=" Match test@example.com: \1", v1 := rc);


      i := 0;
      REPEAT
         rc := restReqClientCertSANGet(req := req, idx := i, san := str);
         DebugFmt(message:= "  SAN[\1]: \2: "+str, v1 := i, v2 := rc);
         IF rc = 4 THEN
            ip := soAddrToIP(address := str);
            IF ip = rip THEN
               DebugMsg(message:="   Matching IP found: "+str);
            END_IF;
         END_IF;
         i := i + 1;
      UNTIL rc = -21
      END_REPEAT;

   ELSE
      DebugFmt(message:="No client cert present: \1", v1 := rc);
   END_IF;
END_FUNCTION;
```

4.2.36.4.11 restReqClientCertSANGet (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

This function will read a Subject Alternative Name (SAN) from the client certificate.
The SAN is a DNS hostname, email address, IP address or other kind of name that the certificate is valid for.
The type of the SAN will be returned from the function. The value of the SAN will be read for some supported SAN types.

As a certificate can contain multiple SAN, it can be necessary to iterate over them to find the needed SAN.

Input:

req : SYSHANDLE
A handle to the request.

idx : INT *(Default 0)*
The index of the SAN to read, starting at 0.

Output:

san : STRING
The SAN as a string, if it is possible for the given type, see below.

Returns: INT

8 - XMPP. `san` is empty.
7 - Distinguished name. `san` contains the name.
6 - Othername. `san` is empty.
5 - IP v6 address. `san` is empty.

4	- IP v4 address. `san` contains the address in the socket address format, soAddrToIP can be used to convert it to a DINT.
3	- URI. `san` contains the URI.
2	- RFC822 name (e-mail). `san` contains the name.
1	- DNS name. `san` contains the name.
0	- Not supported
-1	- Invalid request
-5	- Not an incoming request
-6	- Request does not contain a client certificate.
-21	- SAN not found at the given index, the last SAN might have been read.

Declaration:

```
FUNCTION restReqClientCertSANGet : INT;
VAR_INPUT
   req            : SYSHANDLE;
   idx            : INT;
   san            : ACCESS STRING;
END_VAR;
```

Example:

```
FUNCTION dumpCert;
VAR_INPUT
   req : SYSHANDLE;
   rip : DINT;
END_VAR;
VAR
   rc  : INT;
   str : STRING;
   d   : DINT;
   i   : INT;
   ip  : DINT;
END_VAR;
   rc := restReqClientCertPresent(req:=req);
   IF rc = 1 THEN
      DebugFmt(message:="Client cert present");
      rc := restReqClientCertSubjectGet(req := req, str := str);
      DebugFmt(message:=" Subject: "+str+": \1", v1 := rc);
      rc := restReqClientCertSubjectCNGet(req := req, str := str);
      DebugFmt(message:=" CN: "+str+": \1", v1 := rc);
      rc := restReqClientCertIssuerGet(req := req, str := str);
      DebugFmt(message:=" Issuer: "+str+": \1", v1 := rc);
      rc := restReqClientCertVersionGet(req := req);
      DebugFmt(message:=" Version: \1", v1 := rc);
      d := restReqClientCertValidFrom(req := req);
      DebugFmt(message:=" Valid from: \4, "+linsecToStr(linsec := d), v4 :=
d);
      d := restReqClientCertValidTo(req := req);
      DebugFmt(message:=" Valid to: \4, "+linsecToStr(linsec := d), v4 := d);
      rc := restReqClientCertSerialGet(req := req, str := str);
      DebugFmt(message:=" Serial: "+str+": \1", v1 := rc);
      rc := restReqClientCertFingerprintGet(req := req, type := 0, str :=
str);
      DebugFmt(message:=" SHA1  : "+str+": \1", v1 := rc);
      rc := restReqClientCertFingerprintGet(req := req, type := 1, str :=
str);
      DebugFmt(message:=" MD51  : "+str+": \1", v1 := rc);

      rc := restReqClientCertCheckHostname(req := req, hostname :=
"localhost");
```

```
      DebugFmt(message:=" Match localhost(\2): \1", v1 := rc, v2 := i);

      rc := restReqClientCertCheckEmail(req := req, email :=
"test@example.com");
      DebugFmt(message:=" Match test@example.com: \1", v1 := rc);


      i := 0;
      REPEAT
         rc := restReqClientCertSANGet(req := req, idx := i, san := str);
         DebugFmt(message:= "  SAN[\1]: \2: "+str, v1 := i, v2 := rc);
         IF rc = 4 THEN
            ip := soAddrToIP(address := str);
            IF ip = rip THEN
               DebugMsg(message:="   Matching IP found: "+str);
            END_IF;
         END_IF;
         i := i + 1;
      UNTIL rc = -21
      END_REPEAT;

   ELSE
      DebugFmt(message:="No client cert present: \1", v1 := rc);
   END_IF;
END_FUNCTION;
```

4.2.36.4.12 restReqClientCertCheckHostname (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

This function will check if the client certificate matches the provided hostname.

Input:

req : SYSHANDLE
A handle to the request.

hostname : STRING
The hostname to validate.

allow_wildcards : BOOL (*Default TRUE*)
If set to false, wild card certificates are not considered, so the hostname must match exactly.

Returns: INT

1	- Hostname matches
0	- Not supported
-1	- Invalid request
-5	- Not an incoming request
-6	- Request does not contain a client certificate.
-21	- No match found.

Declaration:

```
FUNCTION restReqClientCertCheckHostname: INT;
VAR_INPUT
   req              : SYSHANDLE;
   hostname         : STRING;
   allow_wildcards  : BOOL := TRUE;
```

```
END_VAR;
```

Example:

```
FUNCTION dumpCert;
VAR_INPUT
   req : SYSHANDLE;
   rip : DINT;
END_VAR;
VAR
   rc  : INT;
   str : STRING;
   d   : DINT;
   i   : INT;
   ip  : DINT;
END_VAR;
   rc := restReqClientCertPresent(req:=req);
   IF rc = 1 THEN
      DebugFmt(message:="Client cert present");
      rc := restReqClientCertSubjectGet(req := req, str := str);
      DebugFmt(message:=" Subject: "+str+": \1", v1 := rc);
      rc := restReqClientCertSubjectCNGet(req := req, str := str);
      DebugFmt(message:=" CN: "+str+": \1", v1 := rc);
      rc := restReqClientCertIssuerGet(req := req, str := str);
      DebugFmt(message:=" Issuer: "+str+": \1", v1 := rc);
      rc := restReqClientCertVersionGet(req := req);
      DebugFmt(message:=" Version: \1", v1 := rc);
      d := restReqClientCertValidFrom(req := req);
      DebugFmt(message:=" Valid from: \4, "+linsecToStr(linsec := d), v4 :=
d);
      d := restReqClientCertValidTo(req := req);
      DebugFmt(message:=" Valid to: \4, "+linsecToStr(linsec := d), v4 := d);
      rc := restReqClientCertSerialGet(req := req, str := str);
      DebugFmt(message:=" Serial: "+str+": \1", v1 := rc);
      rc := restReqClientCertFingerprintGet(req := req, type := 0, str :=
str);
      DebugFmt(message:=" SHA1  : "+str+": \1", v1 := rc);
      rc := restReqClientCertFingerprintGet(req := req, type := 1, str :=
str);
      DebugFmt(message:=" MD51  : "+str+": \1", v1 := rc);

      rc := restReqClientCertCheckHostname(req := req, hostname :=
"localhost");
      DebugFmt(message:=" Match localhost(\2): \1", v1 := rc, v2 := i);

      rc := restReqClientCertCheckEmail(req := req, email :=
"test@example.com");
      DebugFmt(message:=" Match test@example.com: \1", v1 := rc);


      i := 0;
      REPEAT
         rc := restReqClientCertSANGet(req := req, idx := i, san := str);
         DebugFmt(message:= "  SAN[\1]: \2: "+str, v1 := i, v2 := rc);
         IF rc = 4 THEN
            ip := soAddrToIP(address := str);
            IF ip = rip THEN
               DebugMsg(message:="   Matching IP found: "+str);
            END_IF;
         END_IF;
         i := i + 1;
      UNTIL rc = -21
      END_REPEAT;
```

```
   ELSE
      DebugFmt(message:="No client cert present: \1", v1 := rc);
   END_IF;
END_FUNCTION;
```

4.2.36.4.13 restReqClientCertCheckEmail (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

This function will check if the client certificate matches the provided e-mail address.

Input:

req : SYSHANDLE
A handle to the request.

email : STRING
The e-mail address to validate.

Returns: INT

1	- Hostname matches
0	- Not supported
-1	- Invalid request
-5	- Not an incoming request
-6	- Request does not contain a client certificate.
-21	- No match found.

Declaration:

```
FUNCTION restReqClientCertCheckEmail: INT;
VAR_INPUT
   req              : SYSHANDLE;
   hostname         : STRING;
END_VAR;
```

Example:

```
FUNCTION dumpCert;
VAR_INPUT
   req : SYSHANDLE;
   rip : DINT;
END_VAR;
VAR
   rc  : INT;
   str : STRING;
   d   : DINT;
   i   : INT;
   ip  : DINT;
END_VAR;
   rc := restReqClientCertPresent(req:=req);
   IF rc = 1 THEN
      DebugFmt(message:="Client cert present");
      rc := restReqClientCertSubjectGet(req := req, str := str);
      DebugFmt(message:=" Subject: "+str+": \1", v1 := rc);
      rc := restReqClientCertSubjectCNGet(req := req, str := str);
      DebugFmt(message:=" CN: "+str+": \1", v1 := rc);
```

```
      rc := restReqClientCertIssuerGet(req := req, str := str);
      DebugFmt(message:=" Issuer: "+str+": \1", v1 := rc);
      rc := restReqClientCertVersionGet(req := req);
      DebugFmt(message:=" Version: \1", v1 := rc);
      d := restReqClientCertValidFrom(req := req);
      DebugFmt(message:=" Valid from: \4, "+linsecToStr(linsec := d), v4 :=
d);
      d := restReqClientCertValidTo(req := req);
      DebugFmt(message:=" Valid to: \4, "+linsecToStr(linsec := d), v4 := d);
      rc := restReqClientCertSerialGet(req := req, str := str);
      DebugFmt(message:=" Serial: "+str+": \1", v1 := rc);
      rc := restReqClientCertFingerprintGet(req := req, type := 0, str :=
str);
      DebugFmt(message:=" SHA1  : "+str+": \1", v1 := rc);
      rc := restReqClientCertFingerprintGet(req := req, type := 1, str :=
str);
      DebugFmt(message:=" MD51  : "+str+": \1", v1 := rc);

      rc := restReqClientCertCheckHostname(req := req, hostname :=
"localhost");
      DebugFmt(message:=" Match localhost(\2): \1", v1 := rc, v2 := i);

      rc := restReqClientCertCheckEmail(req := req, email :=
"test@example.com");
      DebugFmt(message:=" Match test@example.com: \1", v1 := rc);


      i := 0;
      REPEAT
         rc := restReqClientCertSANGet(req := req, idx := i, san := str);
         DebugFmt(message:= "  SAN[\1]: \2: "+str, v1 := i, v2 := rc);
         IF rc = 4 THEN
            ip := soAddrToIP(address := str);
            IF ip = rip THEN
               DebugMsg(message:="   Matching IP found: "+str);
            END_IF;
         END_IF;
         i := i + 1;
      UNTIL rc = -21
      END_REPEAT;

   ELSE
      DebugFmt(message:="No client cert present: \1", v1 := rc);
   END_IF;
END_FUNCTION;
```

4.2.36.5 Response functions

4.2.36.5.1 restRespFree (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

This function will release a REST response that was created with restClientRequest.

Input:

resp : SYSHANDLE
A handle to the response.

Returns: INT

1 - Success
0 - Not supported
-1 - Invalid response

Declaration:

```
FUNCTION restRespFree : INT;
VAR_INPUT
   resp    : ACCESS SYSHANDLE;
END_VAR;
```

Example:

Please see the "Examples - REST Example"

4.2.36.5.2 restRespBodyGetString (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

This function will read the contents of the body of the response as a string.

Input:

resp : SYSHANDLE
A handle to the response.

Output:

str : STRING
The body contents.

Returns: INT

1 - Success
0 - Not supported
-1 - Invalid response
-7 - The body is too large to store in a string.

Declaration:

```
FUNCTION restRespBodyGetString : INT;
VAR_INPUT
   resp          : SYSHANDLE;
   str           : ACCESS STRING;
END_VAR;
```

Example:

Please see the "Examples - REST Example"

4.2.36.5.3 restRespBodySetString (Function)

```
Architecture:       NX32L
Device support:     ALL
Firmware version:   2.10.00
```

This function will set the body content of the response to the provided string.

When the body needs to contain mostly fixed data, one option is to use strTemplateCreate to add the dynamic data to a fixed template string, which can then be used as the body, without needing to manually create the entire string.

Input:

resp : SYSHANDLE
A handle to the response.

str : STRING
The string to use for the body content.

Returns: INT

1 - Success
0 - Not supported
-1 - Invalid response
-3 - Failed to set body

Declaration:

```
FUNCTION restRespBodySetString: INT;
VAR_INPUT
   resp           : SYSHANDLE;
   str            : STRING;
END_VAR;
```

Example:

Please see the "Examples - REST Example"

4.2.36.5.4 restRespBodyGetJSON (Function)

```
Architecture:       NX32L
Device support:     ALL
Firmware version:   2.10.00
```

This function will read the contents of the body of the response as a JSON structure.

Input:

resp : SYSHANDLE
A handle to the response.

Output:

json : SYSHANDLE
A handle to the JSON structure created from the body content.
When the JSON structure is no longer needed, it must be released using jsonFree.

Returns: INT

1 - Success
0 - Not supported
-1 - Invalid response
-2 - The max number of JSON structures have already been created.
-3 - Failed to create JSON structure. Body content might not be valid JSON.

Declaration:

```
FUNCTION restRespBodyGetJSON : INT;
VAR_INPUT
   resp            : SYSHANDLE;
   json            : ACCESS SYSHANDLE;
END_VAR;
```

Example:

Please see the "Examples - REST Example"

4.2.36.5.5 restRespBodySetJSON (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

This function will set the body content of the response to the provided JSON structure.

Input:

resp : SYSHANDLE
A handle to the response.

json : SYSHANDLE
The JSON structure to use for the body content.

Returns: INT

1 - Success
0 - Not supported
-1 - Invalid response
-3 - Failed to set body
-4 - Invalid JSON handle.

Declaration:

```
FUNCTION restRespBodySetJSON: INT;
VAR_INPUT
   resp            : SYSHANDLE;
   json            : SYSHANDLE;
END_VAR;
```

Example:

Please see the "Examples - REST Basic Authentication Example"

4.2.36.5.6 restRespBodySize (Function)

```
Architecture:       NX32L
Device support:     ALL
Firmware version:   2.10.00
```

This function will return the size of the body content in the response.

Input:

resp : SYSHANDLE
A handle to the response.

Returns: DINT

```
 >=0    - Size of response in bytes
   0    - Not supported
  -1    - Invalid response
  -2    - The response is too large to report.
```

Declaration:

```
FUNCTION restRespBodySize : DINT;
VAR_INPUT
   resp             : SYSHANDLE;
END_VAR;
```

Example:

Please see the "Examples - REST Basic Authentication Example"

4.2.36.5.7 restRespBodyGetRaw (Function)

```
Architecture:       NX32L
Device support:     ALL
Firmware version:   2.10.00
```

This function will copy a part the body contents from the response into the provided buffer.

Input:

resp : SYSHANDLE
A handle to the response.

dst : PTR
A pointer to the buffer to place the body content in.

start : DINT *(default 0)*
The offset to start copying from, starting at 0.

size : DINT
The maximum length to read, i.e. the size of the buffer.

Returns: INT

```
  >0    - Success, number of bytes copied.
   0    - Not supported
  -1    - Invalid response
  -2    - Start is past the end of the body.
```

-4 - Size, start or the destination is invalid.

Declaration:

```
FUNCTION restRespBodyGetRaw : DINT;
VAR_INPUT
   resp            : SYSHANDLE;
   dst             : PTR;
   start           : DINT;
   size            : DINT;
END_VAR;
```

Example:

Please see the "Examples - REST Basic Authentication Example"

4.2.36.5.8 restRespHeaderKey (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

This function will read the name of a header field in a response.
By calling this multiple times with incrementing index, it is possible to list all the header fields in the response.

Input:

resp : SYSHANDLE
A handle to the response.

idx : INT *(Default 0)*
The index of the header field to read the name of.

Output:

name : STRING
The name of the field.

Returns: INT

1 - Success
0 - Not supported
-1 - Invalid response
-2 - The header field was not found

Declaration:

```
FUNCTION restRespHeaderKey : INT;
VAR_INPUT
   resp            : SYSHANDLE;
   idx             : INT;
   name            : ACCESS STRING;
END_VAR;
```

Example:

Please see the "Examples - REST Basic Authentication Example"

4.2.36.5.9 restRespHeaderGet (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

This function will read a header field from a response.

Input:

resp : SYSHANDLE
A handle to the response.

name : STRING
The name of the header field to read.
Common fields include "Content-Type", "Accept" and "User-Agent".

check_case : BOOL *(Default FALSE)*
If set to TRUE, the name will use case sensitive comparison.
If left at FALSE, the name will use case insensitive comparison.
Note that if the same name is present with different case, only the value for the first matching name will be returned.

Output:

value : STRING
The value of the field.

Returns: INT

1	- Success
0	- Not supported
-1	- Invalid response
-2	- The header was not found
-7	- The value was too large for a string.

Declaration:

```
FUNCTION restRespHeaderGet : INT;
VAR_INPUT
   resp           : SYSHANDLE;
   name           : STRING;
   check_case     : BOOL := FALSE;
   value          : ACCESS STRING;
END_VAR;
```

Example:

Please see the "Examples - REST Basic Authentication Example"

4.2.36.5.10 restRespHeaderSet (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

This function will set the value of a header field in a response.

Input:

resp : SYSHANDLE
A handle to the response.

name : STRING
The name of the header field to set.
Common fields include "Content-Type", "Accept" and "User-Agent".

value : STRING
The new value of the field.

Returns: INT

1	- Success
0	- Not supported
-1	- Invalid response
-3	- Failed to set header
-5	- Response is read only.

Declaration:

```
FUNCTION restRespHeaderSet: INT;
VAR_INPUT
   req            : SYSHANDLE;
   name           : STRING;
   value          : STRING;
END_VAR;
```

Example:

Please see the "Examples - REST Example"

4.2.36.5.11 restRespBasicAuthSet (Function)

```
Architecture:       NX32L
Device support:     ALL
Firmware version:   2.10.00
```

This function will enable the use of basic authentication for a response, when returning status 401 (Unauthorized).
This is done by setting the WWW-Authenticate header in the response to use basic authentication.
This must be done to inform the client that it must provide username and password which can then be read with restReqBasicAuthGet.

Input:

resp : SYSHANDLE
A handle to the response.

realm : STRING
A string describing which login is needed, allowing for different usernames/passwords for different pages. If not specified, a default value will be used.

enable : BOOL *(Default TRUE)*
Set to false to disable basic authentication.

Returns: INT

1 - Success
0 - Not supported
-1 - Invalid request
-3 - Failed to set authentication
-5 - Response is read only.

Declaration:

```
FUNCTION restRespBasicAuthSet: INT;
VAR_INPUT
   resp           : SYSHANDLE;
   realm          : STRING;
   enable         : BOOL := TRUE;
END_VAR;
```

Example:

Please see the "Examples - REST Basic Authentication Example"

4.2.36.6 Client functions

4.2.36.6.1 restClientRequest (Function)

Architecture: NX32L
Device support: ALL
Firmware version: 2.10.00

This function will execute a client request and wait for the response.
If the path parameter is not set, the body of the response will be stored in memory and can be read using the restRespBodyGetString, restRespBodyGetJSON and restRespBodyGetRaw functions.
The maximum response size of a response in memory is 1 MB.
If the path parameter is set, the body content will instead be written to the file. The file will be created even if the request fails, e.g. because it can not find the server. If the request fails because the connection times out or if there is not enough free space for the entire file, the data that was received will still be available in the file.

Input:

req : SYSHANDLE
The request to execute.

path : STRING
The path to a file to store the response in.

iface : SINT *(Default 0)*
The network interface to use for the request. 0 = Any network, 1 = Mobile network, 2 = LAN network, etc. (See Network).

Output:

resp : SYSHANDLE
A handle to the response to the request. Must be freed with restRespFree when no longer needed.

Returns: INT

>0 - HTTP status code
0 - Not supported

-1	- Invalid request.
-2	- The response can not be created before a different response is freed.
-3	- Failed to execute request.
-7	- Response is too large to fit in memory or on the drive.
-10	- Invalid protocol requested.
-11	- Invalid url requested.
-12	- Failed to connect to server.
-13	- Invalid response received.
-14	- SSL Handshake failed.
-15	- Client certificate error.
-16	- Remote certificate could not be validated. Make sure that the certificate is valid and that the device time is correct.
-17	- Too many redirects.
-18	- Failed to escape parameter or key.
-19	- Failed to add headers.
-20	- Failed to add body content.
-21	- File not found
-22	- Access denied
-23	- Timeout
-24	- Could not resolve host.
-25	- Empty reply, server might have closed the connection.

Declaration:

```
FUNCTION restClientRequest : INT;
VAR_INPUT
   req       : SYSHANDLE;
   path      : STRING;
   iface     : SINT;
   resp      : ACCESS SYSHANDLE;
END_VAR;
```

Example:

Please see the "Examples - REST Example"

4.2.37 rf : RF Functions

4.2.37.1 rf : RF Functions

The RF functions offer the basic functionality for near to medium range wireless communication in the ISM frequency band.
The RF functionality is not available on all RTCU products. Therefore please refer to the technical documentation on the specific product for further information.

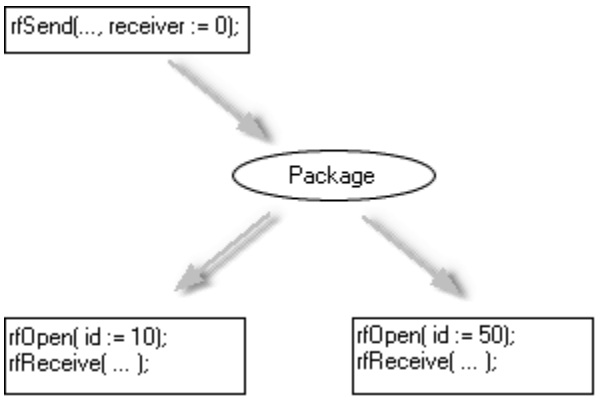
The RF functionality is based on state-of-the-art technology to minimize data loss and to ensure the best possible coverage. It includes advanced features such as: hardware packet filtering, "listen-before-send", transmission power adaptation and sophisticated AGC.

The hardware filtering ensures that packages are only received that are actually accepted by the application. This functionality is typically used for transparent point-to-point communication between two communication nodes with minimal handling.
When sending a package with the rfSend() function, it is ensured that the package is sent, but it is not guaranteed that the package has actually been received by the peer node. If, however, the package is received by the peer-node, it is guaranteed to be complete and unaltered in the transmission process.
It is therefore the responsibility of the application to implement a suitable handshake schema to ensure that a package has actually been received by the peer.

All packages are processed by the hardware filter, and only packages addressed to the ID as set by the rfOpen() function are accepted. "Broadcast" packages are accepted by the hardware filtering as well. Broadcast packages have their destination set to ID=0 when sent by rfSend() and will be received by any node that is listening for packages.

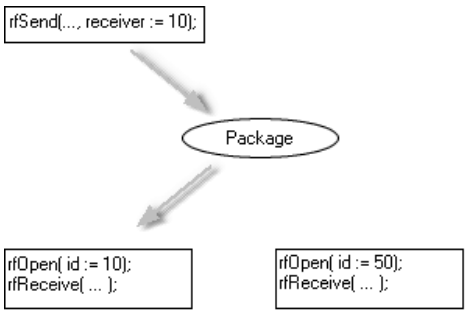
These two filtering scenarios are illustrated below:

Broadcasting:



When broadcasting, the receiver ID is "0". This ensures that any listing devices will receive the package - regardless of its own ID.

Addressing:



When defining a receiver ID, only those with same ID will receive the package

Transmission range:

Several parameters have influence on the transmission range. These include:

- Noise from surroundings.
- Transmitting power (see rfSetPower).
- Baud rate - lower baud rate will increase range (see rfOpen).

Architecture of the RF API

The RF API is based on a synchronous model where data is either sent (rfSend) or received (rfReceive). Each function will lock the API until completed.

The following RF functions are available:

- rfOpen — Opens the RF interface.
- rfClose — Closes the RF interface.
- rfReceive — Receives a package.
- rfSend — Sends a package.
- rfSetPower — Sets transmission power.

The implementation of RF has the following limitations:

- Maximum package size is 60 bytes.

4.2.37.2 rfOpen (Function)

```
Architecture:       X32 / NX32L
Device support:     AX9 pro, CX1 pro/pro-c/warp/warp-c, SX1, AX9
                    turbo, NX-900, LX5
Firmware version:   2.20 / 1.51.00
```

This function will open the RF interface. Calling this function is required before the RF interface can be used.
It is necessary to call rfClose after using the interface.

The ID must be specified when calling rfOpen and is used for the filtering of packages. Only packages sent with rfSend that have the specified ID as their destination or are sent out as a broadcast will be received (see RF Functions for details).

The communication speed set by the "baud" parameter must be equal for peers to communicate. A lower communication speed will yield a longer transmission range.

Calling rfOpen will revert the power level set by rfSetPower to the default value.

Input:

id : INT (1..254)
ID - used for filtering of incoming packages.

baud : DINT (1200,2400,4800,9600,38400) *(default* 4800*)*
Baud rate. A lower baud rate will increase the communication range.

Returns: INT

0 - Ok. Interface is open.
-2 - Already open.
-3 - Failed to power on RF module.
-4 - RF not supported on device or module is used by another interface (see rfbcClose).
-6 - Error in parameter.

Declaration:

```
FUNCTION rfOpen : INT;
VAR_INPUT
   id       : INT  := -1;
```

```
   baud    : DINT := 4800;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
BEGIN
  rfOpen(id := 1);
  ...
  rfClose();
END;
END_PROGRAM;
```

4.2.37.3 rfClose (Function)

Architecture: X32 / NX32L
Device support: AX9 pro, CX1 pro/pro-c/warp/warp-c, SX1, AX9 turbo, NX-900, LX5
Firmware version: 2.20 / 1.51.00

This function will close the RF interface. After the RF interface is closed, it cannot be used until opened again.
See rfOpen on how to open the RF interface.

When rfClose is called, any threads blocked in rfReceive will resume operation with error code -1.

Input:
None.

Returns: **INT**
0 - Success.
- 1 - RF not open. Use rfOpen to open interface.
- 4 - RF not supported on device.

Declaration:

```
FUNCTION rfClose : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
BEGIN
  rfOpen(id := 1);
  ...
  rfClose();
END;
END_PROGRAM;
```

4.2.37.4 rfSend (Function)

Architecture: X32 / NX32L
Device support: AX9 pro, CX1 pro/pro-c/warp/warp-c, SX1, AX9 turbo, NX-900, LX5
Firmware version: 2.20 / 1.51.00

rfSend will send a package (see RF Functions for details) to any peer set up with the correct parameters and actively listening for a package with rfReceive.

A package is defined by a block of data with a given length specified with the "data" and "size" parameters. A package can contain up to 60 bytes of data.

The "receiver" is the ID of the peer nodes to accept the package as specified by rfOpen on the peer node. When the "receiver" parameter is 0, a broadcast package is transmitted and will be received by any peer awaiting a package with rfReceive.
Also see the RF Functions for more information.

When simultaneously using rfSend and rfReceive, it may occur that rfSend will be temporarily blocked until rfReceive has finished the operation. The time to actually perform a transmission is therefore variable as a "listen-before-send" technique is deployed to minimize data transmission collision which maximizes the actual data throughput.

Input:

data : PTR
Address of the buffer that contains the data.

size : INT (0 .. 60) (*default 0*)
Number of bytes to send from the buffer.

receiver : INT (0 .. 254) (*default 0*)
Used by rfReceive on receiving device(s) to filter packages. When broadcasting to all IDs, use "*0*".

Returns: INT

0 - Success.
- 1 - RF not open. Use rfOpen to open interface.
- 4 - RF not supported on device.
- 6 - Invalid parameter.

Declaration:

```
FUNCTION rfSend : INT;
VAR_INPUT
   data     : PTR;       // Address of the buffer to send
   size     : INT;       // Number of bytes to send from the buffer
   receiver : INT := 0;  // Receiver id, 0 = broadcast to all
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   send  : BOOL R_EDGE;
   echo  : BOOL;
END_VAR;

VAR_OUTPUT
   sending   : BOOL;
   receiving : BOOL;
   no_data   : BOOL;
END_VAR;

PROGRAM test;
```

```
VAR
   rc       : INT;
   data     : ARRAY[ 1 .. 60 ] OF SINT;
   size     : INT;
   msg      : STRING;
   my_id    : INT := 1;
   sender   : INT;
   broadcast: BOOL;
END_VAR;
   msg := strFormat(format := "Message from \1", v1 := my_id);
   rfOpen(id := my_id);
BEGIN
   IF sending THEN
      size := strLen(str := msg);
      strToMemory(dst := ADDR(data), str := msg, len := size);
      rc := rfSend(data := ADDR(data), size := strLen(str := msg));
      DebugFmt(message := "rfSend(...) = \1", v1 := rc);
   ELSIF receiving THEN
      rc := rfReceive(data := ADDR(data), size := SIZEOF(data), sender :=
sender, broadcast := broadcast);
      IF rc < 0 THEN
         no_data := NOT no_data;
      ELSE
         no_data := FALSE;
         size := rc;
         rc := rfSend(receiver := sender, data := ADDR(data), size := size);
         DebugFmt(message := "echo : rfSend(...) = \1", v1 := rc);
      END_IF;
   END_IF;
   sending := send;
   receiving := echo;
END;
END_PROGRAM;
```

4.2.37.5 rfReceive (Function)

Architecture: X32 / NX32L
Device support: AX9 pro, CX1 pro/pro-c/warp/warp-c, SX1, AX9 turbo, NX-900, LX5
Firmware version: 2.20 / 1.51.00

---

rfReceive will listen for a package transmitted by rfSend from any peer node set up with the correct parameters.

A package will be received by rfReceive when the "receiver" parameter in rfSend of the peer node is equal to the ID as specified in the rfOpen function. Broadcast packages that have the ID=0 as their destination will be received.
Also refer to the RF Functions for additional details.

The package received will be placed in the buffer specified by the "data" parameter. The maximum size of the expected package to be received must be specified in the "size" buffer. Automatic truncation of the received package will occur in cases where the size of the incoming package is greater than the "size" specified. It is recommended to use a receive buffer of 60 bytes to avoid this situation.
Finally a "timeout" parameter is also specified. It allows the application to regain control within a time period in case no package has been received.

rfReceive will therefore block until a package in received, a timeout has occurred, or the interface is closed from another thread by calling rfClose.

When rfReceive returns, the ID of the peer node will be returned in "sender" and also information on whether the package was broadcast.

Input:

data : PTR
Address of the buffer to receive the package.

size : INT (0 .. 32767)
Maximum number of bytes to receive into buffer specified.
The buffer must be able to hold this number of bytes.
The number of bytes received will not exceed 60 but a larger buffer is allowed.

timeout : INT (-1 .. 32767) (*default 5000*)
milliseconds to waiting for incoming packages. "*-1*" means wait forever.

Output:

sender : INT
The ID used when rfOpen was called on the sender.

broadcast : BOOL
Whether the message was broadcast to all IDs or addressed directly to the ID used when rfOpen was called. See RF Functions for details.

Returns: INT

0 .. 60 - Number of bytes received.
- 1 - RF not open. Use rfOpen to open the interface.
- 3 - Error communicating with RF module.
- 4 - RF not supported on device.
- 5 - Timeout waiting for packet.
- 6 - Invalid parameter.

Declaration:

```
FUNCTION rfReceive : INT;
VAR_INPUT
   size     : INT;
   timeout  : INT := 5000; // milliseconds to wait for data
   data     : PTR;         // Address of the buffer to send
   sender   : ACCESS INT;  // rf id of sending device
   broadcast: ACCESS BOOL; // is data send to all id's.
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   send  : BOOL R_EDGE;
   echo  : BOOL;
END_VAR;

VAR_OUTPUT
   sending   : BOOL;
   receiving : BOOL;
   no_data   : BOOL;
END_VAR;

PROGRAM test;
VAR
   rc       : INT;
   data     : ARRAY[ 1 .. 60 ] OF SINT;
```

```
   size     : INT;
   msg      : STRING;
   my_id    : INT := 1;
   sender   : INT;
   broadcast: BOOL;
END_VAR;
   msg := strFormat(format := "Message from \1", v1 := my_id);
   rfOpen(id := my_id);
BEGIN
   IF sending THEN
      size := strLen(str := msg);
      strToMemory(dst := ADDR(data), str := msg, len := size);
      rc := rfSend(data := ADDR(data), size := strLen(str := msg));
      DebugFmt(message := "rfSend(...) = \1", v1 := rc);
   ELSIF receiving THEN
      rc := rfReceive(data := ADDR(data), size := SIZEOF(data), sender :=
sender, broadcast := broadcast);
      IF rc < 0 THEN
         no_data := NOT no_data;
      ELSE
         no_data := FALSE;
         size := rc;
         rc := rfSend(receiver := sender, data := ADDR(data), size := size);
         DebugFmt(message := "echo : rfSend(...) = \1", v1 := rc);
      END_IF;
   END_IF;
   sending := send;
   receiving := echo;
END;
END_PROGRAM;
```

4.2.37.6 rfSetPower (Function)

Architecture: X32 / NX32L
Device support: AX9 pro, CX1 pro/pro-c/warp/warp-c, SX1, AX9 turbo, NX-900, LX5
Firmware version: 2.20 / 1.51.00

rfSetPower will change the transmission power of the device.

As stated in RF Functions, many factors influence the range when communicating over the RF ISM frequency band. The most significant is the power of the transmission signal.

The RTCU is able to control the power in 6 levels - from 0 (no transmission) to 5 (full power). The environment has a great impact on the transmission range.
Also be aware that a lower transmission speed as set with rfOpen has a big impact on the transmission range.

A high power level may result in a dead zone around the transmitting device where the signal is too powerful for the AGC on the peer node to work reliably.

As a special case, a power level of 0 (zero) will disable the transmission completely. This may be used as a central "switch" to conveniently disable transmission.

Input:

power : SINT (0 .. 5) (*default 5*)
Transmission power where "0" is no transmission, "1" minimum and "5" is maximum.

Returns: INT

0 - Ok.
- 1 - RF not open. Use rfOpen to open interface.
- 4 - RF not supported on device.

- 6 - Invalid parameter.

Declaration:

```
FUNCTION rfSetPower : INT;
VAR_INPUT
   power    : SINT := 5; // Transmission power, where '0' is no transmission,
'1' minimum and '5' is maximum.
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   open      : BOOL;
END_VAR;

BEGIN
  open := (rfOpen(id := 1) = 0);
  rfSetPower(power := 1); // close range transmission.
  ...
  rfClose();
END;
END_PROGRAM;
```

4.2.37.7 rfSetAntennaMode (Function)

Architecture: X32
Device support: SX1
Firmware version: 2.70

Switch between the on-board internal and optional external antenna. The selection is volatile and the default state after restart is to use the internal antenna.

Input:

mode : SINT (1, 2)
Antenna mode where "1" internal and "2" is external.

Returns: INT

0 - Successful.
1 - Illegal mode.

Declaration:

```
FUNCTION rfSetAntennaMode : INT;
VAR_INPUT
   mode : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;

// switch to external antenna
```

```
rfSetAntennaMode(mode := 2);
BEGIN
    ...
END;
END_PROGRAM;
```

4.2.38 rfbc: RFBC functions

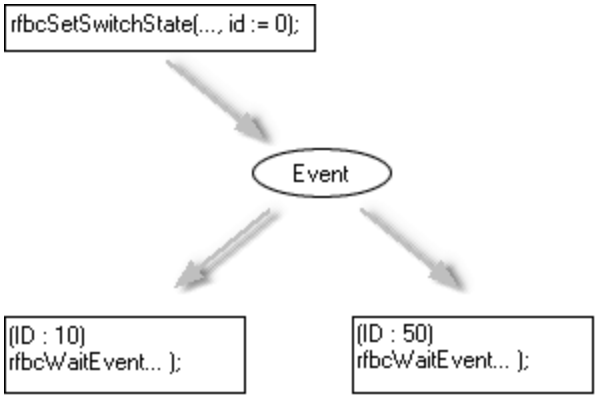
4.2.38.1 rfbc: RFBC Functions

The RFBC functions offer bidirectional communication functionality for close to medium range wireless communication in the ISM frequency band.
The RFBC functionality requires that RF is present. The RF functionality is not available on all RTCU products. Therefore please refer to the technical documentation on the specific product for further information.

The RFBC protocol offers safe bidirectional communication with increased operational reliability and provides the basis for an extensive range of devices for remote control and monitoring.
To increase the security of systems that use the RFBC protocol, the concept of pairing devices is utilized. Pairing is a way to associate a sender (for example a remote control) with a receiver (for example a switch adapter) so that the receiver will only accept requests from that specific sender.

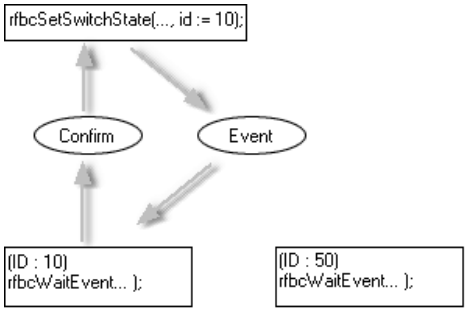
The RFBC protocol requires devices to have a unique ID which is used to identify the sender and receiver when sending requests and receiving confirmations. If a receiver ID of zero is used when sending a request, it is considered to be broadcast to all receivers and no confirmation is returned.
The illustrations below shows the difference between a normal request and a broadcast request.

Broadcasting



When the receiver ID is zero, the request is broadcast to any listening devices regardless of their IDs.

Addressing



When the receiver ID is not zero, only the device with the ID will receive the request. Reception will be confirmed by sending a confirmation back.

Please note that some devices will not accept broadcast requests or will only accept broadcast requests until they are paired.

The RFBC protocol is designed to be compatible with other home automation systems, like the HomeMatic system, and is verified to work with the following products:

Manufacturer	Type/Model	Description
eQ-3 AG.	HM-RC-4-B / HM-RC-4.	4 buttons remote control.
eQ-3 AG.	HM-RC-Key3.	3 buttons remote control.
eQ-3 AG.	HM-PBI-4-FM.	4 channel push-button remote.
eQ-3 AG.	HM-LC-Sw2-FM.	2 channel flush-mount switch adapter.
eQ-3 AG.	HM-LC-Sw1-P.	1 channel socket switch adapter.
eQ-3 AG.	HM-WDS30-T-O.	Outdoor temperature sensor.
eQ-3 AG.	HM-Sec-MDIR.	Indoor motion sensor.

.
For technical information on the above products, please see www.homematic.com.

Architecture of the RFBC API

The RFBC API is based on an event-based model where the function rfbcWaitEvent is used to return information about the various events generated from compatible products. Incoming events are only received and handled as long as a thread is waiting for an event by calling rfbcWaitEvent.
It is possible to send events/requests while another thread is waiting for incoming events and it is therefore encouraged to use multithreading to accomplish a responsive application.

Please consult rfbcWaitEvent for detailed information about the various events and how they are handled.

The following RFBC functions are available:

- rfbcPresent — Queries if RFBC functionality is present.
- rfbcOpen — Opens the RFBC interface.
- rfbcClose — Closes the RFBC interface.
- rfbcGetConfig — Reads configuration parameters.
- rfbcSetConfig — Writes configuration parameters.
- rfbcWaitEvent — Waits for an event (request or notification) to occur.
- rfbcPairRequestReceive — Gets pair request.
- rfbcRawEventReceive — Gets raw/filtered event.
- rfbcRawFilter — Sets the raw event filter.
- rfbcRawSend — Sends a raw RFBC packet/event.
- rfbcRemoteControlEventReceive — Gets remote control event.
- rfbcSwitchEventReceive — Gets switch change notification.
- rfbcTemperatureEventReceive — Gets temperature notification.
- rfbcSensorEventReceive — Gets remote sensor notification.
- rfbcSendAck — Sends an acknowledge to the last received packet.
- rfbcSendButtonPress — Sends a button press event.
- rfbcSetSwitchState — Sends a switch change request.

4.2.38.2 rfbcPresent (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 2.60 (will report FALSE on earlier) / 1.51.00

This function will determine if the necessary functionality is present to support the RFBC protocol on the RTCU device.
If this function returns true, the rfbcOpen can be used to open the interface.

Input:
None.

Returns: BOOL
TRUE - RFBC functionality is present.

FALSE - RFBC functionality is not present.

Declaration:
```
FUNCTION rfbcPresent : BOOL;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
IF rfbcPresent() THEN
  rfbcOpen();
  ...
ELSE
   DebugMsg(message := "RFBC hardware functionality is not present on this
platform, or not supported by this firmware.");
END_IF;

BEGIN
END;
END_PROGRAM;
```

4.2.38.3 rfbcOpen (Function)

Architecture: X32 / NX32L
Device support: AX9 pro, CX1 pro/pro-c/warp/warp-c, SX1, AX9 turbo, NX-900, LX5
Firmware version: 2.60 / 1.51.00

This function will open the RFBC protocol interface. Calling this function is required before the RFBC interface can be used.
It is necessary to call rfbcClose after using the interface.

Input:
None.

Returns: INT
0 - Ok. Interface is open.
-2 - Interface already open or RF module locked by another interface (see rfClose()).
-4 - RF communication is not available.

Declaration:
```
FUNCTION rfbcOpen : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;

BEGIN
   rc := rfbcOpen();
   IF rc <> 0 THEN
      DebugFmt(message := "Error: rfbcOpen = \1", v1 := rc);
   END_IF;

   ...

   // Close the rfbc interface
   rfbcClose();

   ...
END;
END_PROGRAM;
```

4.2.38.4 rfbcClose (Function)

Architecture: X32 / NX32L
Device support: AX9 pro, CX1 pro/pro-c/warp/warp-c, SX1, AX9 turbo, NX-900, LX5
Firmware version: 2.60 / 1.51.00

This function will close the RFBC protocol interface. After the interface is closed, it cannot be used until opened again. See rfbcOpen on how to open the RFBC interface.
When rfbcClose is called, any threads blocked in a rfbc function will resume operation with error code -1.

Input:
None.

Returns: INT
0 - Ok. Interface is closed.
-1 - Interface is not open (see rfbcOpen).
-4 - RF communication is not available.

Declaration:
```
FUNCTION rfbcClose : INT;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM example;

BEGIN
   rc := rfbcOpen();
   IF rc <> 0 THEN
      DebugFmt(message := "Error: rfbcOpen = \1", v1 := rc);
   END_IF;

   ...

   // Close the rfbc interface
   rfbcClose();

   ...
END;
END_PROGRAM;
```

4.2.38.5 rfbcGetConfig (Function)

Architecture: X32 / NX32L
Device support: AX9 pro, CX1 pro/pro-c/warp/warp-c, SX1, AX9 turbo, NX-900, LX5
Firmware version: 2.60 / 1.51.00

This function will return the current configuration used when communication with other devices through the RFBC protocol.
To change configuration parameters, use rfbcSetConfig.
Configuration will be remembered between rfbcClose and rfbcOpen but not through a device reset.

When the interface is first opened, it will have a default configuration based on the serial number of the device.

Input:

None.

Output:

id : DINT
The unique ID used when communicating with other devices to tell who sent the message and whether it is for this device and should be replied.
Only messages which are sent to this id or broadcasted will be reported.

serialNumber : STRING
Serial number used when processing pairing requests.

pairMode : SINT
This defines how incoming pairing requests are handled.
'0' - Rejects all.
'1' - Accepts all.

acknowledge : BOOL (*default : TRUE*)
If true, then a confirmation is sent back when a message addressed to the RTCU is received.

Returns: INT

0 - Successful.
-1 - Interface is not open (see rfbcOpen).
-4 - RF communication is not available.

Declaration:

```
FUNCTION rfbcGetConfig : INT;
VAR_INPUT
   id             : ACCESS DINT;
   serialNumber   : ACCESS STRING;
   pairMode       : ACCESS SINT;
   acknowledge    : ACCESS BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   myId             : DINT;
   mySerialNumber   : STRING;
   myPairMode       : SINT;
   myAcknowledge    : BOOL;
END_VAR;

rfbcOpen();
rfbcGetConfig(id := myId, serialnumber := mySerialNumber, pairmode :=
myPairMode, acknowledge := myAcknowledge);

BEGIN
   ...
END;
END_PROGRAM;
```

4.2.38.6 rfbcSetConfig (Function)

Architecture: X32
Device support: AX9 pro, CX1 pro/pro-c/warp/warp-c, SX1, AX9 turbo, NX-900, LX5
Firmware version: 2.60 / 1.51.00

This function will set the current configuration used when communication with other devices through the RFBC protocol. To fetch the current configuration, use rfbcGetConfig.
Configuration is persistent between rfbcClose and rfbcOpen but not through a device reset.

When the interface is first opened, it will have a default configuration based on the serial number of the device until this is changed by the function.
Note: if a parameter is not given, previously set values will be reverted to default.

Input:

id : DINT (0..16777215, *default: 0*)
The unique ID used when communicating with other devices.
If the ID is set to zero, the unique ID will be generated based on the serial number of the device.

serialNumber : STRING (fixed length of 10)
Serial number used when processing pairing requests. If this is empty, the serial number of the device will be used when auto-generating.

pairMode : SINT (0/1, *default: 0*)
This defines how incoming pairing requests are handled.
'0' - Rejects all.
'1" - Accepts all.

acknowledge : BOOL (*default : TRUE*)
If true, then a confirmation is sent back when a message addressed to the RTCU is received.

Returns: INT

0 - Successful.
-1 - Interface is not open (see rfbcOpen).
-4 - RF communication is not available.
-6 - Error in parameter.

Declaration:

```
FUNCTION rfbcSetConfig : INT;
VAR_INPUT
   id             : DINT   := 0; //    Auto generate from serial number
   serialNumber   : STRING := ""; //   Auto generate from serial number
   pairMode       : SINT   := 0; //    reject requests
   acknowledge    : BOOL   := TRUE; // send acknowledges
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
rfbcOpen();
rfbcSetConfig(pairmode := 1);
BEGIN
  ...
```

```
END;
END_PROGRAM;
```

4.2.38.7 rfbcWaitEvent (Function)

```
Architecture:       X32 / NX32L
Device support:     AX9 pro, CX1 pro/pro-c/warp/warp-c, SX1, AX9
                    turbo, NX-900, LX5
Firmware version:   2.60 / 1.51.00
```

This function will wait until an event is raised. It has an optional timeout.

An event is a notification that something has happened in the RFBC interface.
There are three types of events:

1. Unknown: a message which could not be processed was received. No acknowledgment has been sent.
2. Notification: a message broadcasted from another device. No acknowledgment has been sent.
3. Requests/updates: a message addressed to this device. The device will process and acknowledge according to current configuration (see rfbcGetConfig/rfbcSetConfig).

The events are queued until appropriate action is taken as described in this table:

Event#	Event	Type	Action
1	An unknown message was received.	1	None.
2	A pair request was received.	3	Call rfbcPairRequestReceive.
4	A remote control event was received.	2/3	Call rfbcRemoteControlEventReceive.
5	A sensor notification was received.	2/3	Call rfbcSensorEventReceive.
6	A switch notification was received.	2/3	Call rfbcSwitchEventReceive.
7	A temperature notification was received.	2/3	Call rfbcTemperatureEventReceive.
100	A raw event notification was received.	2/3	Call rfbcRawEventReceive.

Only a single event of the type 2 and 3 can be handled at a time. This means that if the appropriate action is not taken, rfbcWaitEvent will continue to report the same event.

Note: waiting for an event using this function will block the calling thread.

Input:

timeout : INT (-1,0..32000) *(default -1)*
Timeout period in milliseconds to wait.
0 - Disable timeout. The function will return immediately.
-1 - Wait forever. The function will only return when data is received.

Output:

id : DINT
The ID of the sending device. This ID must be used when sending direct requests/notifications to a specific device.

broadcast : BOOL
If true, then this message was not sent to a specific device and was therefore not replied with an acknowledgment to confirm the reception.

Returns: INT

7 - A temperature notification was received.
6 - A switch notification was received.
5 - A sensor notification was received.
4 - A remote control event was received.

2 - A pair request was received.
1 - An unknown request was received.
0 - Timeout.
-1 - Interface is not open (see rfbcOpen).
-4 - RF communication is not available.
-6 - Invalid time.

Declaration:

```
FUNCTION rfbcWaitEvent : INT;
VAR_INPUT
   id       : ACCESS DINT;
   timeout  : DINT := -1;
   broadcast: ACCESS BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

//---------------------------------------------------------------------------
-
// THREAD_BLOCK rfbcMonitor
//---------------------------------------------------------------------------
-
THREAD_BLOCK rfbcMonitor;
VAR
   event       : INT := 0;
   hisId       : DINT;
   broadcast   : BOOL;
END_VAR;

WHILE event <> -1 DO
   event := rfbcWaitEvent(timeout := -1, id := hisId, broadcast := broadcast);
   CASE event OF
      1   : DebugFmt(message := "Unknown message received from '\4'", v4 :=
hisId);
      2   : GetPairEvent(id := hisId);
      4   : GetRemoteEvent(id := hisId);
      5   : GetSensorEvent(id := hisId);
      6   : GetSwitchEvent(id := hisId);
      7   : GetTemperatureEvent(id := hisId);
      100 :
         IF NOT broadcast THEN rfbcSendAck(); END_IF;
         GetRawEvent(id := hisId);
   ELSE
      DebugFmt(message := "rfbcWaitEvent - event=\1", v1 := event);
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;
```

4.2.38.8 rfbcPairRequestReceive (Function)

Architecture: X32 / NX32L
Device support: AX9 pro, CX1 pro/pro-c/warp/warp-c, SX1, AX9 turbo, NX-900, LX5
Firmware version: 2.60 / 1.51.00

This function will return details about the last pair request event received.
To get the ID of the sender related to the event, please call rfbcWaitEvent before this function.

As incoming messages are filtered on the receiver address ("0" or our ID), the actual pair data is not stored automatically by the firmware.

Note: the actual pairing is dependent on the current configuration (see rfbcGetConfig/rfbcSetConfig).

Note: when pairing with the eQ-3 HM-WDS30-T-O, the actual pair has not taken place no matter the configuration. This device will continuously transmit the current temperature.

Input:
None.

Output:
serialNumber : STRING
The serial number of the device requesting the pairing.

Returns: INT
0 - Ok. Event cleared.
-1 - Interface is not open (see rfbcOpen).
-4 - RF communication is not available.
-5 - Event is not present.

Declaration:
```
FUNCTION rfbcPairRequestReceive : INT;
VAR_INPUT
   serialNumber : ACCESS STRING;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

//---------------------------------------------------------------------------
-
// FUNCTION GetPairEvent
//---------------------------------------------------------------------------
-
FUNCTION GetPairEvent;
VAR_INPUT
   id : DINT;
END_VAR;
VAR
   serial : STRING;
END_VAR;
   rfbcPairRequestReceive(serialnumber := serial);
   DebugFmt(message := "Device id \4 serial number '" + serial +
                       "' has sent a pairing request.", v4 := id);
END_FUNCTION;

//---------------------------------------------------------------------------
-
// THREAD_BLOCK rfbcMonitor
//---------------------------------------------------------------------------
-
THREAD_BLOCK rfbcMonitor;
VAR
   event       : INT := 0;
   hisId       : DINT;
   broadcast   : BOOL;
END_VAR;

WHILE event <> -1 DO
   event := rfbcWaitEvent(timeout := -1, id := hisId, broadcast := broadcast);
```

```
   CASE event OF
      ...
      2 : GetPairEvent(id := hisId);
      ...
   ELSE
      DebugFmt(message := "rfbcWaitEvent - event=\1", v1 := event);
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;
```

4.2.38.9 rfbcRawEventReceive (Function)

Architecture: X32 / NX32L
Device support: AX9 pro, CX1 pro/pro-c/warp/warp-c, SX1, AX9 turbo, NX-900, LX5
Firmware version: 2.86 / 1.51.00

This function will return details about the last raw event received which matches the filter set by rfbcRawFilter.
To get the ID of the sender related to the event, please call rfbcWaitEvent before this function.

Input:

datasize : INT [0 .. 54]
Maximum size to load into "data".

data : ADR
Address of the memory block that receives the payload.

Output:

length : INT;
Length of payload loaded into "data".

number : INT;
Packet count number.

flags : INT;
Packet flags.

type : INT;
Packet message type.

Returns: INT

0 - Ok. Event cleared.
-1 - Interface is not open (see rfbcOpen).
-4 - RF is communication not available.
-5 - Event is not present.

Declaration:

```
FUNCTION rfbcRawEventReceive : INT;
VAR_INPUT
   datasize : INT;
   data     : PTR;
   length   : ACCESS INT;
   number   : ACCESS INT;
   flags    : ACCESS INT;
   type     : ACCESS INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

//---------------------------------------------------------------------------
-
// FUNCTION GetRawEvent
//---------------------------------------------------------------------------
-
FUNCTION GetRawEvent;
VAR_INPUT
   id       : DINT;
END_VAR;
VAR
   data     : ARRAY[1..100] OF SINT;
   length   : INT;
   number   : INT;
   flags    : INT;
   type     : INT;
   rc       : INT;
END_VAR;

rfbcRawEventReceive(
   datasize := SIZEOF(data),
   data     := ADDR(data),
   length   := length,
   number   := number,
   flags    := flags,
   type     := type);

DebugFmt(message := "Received a message from \4", v4 := id);
DebugFmt(message := "  Message type   = \1", v1 := type);
DebugFmt(message := "  Message flags  = \1", v1 := flags);
DebugFmt(message := "  Message number = \1", v1 := number);
DebugFmt(message := "  Message length = \1", v1 := length);
DebugFmt(message := "  Message = " + strFromMemory(src := ADDR(data), len :=
length));
END_FUNCTION;

//---------------------------------------------------------------------------
-
// THREAD_BLOCK rfbcMonitor
//---------------------------------------------------------------------------
-
THREAD_BLOCK rfbcMonitor;
VAR
   event       : INT := 0;
   hisId       : DINT;
   broadcast   : BOOL;
END_VAR;

WHILE event <> -1 DO
   event := rfbcWaitEvent(timeout := -1, id := hisId, broadcast := broadcast);
   CASE event OF
      ...
      100 :
         IF NOT broadcast THEN rfbcSendAck(); END_IF;
         GetRawEvent(id := hisId);
      ...
   ELSE
      DebugFmt(message := "rfbcWaitEvent - event=\1", v1 := event);
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;
```

4.2.38.10 rfbcRawFilter (Function)

Architecture: X32 / NX32L
Device support: AX9 pro, CX1 pro/pro-c/warp/warp-c, SX1, AX9 turbo, NX-900, LX5
Firmware version: 2.86 / 1.51.00

This function will set a reception filter for raw packet handling.
Any packet matching the filter must be handled manually - including acknowledgment. No automatic acknowledgment will be sent.

Input:

start_id : DINT [0 .. 16_777_215] (default 0)
ID of the sender must be greater or equal if not set to 0.

end_id : DINT [0 .. 16_777_215] (default 0)
ID of the sender must be less or equal if not set to 0.

flags : INT [-1, 0 .. 255] (default -1)
Packet flags must match this value if greater than -1.

type : INT [-1, 0 .. 255] (default -1)
Packet message type must match this value if greater than -1.

Output:

None.

Returns: INT

0 - Ok filter set
-1 - Interface is not open (see rfbcOpen).
-4 - RF communication is not available.
-6 - Error in parameter.

Declaration:

```
FUNCTION rfbcRawFilter : INT;
VAR_INPUT
   start_id  : DINT;
   end_id    : DINT;
   flags     : INT;
   type      : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
BEGIN
   IF addFilter THEN
       rfbcRawFilter(start_id := 16#00_00_00, end_id := 16#00_FF_FF);
   ELSIF removeFilter THEN
       rfbcRawFilter();
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.38.11 rfbcRawSend (Function)

```
Architecture:       X32 / NX32L
Device support:     AX9 pro, CX1 pro/pro-c/warp/warp-c, SX1, AX9
                    turbo, NX-900, LX5
Firmware version:   2.86 (3.10: wait) / 1.51.00
```

This function will send a raw RFBC packet.
If a destination address is given an acknowledgment is expected to be received.
If no destination address is given the function simply returns without waiting for an acknowledge.

Input:

id : DINT [0 .. 16_777_215] (default 0)
ID of the device the request is sent to.
0 = Broadcast notification to all listening devices.

number : INT [-1,0 .. 255] (default -1)
This sets the current packet number.
If "-1", then last packet number +1 will be used.

flags : INT [0 .. 255] (default 0)
Packet flags.

type : INT [0 .. 255] (default 0)
Packet message type.

datasize : INT [0 .. 54]
Size of data.

data : PTR
Address of the memory block that will be sent as a payload.

wait : BOOL [TRUE, FALSE] (default TRUE)
Wait for acknowledgment of reception, if set to FALSE then the function will return without waiting for an acknowledgment after transmitting the packet once.

Output:

None.

Returns: INT

0 - Success.
-1 - Interface is not open (see rfbcOpen).
-4 - RF communication is not available.
-5 - Communication error - addressed message sent but no acknowledgment received.
-6 - Error in parameter.

Declaration:

```
FUNCTION rfbcRawSend : INT;
VAR_INPUT
   id         : DINT := 0;
   number     : DINT := -1;
   flags      : INT  := 0;
   type       : INT  := 0;
   datasize   : INT;
   data       : PTR;
   wait       : BOOL := TRUE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   send     : BOOL R_EDGE;
END_VAR;

PROGRAM example;
VAR
  data  : ARRAY[1..54] OF SINT;
  msg   : STRING := "Hello world!";
END_VAR;

BEGIN
  IF send THEN
    // Copy contents of a string to memory
    strToMemory(dst:=ADDR(data), str:=msg, len:=strLen(str:=msg));
    // Broadcast the message as a raw packet
    rfbcRawSend(data:=ADDR(data), datasize:=strLen(str:=msg));
  END_IF;
  ...
END;
END_PROGRAM;
```

4.2.38.112 rfbcRemoteControlEventReceive (Function)

Architecture: X32 / NX32L
Device support: AX9 pro, CX1 pro/pro-c/warp/warp-c, SX1, AX9 turbo, NX-900, LX5
Firmware version: 2.60 / 1.51.00

This function will return details about the last remote control event received.
To get the ID of the sender related to the event, please call rfbcWaitEvent before this function.

Note: if configured to send an acknowledgment, then a switch on/off confirmation will be sent based on the button number ("off" for even numbers and "on" for odd numbers).

Input:

None.

Output:

button : **SINT**
The index of the button that is pressed. On some devices this may also be referred to as a channel number.

longPress : BOOL
TRUE - The button is pressed and held. The definition of a long press depends on the configuration of the sending remote control device.
FALSE - The button is pressed.

lowBattery : BOOL
TRUE - Battery level of the remote control is low/critical.
FALSE - Battery level of the remote control is normal.

Returns: INT

0 - Ok. Event cleared.

-1 - Interface is not open (see rfbcOpen).
-4 - RF communication is not available.
-5 - Event is not present.

Declaration:

```
FUNCTION rfbcRemoteControlEventReceive : INT;
VAR_INPUT
   button      : ACCESS SINT;
   longPress   : ACCESS BOOL;
   lowBattery  : ACCESS BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

//-------------------------------------------------------------------------------
-
// FUNCTION GetRemoteEvent
//-------------------------------------------------------------------------------
-
FUNCTION GetRemoteEvent;
VAR_INPUT
   id : DINT;
END_VAR;
VAR
   button   : SINT;
   long     : BOOL;
   low      : BOOL;
END_VAR;

rfbcRemoteControlEventReceive(
   button := button, longPress := long, lowBattery := low);

DebugFmt(message := "Remote event received from \4", v4 := id);
DebugFmt(message := "   Button  : \1", v1 := button);
IF long THEN
   DebugMsg(message := "   Long    : Yes");
ELSE
   DebugMsg(message := "   Long    : No");
END_IF;
IF low THEN
   DebugMsg(message := "   Battery : Low");
ELSE
   DebugMsg(message := "   Battery : Normal");
END_IF;

END_FUNCTION;

//-------------------------------------------------------------------------------
-
// THREAD_BLOCK rfbcMonitor
//-------------------------------------------------------------------------------
-
THREAD_BLOCK rfbcMonitor;
VAR
   event       : INT := 0;
   hisId       : DINT;
   broadcast   : BOOL;
END_VAR;

WHILE event <> -1 DO
   event := rfbcWaitEvent(timeout := -1, id := hisId, broadcast := broadcast);
```

```
   CASE event OF
      ...
      4 : GetRemoteEvent(id := hisId);
      ...
   ELSE
      DebugFmt(message := "rfbcWaitEvent - event=\1", v1 := event);
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;
```

4.2.38.13 rfbcSwitchEventReceive (Function)

Architecture: X32 / NX32L
Device support: AX9 pro, CX1 pro/pro-c/warp/warp-c, SX1, AX9 turbo, NX-900, LX5
Firmware version: 2.60 / 1.51.00

This function will return details about the last switch notification event received.
To get the ID of the sender related to the event, please call rfbcWaitEvent before this function.

Note: many switches will send a notification with channel ID "0" when rebooted/powered on.

Input:

None.

Output:

ch : **SINT**
The channel on the switch which has been changed.
If the channel is "0" (zero), then it is a wake-up event notifying about the fact that the switch has been without power.

state : BOOL
TRUE/ON - On notification.
FALSE/OFF - Off notification.

Returns: INT

0 - Ok. Event cleared.
-1 - Interface is not open (see rfbcOpen).
-4 - RF communication is not available.
-5 - Event is not present.

Declaration:

```
FUNCTION rfbcSwitchEventReceive : INT;
VAR_INPUT
   ch    : ACCESS SINT;
   state : ACCESS BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

//-------------------------------------------------------------------------
-
// FUNCTION GetSwitchEvent
//-------------------------------------------------------------------------
-
FUNCTION GetSwitchEvent;
```

```
VAR_INPUT
   id     : DINT;
END_VAR;
VAR
   ch    : SINT;
   state : BOOL;
END_VAR;
rfbcSwitchEventReceive(ch := ch, state := state);
IF ch = 0 THEN
   DebugFmt(message := "Switch \4 is powered on.", v4 := id);
ELSIF state THEN
   DebugFmt(message := "Channel \1 on switch \4 is powered on.", v4 := id, v1
:= ch);
ELSE
   DebugFmt(message := "Channel \1 on switch \4 is powered off.", v4 := id, v1
:= ch);
END_IF;
END_FUNCTION;

//-------------------------------------------------------------------------------
-
// THREAD_BLOCK rfbcMonitor
//-------------------------------------------------------------------------------
-
THREAD_BLOCK rfbcMonitor;
VAR
   event       : INT := 0;
   hisId       : DINT;
   broadcast   : BOOL;
END_VAR;

WHILE event <> -1 DO
   event := rfbcWaitEvent(timeout := -1, id := hisId, broadcast := broadcast);
   CASE event OF
      ...
      6 : GetSwitchEvent(id := hisId);
      ...
   ELSE
      DebugFmt(message := "rfbcWaitEvent - event=\1", v1 := event);
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;
```

4.2.38.14 rfbcTemperatureEventReceive (Function)

Architecture: X32 / NX32L
Device support: AX9 pro, CX1 pro/pro-c/warp/warp-c, SX1, AX9 turbo, NX-900, LX5
Firmware version: 2.60 / 1.51.00

This function will return details about the last temperature notification event received.
To get the ID of the sender related to the event, please call rfbcWaitEvent before this function.

Input:

None.

Output:

temperature : INT
The measured temperature in 1/10 deg. Celsius.

humidity : INT
Humidity reported by remote device if supported.
Note: not all temperature sensors report humidity, and in these cases, it will be -9999.

Returns: INT

0 - Ok. Event cleared.
-1 - Interface is not open (see rfbcOpen).
-4 - RF communication is not available.
-5 - Event is not present.

Declaration:

```
FUNCTION rfbcTemperatureEventReceive : INT;
VAR_INPUT
   temperature  : ACCESS INT;
   humidity     : ACCESS INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

//-------------------------------------------------------------------------------
-
// FUNCTION GetTemperatureEvent
//-------------------------------------------------------------------------------
-
FUNCTION GetTemperatureEvent;
VAR_INPUT
   id : DINT;
END_VAR;
VAR
   temperature : INT;
   humidity    : INT;
END_VAR;

rfbcTemperatureEventReceive(temperature := temperature, humidity := humidity);
IF humidity >= 0 THEN
   DebugFmt(message := "Temperature received from \4 : \1/10 (\2)",
            v1 := temperature, v2 := humidity, v4 := id);
ELSE
   DebugFmt(message := "Temperature received from \4 : \1/10",
            v1 := temperature, v4 := id);
END_IF;

END_FUNCTION;

//-------------------------------------------------------------------------------
-
// THREAD_BLOCK rfbcMonitor
//-------------------------------------------------------------------------------
-
THREAD_BLOCK rfbcMonitor;
VAR
   event       : INT := 0;
   hisId       : DINT;
   broadcast   : BOOL;
END_VAR;

WHILE event <> -1 DO
   event := rfbcWaitEvent(timeout := -1, id := hisId, broadcast := broadcast);
   CASE event OF
      ...
```

```
      7 : GetTemperatureEvent(id := hisId);
      ...
   ELSE
      DebugFmt(message := "rfbcWaitEvent - event=\1", v1 := event);
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;
```

4.2.38.15 rfbcSendAck (Function)

Architecture: X32 / NX32L
Device support: AX9 pro, CX1 pro/pro-c/warp/warp-c, SX1, AX9 turbo, NX-900, LX5
Firmware version: 2.86 / 1.51.00

This function will send a default acknowledgment packet based on the last received event. Acknowledgments are always sent even if the last message was a broadcast or the interface is configured not to send acknowledgments (see rfbcSetConfig).

Note: acknowledgments should be sent as soon as possible to avoid the sender repeating the message.

Input:
None.

Output:
None.

Returns: INT
0 - Success
-1 - Interface is not open (see rfbcOpen).
-4 - RF communication is not available.

Declaration:
```
FUNCTION rfbcSendAck : INT;
```

Example:
```
INCLUDE rtcu.inc

//-------------------------------------------------------------------------------
-
// THREAD_BLOCK rfbcMonitor
//-------------------------------------------------------------------------------
-
THREAD_BLOCK rfbcMonitor;
VAR
   event       : INT := 0;
   hisId       : DINT;
   broadcast   : BOOL;
END_VAR;

WHILE event <> -1 DO
   event := rfbcWaitEvent(timeout := -1, id := hisId, broadcast := broadcast);
   CASE event OF
      ...
      100 :
         IF NOT broadcast THEN rfbcSendAck(); END_IF;
         GetRawEvent(id := hisId);
      ...
```

```
   ELSE
      DebugFmt(message := "rfbcWaitEvent - event=\1", v1 := event);
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;
```

4.2.38.16 rfbcSendButtonPress (Function)

Architecture: X32 / NX32L
Device support: AX9 pro, CX1 pro/pro-c/warp/warp-c, SX1, AX9 turbo, NX-900, LX5
Firmware version: 2.60 / 1.51.00

This function will send a remote button press notification.
Most devices will only accept button press requests when paired with the RTCU device.

Note: on some devices, the button number may also be referred to as a channel.

Input:

id : DINT [0 .. 16_777_215] (default 0)
The ID of the device to receive the notification.
If the ID is 0 (zero), the notification will be broadcast.

button : SINT [0 .. 63] (default 1)
The index of the button that is pressed.

long : BOOL (default FALSE)
TRUE - The button is pressed and held down.
FALSE - The button is pressed.

low : BOOL (default FALSE)
TRUE - Report that the RTCU battery level is low.
FALSE - Report that the RTCU battery level is normal.

Output:

None.

Returns: INT

0 - The notification is sent.
-1 - Interface is not open (see rfbcOpen).
-4 - RF communication is not available.
-5 - Communication error - addressed message sent but not confirmed
-6 - Error in parameter.

Declaration:

```
FUNCTION rfbcSendButtonPress : INT;
VAR_INPUT
   id          : DINT := 0;
   button      : SINT := 1;
   long        : BOOL := FALSE;
   low         : BOOL := FALSE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
```

```
PROGRAM example;
rfbcOpen();
BEGIN
  ...
  rfbcSendButtonPress(id := 0, button := 1);
  ...
END;
END_PROGRAM;
```

4.2.38.17 rfbcSensorEventReceive (Function)

Architecture: X32 / NX32L
Device support: AX9 pro, CX1 pro/pro-c/warp/warp-c, SX1, AX9 turbo, NX-900, LX5
Firmware version: 2.86 / 1.51.00

This function will return details about the last sensor notification event received.
To get the ID of the sender related to the event, please call rfbcWaitEvent before this function.

Known sensor values:

Device	Description
HM-Sec-MDIR	First value is a motion detection counter since boot. Motion detection will be reported with a 4-5 minutes interval when continuous motion is detected.

Input:

None.

Output:

ch : **INT**
The channel on the sensor which has been changed.

v1 : INT
First sensor value.

v2 : INT
Second sensor value.

Returns: INT

0 - Ok. Event cleared.
-1 - Interface is not open (see rfbcOpen).
-4 - RF communication is not available.
-5 - Event is not present.

Declaration:

```
FUNCTION rfbcSensorEventReceive : INT;
VAR_INPUT
   ch     : ACCESS INT;
   v1     : ACCESS INT;
   v2     : ACCESS INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

//-------------------------------------------------------------------------
-
// FUNCTION GetSensorEvent
```

```
//-------------------------------------------------------------------------------
-
FUNCTION GetSensorEvent;
VAR_INPUT
   id     : DINT;
END_VAR;
VAR
   ch     : INT;
   v1     : INT;
   v2     : INT;
END_VAR;
rfbcSensorEventReceive(ch := ch, v1 := v1, v2 := v2);
DebugFmt(message := "Channel \1 on sensor \4 reports \2 and \3.",
           v1 := ch, v4 := id, v2 := v1, v3 := v2);

END_FUNCTION;

//-------------------------------------------------------------------------------
-
// THREAD_BLOCK rfbcMonitor
//-------------------------------------------------------------------------------
-
THREAD_BLOCK rfbcMonitor;
VAR
   event       : INT := 0;
   hisId       : DINT;
   broadcast   : BOOL;
END_VAR;

WHILE event <> -1 DO
   event := rfbcWaitEvent(timeout := -1, id := hisId, broadcast := broadcast);
   CASE event OF
      ...
      5 : GetSensorEvent(id := hisId);
      ...
   ELSE
      DebugFmt(message := "rfbcWaitEvent - event=\1", v1 := event);
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;
```

4.2.38.18 rfbcSetSwitchState (Function)

Architecture: X32 / NX32L
Device support: AX9 pro, CX1 pro/pro-c/warp/warp-c, SX1, AX9 turbo, NX-900, LX5
Firmware version: 2.60 / 1.51.00

This function will send a switch-change request.
Most devices will only accept switch-change requests when paired with the RTCU device.

Note: if the request is broadcasted then the function will not wait for any confirmation.

Input:

id : DINT [0 .. 16_777_215] (default 0)
The ID of the device to receive the request.
If the ID is "0" (zero), the request will be broadcast.

output : SINT [0..31] (default 1)
The index of the output to update.

state : BOOL (default ON)
ON/TRUE - Request switch to power on output.
OFF/FALSE - Request switch to power off output.

Output:
None.

Returns: INT
0 - Sent.
-1 - Interface is not open (see rfbcOpen).
-4 - RF communication is not available.
-5 - Communication error - addressed message sent but not confirmed.
-6 - Error in parameter.

Declaration:
```
FUNCTION rfbcSetSwitchState : INT;
VAR_INPUT
   id          : DINT := 0;
   output      : SINT := 1;
   State       : BOOL := ON;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

VAR_INPUT
   hisId : DINT;
END_VAR;

PROGRAM example;
hisId := 208024011 MOD 16#FFFFFF; // id of known device
rfbcOpen();
BEGIN
  ...
 rfbcSetSwitchState(id := hisId, output := 1, state := ON);
  ...
END;
END_PROGRAM;
```

4.2.39 sec: Certificate functions

4.2.39.1 sec: Certificate functions

These functions are used to manage the certificates installed on the device.
See Security Features for more information about certificates.

- secCertificateImport — Import a certificate.
- secCertificateRemove — Remove a certificate.
- secCertificateEnumerate — Enumerate installed certificates.
- secCertificateInformation — Read information about a certificate

4.2.39.2 secCertificateImport (Function)

```
Architecture:       NX32L
Device support:     All
Firmware version:   1.08.00
```

The secCertificateImport function imports a certificate to the device.

Input:

name : STRING
The name to identify the certificate.

certificate : STRING
The file name of the certificate to import.

key : STRING
The file name of an optional private encryption key to use for the certificate.

replace : BOOL *(default FALSE)*
When true an existing certificate will be replaced with the certificate.

Returns: INT

1	- Success.
0	- The function is not supported.
-1	- Could not find the file.
-2	- Illegal parameter.
-3	- Not a valid certificate file.
-4	- Failed to import certificate.
-5	- The certificate is already in store.

Declaration:

```
FUNCTION secCertificateImport : INT;
VAR_INPUT
   name        : STRING;
   certificate : STRING;
   key         : STRING;
   replace     : BOOL := FALSE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;

fsMediaOpen(media := 0);

BEGIN
  ...
  // Import certificate
  secCertificateImport(name := "Certificate", certificate := "A:\cert.pem");
  ...
END;
END_PROGRAM;
```

4.2.39.3 secCertificateRemove (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.08.00

The secCertificateRemove function removes a certificate from the device.

Input:

name : STRING
The name of the certificate.

Returns: INT

1 - Success.
0 - The function is not supported.
-1 - Could not find the certificate.

Declaration:

```
FUNCTION secCertificateRemove : INT;
VAR_INPUT
   name : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;

BEGIN
  ...
  // Remove certificate
  secCertificateRemove(name := "CertificateName");
  ...
END;
END_PROGRAM;
```

4.2.39.4 secCertificateEnumerate (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.08.00

The secCertificateEnumerate function enumerates the certificates installed on the device.

Input:

index : INT
The index of the certificate.

Output:

name : STRING
The name of the certificate.

Returns: **INT**

1 - Success.
0 - The function is not supported.
-1 - Could not find the certificate.

Declaration:

```
FUNCTION secCertificateEnumerate : INT;
VAR_INPUT
   index : INT;
   name  : ACCESS STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

FUNCTION list_certificates
VAR
   i     : INT := 1;
   count : INT := 0;
   name  : STRING;
END_VAR;

   DebugMsg(message := "------------------------------");
   DebugMsg(message := " Certificates:");
   WHILE secCertificateEnumerate(index := i, name := name) = 1 DO
      DebugMsg(message := "   [" + name + "]");
      count := count + 1;
      i     := i + 1;
   END_WHILE;
   IF count < 1 THEN
      DebugMsg(message := "   <NONE>");
   END_IF;

END_FUNCTION;
```

4.2.39.5 secCertificateInformation (Functionblock)

```
Architecture:       NX32L
Device support:     All
Firmware version:   1.08.00
```

The secCertificateInformation function retrieves information about a certificate installed on the device.

Input:

name : STRING
The name of the certificate.

Output:

status : BOOL
True if the certificate is found.

type : SINT
The type of certificate.
- A root CA certificate.
- A certificate chain.

subject : STRING
The subject string.

issuer : STRING
The issuer string.

serial : STRING
The serial number of the certificate.

valid_from : DINT
The time (linsec) from when the certificate is valid.

valid_to : DINT
After this time (linsec), the certificate is not valid.

key : BOOL
True if the certificate includes an encryption key.

Declaration:

```
FUNCTION_BLOCK secCertificateInformation;
VAR_INPUT
   name       : STRING;
END_VAR;
VAR_OUTPUT
   status     : BOOL;
   type       : SINT;
   subject    : STRING;
   issuer     : STRING;
   serial     : STRING;
   valid_from : DINT;
   valid_to   : DINT;
   key        : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   sec : secCertificateInformation;
END_VAR;

FUNCTION show_certificate
VAR_INPUT
   name : STRING;
END_VAR;
VAR
   str  : STRING;
END_VAR;

   // Get information
   sec(name := name);

   // Output
   DebugMsg(message := "-----------------------------");
   DebugMsg(message := " Certificate [" + name + "]");
   IF sec.status THEN
      DebugFmt(message := "   type     = \1", v1 := sec.type);
      DebugMsg(message := "   subject  = [" + sec.subject + "]");
      DebugMsg(message := "   issuer   = [" + sec.issuer + "]");
      DebugMsg(message := "   serial   = [" + sec.serial + "]");
      DebugMsg(message := "   valid");
      DebugFmt(message := "     from   = \4", v4 := sec.valid_from);
      DebugFmt(message := "     to     = \4", v4 := sec.valid_to);
      DebugFmt(message := "   key      = \1", v1 := INT(sec.key));
   ELSE
      DebugMsg(message := "   Not found!");
   END_IF;

END_FUNCTION;
```

4.2.40 ser: Serial port

4.2.40.1 Ser: Serial port

The 'ser' functions are functions that are used for direct access to the RS232 and RS485 ports. Please consult the technical manual for details on how many serial ports a specific device offers.

- serOpen — Opens a serial port.
- serClose — Closes a serial port after use.
- serGetStatus — Get the status of a serial port.
- serSendChar — Sends a single character on a serial port.
- serSendString — Sends a string on a serial port.
- serSendData — Sends a block of memory on a serial port.
- serFrameReceiver — Receives data from a serial port.
- serFrameReceiveDone — Releases the buffer from SerFrameReceiver after use.
- serFlush — Empties the receive buffer.
- serForceDataReady — Forces data to be ready in framed receive.
- serSetHandshake — Sets the handshake control.
- serGetBufferLevel — Returns the amount of buffer used for a serial port.
- serGetCTS — Returns the state of the CTS signal.
- serSetRTS — Sets the RTS signal.
- serSetDTR — Sets the DTR signal.
- serGetDCD — Returns the state of the DCD signal.
- serGetDSR — Returns the state of the DSR signal.
- serSetWakeup — Configures the serial port as a wake-up source for pmSuspend.

Serial port enumeration

The technical manuals and the serial port API enumerates the serial ports differently. Basically, the technical manual enumerates the port starting with 1 and the serial port API starting with 0. The following relationship exists:

Serial port API	Technical Manual	Port
0	1	1st serial port
1	2	2nd serial port
2	3	3rd serial port
3	4	4th serial port

RS485 port enumeration

The table below shows the relation between the Serial port API number and the RS485 channels as documented in the technical manual of the various devices:

Device type	Serial port 0	Serial port 1	Serial port 2	Serial port 3
MX2 turbo	n/a	n/a	RS485 port 1	n/a
MX2 encore / MX2 warp / DX4i warp / flex	n/a	n/a	On-demand RS485 port 1	n/a
MX2i series	Optional RS485 port 1*	n/a	n/a	n/a
DX4i pro / AX9i pro / AX9 turbo	RS485 port 2*	n/a	RS485 port 1	n/a
AX9 encore	On-demand RS485 port 2*	n/a	n/a	n/a

Device type	Serial port 0	Serial port 1	Serial port 2	Serial port 3
NX-200	n/a	n/a	RS485 port 1	n/a
NX-400	n/a	n/a	RS485 port 1	RS485 port 2
NX-900 / NX-910	n/a	n/a	RS485 port 1	RS485 port 2
LX2	n/a	n/a	RS485 port 1	n/a
LX4	n/a	RS485 port 1	RS485 port 2	n/a
LX5	n/a	RS485 port 1	n/a	n/a

*Channels marked with a star ***** indicates that the RS485 shares resources with an RS232 port. Please consult the technical manual for details.*

Primary and Secondary RS485 port
To support a certain degree of portability in the configuration the notion of a Primary and Secondary RS485 port is introduced. This is especially useful when working with the I/O extension so that the configuration will work on any device with a variable number of RS485 ports.

The Primary and Secondary RS485 ports on the various devices are:

Device type	Primary RS485 port	Secondary RS485 port
MX2 turbo	Port 1	Port 1
MX2 encore / MX2 warp / DX4i warp / flex	On-demand port 1	On-demand port 1
MX2i series	Optional port 1	Optional port 1
DX4i pro / AX9i pro / AX9 turbo	Port 1	Port 2
AX9 encore	On-demand port 2	On-demand port 2
NX-200	Port 1	Port 1
NX-400	Port 1	Port 2
NX-900 / NX-910	Port 1	Port 2
LX2	Port 1	Port 1
LX4	Port 1	Port 2
LX5	Port 1	Port 1

Device types not included in the above table does not offer any RS485 functionality.

Hardware handshake
RTS/CTS hardware handshake is available on certain devices such as RS232 port 2 or on certain dynamic ports.

Programming port
The programming/service port either uses a USB interface or an RS232 port shared with the 1st serial port (API port 0).
When using an RS232 based programming port the programming mode is detected by a signal in the cable. Please refer to the technical manual for details.

Dynamic ports
Some devices support externally connected serial ports, e.g. using Bluetooth or USB. These ports are assigned dynamic numbers, outside of the normal range, that are valid only while the port is connected.

4.2.40.2 serOpen (Function)

```
Architecture:       X32 / NX32 / NX32L
Device support:     All
Firmware version:   1.00 / 1.00.00
```

serOpen will open the serial port for direct access in either RS232 or RS485 mode depending on the capabilities of the port.
Please note that a special cable may be required (see ser: Serial port).

Input:

port : SINT (0..127) *(default 0)*
Selects which serial port to use. 0 is the programming port on non-USB devices, 1 is serial port 2, 2 is serial port 3, 3 is serial port 4.
Not all serial ports are available on all devices. Please consult the technical manual of the specific device for more information.
On the NX32L, the port numbers from usbHostGetSerialPort, btSerialPortProfileConnect and btHandleSerialPortIncomingConnection can also be used.

baud : DINT (1200,2400,4800,9600,19200,38400,57600,115200) *(default 9600)*
The desired baud rate.

bit : SINT (7/8) *(default 8)*
Selects the number of bits/character

Note:
7 bit without parity bit only works on LX devices.

parity : SINT (0,1,2) *(default 0)*
Selects the desired parity. 0 is none, 1 is even, and 2 is odd.

stopbit : SINT (1,2) *(default 1)*
Selects the number of stop bits.

Note:
Only 1 stop bit is supported on the RS485 ports on NX devices.

rs485 : BOOL *(default false)*
This sets to true if communication is through the RS485 port, and false if the RS232 port is used.

Returns: INT

0 - Successful operation.
1 - Unsupported baud rate (X32 only).
2 - Baud rate not supported at the selected CPU speed.
4 - Serial port is already open (this might be a firmware option like I/O Extension).
5 - Serial port specified is not present on the device.
6 - Unsupported RS485 option.
7 - Unsupported parameter.

Declaration:

```
FUNCTION serOpen : INT;
VAR_INPUT
   port    : SINT := 0;     // Port number, 0 is programming port on non-USB
devices, 1 is serial port 2 (only available on some devices)
   baud    : DINT := 9600;  // baud rate
(1200,2400,4800,9600,19200,38400,57600)
   bit     : SINT := 8;     // number of bits (7/8)
   parity  : SINT := 0;     // 0=none,1=even,2=odd
   stopbit : SINT := 1;     // number of stopbits (1/2)
   rs485   : BOOL := FALSE; // TRUE if using RS485
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
```

```
VAR_OUTPUT
   led      : BOOL; | Indicates that a serial frame has been received
END_VAR;

PROGRAM test;

VAR
   RX       : serFrameReceiver;
   rxbuffer : ARRAY[0..63] of SINT; // Declare a buffer for receiving frames
from the serial port
   txbuffer : ARRAY[0..63] of SINT; // Declare a buffer for sending frames to
the serial port
   portNum  : SINT := 0;            // Select which port to use for the
communication
END_VAR;

// Open the serial port, set baudrate, parity and number of bits
serOpen(port:=portNum, baud:=9600, bit:=8, parity:=0);
// Enable receiving from the serial port, data will be stored in 'rxbuffer',
start of frame is a <CR> and end of frame is a <LF>
RX(port:=portNum, enable:=TRUE, frame:=ADDR(rxbuffer),
maxSize:=SIZEOF(rxbuffer), sof:=16#0D, eof:=16#0A);

// Send the string <CR>ABC<CR><LF> using a buffer
txbuffer[0]:=16#0D;  // <CR>
txbuffer[1]:=16#41;  // 'A'
txbuffer[2]:=16#42;  // 'B'
txbuffer[3]:=16#43;  // 'C'
txbuffer[4]:=16#0D;  // <CR>
txbuffer[5]:=16#0A;  // <LF>
serSendData(port:=portNum, data:=ADDR(txbuffer), size:=6);

// Send a <CR>
serSendChar(port:=portNum, ch:=16#0D);
// Send the string 'Hello world'
serSendString(port:=portNum, str:="Hello world");
// Send a <CR>
serSendChar(port:=portNum, ch:=16#0D);
// Send a <LF>
serSendChar(port:=portNum, ch:=16#0A);

BEGIN
   // Allow the receive functionblock to be updated
   RX();
   // Check if a frame has been received (at least a <CR> and a <LF> has been
received)
   IF RX.ready THEN
      // Indicate a frame has been received, data is available in 'rxbuffer',
length in 'RX.size'
      led:=NOT led;
      // Here we do something with the frame that has been received
      // ..
      // ..
      // ..
      // Release the buffer for the receiver as we are now finished with the
received data
      serFrameReceiveDone(port:=portNum);
   END_IF;
END;

END_PROGRAM;
```

4.2.40.3 serClose (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

---

serClose will close the serial port. Use of the serial port after calling this function is not allowed.

Input:

port : SINT (0..127) ***(default 0)***
Selects which serial port to close.

Returns:

None.

Declaration:

```
FUNCTION serClose;
VAR_INPUT
   port  : SINT := 0;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   ignition : BOOL;
END_VAR;

PROGRAM test;
VAR
   ser_open : BOOL;
END_VAR;

BEGIN
   ...
   IF ignition THEN
      IF NOT ser_open THEN
         IF serOpen(port:=1,baud:=57600) = 0 THEN
            ser_open := TRUE;
         ELSE
            DebugMsg(message:="serOpen - Failed!");
         END_IF;
      END_IF;
   ELSE
      IF ser_open THEN
         serClose(port:=1);
         ser_open := FALSE;
      END_IF;
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.40.4 serGetStatus (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.10.00

Query the status of a serial port.

Input:

port : SINT (0..127) *(default 0)*
Selects which serial port to query.

Returns: INT

```
 1    - The serial port is open and ready for use.
 0    - This function is not supported.
-1    - The serial port is not present.
-2    - The serial port is not open.
-3    - The serial port is open, but the device is not present.
```

Declaration:

```
FUNCTION serGetStatus : INT;
VAR_INPUT
   port  : SINT := 0;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   status   : INT;
   old      : INT := 999;
END_VAR;

   // Enable USB serial port
   usbHostEnable(port := 1, enable := ON);

BEGIN
   // Get the status of the USB serial port
   status := serGetStatus(port := 10);
   IF status <> old THEN
      CASE status OF
          1: DebugMsg(message := "Serial port:   Open and ready for use!");
          0: DebugMsg(message := "serGetStatus is not supported.");
         -1: DebugMsg(message := "Serial port:   Not present.");
         -2: DebugMsg(message := "Serial port:   Not open.");
         -3: DebugMsg(message := "Serial port:   Device is removed!");
      END_CASE;
   END_IF;
   old := status;
END;
END_PROGRAM;
```

4.2.40.5 serSendChar (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

serSendChar will send a single character on the serial port.
It is not recommended to use this function in RS485 mode due to the incurred overhead.

Input:

port : SINT (0..127) ***(default 0)***
Selects which serial port to use.

ch : SINT
The character to send.

timeout : DINT ***(default -1)***
The number of milliseconds to wait (Only used when hardware handshake is enabled. See serSetHandshake).
0 - Disable timeout. The function will return immediately.
-1 - Wait forever. The function will only return when data is send.
Supported from firmware 4.68 / R1.10.00

Returns: INT

1 - Success.
-1 - The serial port is not open.
-3 - Timeout waiting for send.

Declaration:

```
FUNCTION serSendChar : INT;
VAR_INPUT
   port    : SINT := 0;    // Port number
   ch      : SINT;         // Character to send
   timeout : DINT := -1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_OUTPUT
   led      : BOOL; | Indicates that a serial frame has been received
END_VAR;

PROGRAM test;

VAR
   RX       : serFrameReceiver;
   rxbuffer : ARRAY[0..63] of SINT; // Declare a buffer for receiving frames
from the serial port
   txbuffer : ARRAY[0..63] of SINT; // Declare a buffer for sending frames to
the serial port
   portNum  : SINT := 0;            // Select which port to use for the
communication
END_VAR;

// Open the serial port, set baudrate, parity and number of bits
serOpen(port:=portNum, baud:=9600, bit:=8, parity:=0);
// Enable receiving from the serial port, data will be stored in 'rxbuffer',
start of frame is a <CR> and end of frame is a <LF>
RX(port:=portNum, enable:=TRUE, frame:=ADDR(rxbuffer),
maxSize:=SIZEOF(rxbuffer), sof:=16#0D, eof:=16#0A);

// Send the string <CR>ABC<CR><LF> using a buffer
txbuffer[0]:=16#0D;  // <CR>
txbuffer[1]:=16#41;  // 'A'
txbuffer[2]:=16#42;  // 'B'
txbuffer[3]:=16#43;  // 'C'
txbuffer[4]:=16#0D;  // <CR>
txbuffer[5]:=16#0A;  // <LF>
serSendData(port:=portNum, data:=ADDR(txbuffer), size:=6);
```

```
// Send a <CR>
serSendChar(port:=portNum, ch:=16#0D);
// Send the string 'Hello world'
serSendString(port:=portNum, str:="Hello world");
// Send a <CR>
serSendChar(port:=portNum, ch:=16#0D);
// Send a <LF>
serSendChar(port:=portNum, ch:=16#0A);

BEGIN
   // Allow the receive functionblock to be updated
   RX();
   // Check if a frame has been received (at least a <CR> and a <LF> has been
received)
   IF RX.ready THEN
      // Indicate a frame has been received, data is available in 'rxbuffer',
length in 'RX.size'
      led:=NOT led;
      // Here we do something with the frame that has been received
      // ..
      // ..
      // ..
      // Release the buffer for the receiver as we are now finished with the
received data
      serFrameReceiveDone(port:=portNum);
   END_IF;
END;

END_PROGRAM;
```

4.2.40.6 serSendString (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

---

serSendString will send a string out on the serial port.

Input:

port : SINT (0..127) ***(default 0)***
Selects which serial port to use.

str : STRING
The string to send.

timeout : DINT ***(default -1)***
The number of milliseconds to wait (Only used when hardware handshake is enabled. See serSetHandshake).
0 - Disable timeout. The function will return immediately.
-1 - Wait forever. The function will only return when data is send.
Supported from firmware 4.68 / R1.10.00

Returns: INT

1 - Success.
-1 - The serial port is not open.
-2 The serial port is no longer present.
-3 - Timeout waiting for send.

Declaration:

```
FUNCTION  serSendString;
VAR_INPUT
   port : SINT := 0; // Port number
   str  : STRING;    // String to send
   timeout : DINT := -1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_OUTPUT
   led      : BOOL; | Indicates that a serial frame has been received
END_VAR;

PROGRAM test;

VAR
   RX       : serFrameReceiver;
   rxbuffer : ARRAY[0..63] of SINT; // Declare a buffer for receiving frames
from the serial port
   txbuffer : ARRAY[0..63] of SINT; // Declare a buffer for sending frames to
the serial port
   portNum  : SINT := 0;            // Select which port to use for the
communication
END_VAR;

// Open the serial port, set baudrate, parity and number of bits
serOpen(port:=portNum, baud:=9600, bit:=8, parity:=0);
// Enable receiving from the serial port, data will be stored in 'rxbuffer',
start of frame is a <CR> and end of frame is a <LF>
RX(port:=portNum, enable:=TRUE, frame:=ADDR(rxbuffer),
maxSize:=SIZEOF(rxbuffer), sof:=16#0D, eof:=16#0A);

// Send the string <CR>ABC<CR><LF> using a buffer
txbuffer[0]:=16#0D;  // <CR>
txbuffer[1]:=16#41;  // 'A'
txbuffer[2]:=16#42;  // 'B'
txbuffer[3]:=16#43;  // 'C'
txbuffer[4]:=16#0D;  // <CR>
txbuffer[5]:=16#0A;  // <LF>
serSendData(port:=portNum, data:=ADDR(txbuffer), size:=6);

// Send a <CR>
serSendChar(port:=portNum, ch:=16#0D);
// Send the string 'Hello world'
serSendString(port:=portNum, str:="Hello world");
// Send a <CR>
serSendChar(port:=portNum, ch:=16#0D);
// Send a <LF>
serSendChar(port:=portNum, ch:=16#0A);

BEGIN
   // Allow the receive functionblock to be updated
   RX();
   // Check if a frame has been received (at least a <CR> and a <LF> has been
received)
   IF RX.ready THEN
      // Indicate a frame has been received, data is available in 'rxbuffer',
length in 'RX.size'
      led:=NOT led;
      // Here we do something with the frame that has been received
      // ..
```

```
      // ..
      // ..
      // Release the buffer for the receiver as we are now finished with the
received data
      serFrameReceiveDone(port:=portNum);
   END_IF;
END;

END_PROGRAM;
```

4.2.40.7 serSendData (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

---

serSendData will send the contents of a buffer to the specified serial port.
If the "sof" and "eof" variables are set (different from 0), the frame will be "framed". The "sof" character will be sent followed by the data pointed to by "data" with the length in "size" followed by the "eof" character. If the data pointed to by "data" contains either a "sof" or an "eof" character, the "stuffch" will be output first followed by the "sof"/"eof" character. If "stuffch" is part of the data stream, it will be duplicated. This means that the receiver can detect that a "sof", "eof", or "stuffch" is embedded within the data and therefore able to restore the original data.

Input:

port : SINT (0..127) ***(default 0)***
Selects which serial port to use.

data : PTR
The address of the buffer that contain the data.

size : INT
This is the number of bytes to send from the buffer.

sof : SINT
The start of frame character (indicates when a frame begins).

eof : SINT
The end of frame character (indicates when a frame ends).

stuffch : SINT
The character that will be inserted in the frame if the "sof" or "eof" characters are to be sent in the data stream (provided "stuffch" is part of the data stream, it will then be duplicated).

timeout : DINT ***(default -1)***
The number of milliseconds to wait (Only used when hardware handshake is enabled. See serSetHandshake).
0 - Disable timeout. The function will return immediately.
-1 - Wait forever. The function will only return when data is send.
Supported from firmware 4.68 / R1.10.00

Returns: INT

1 - Success.
-1 - The serial port is not open.
-2 The serial port is no longer present.
-3 - Timeout waiting for send.

Declaration:

```
FUNCTION serSendData;
VAR_INPUT
   port    : SINT := 0; // Port number
   data    : PTR;       // Address of the buffer to send
   size    : INT;       // Number of bytes to send from the buffer
   sof     : SINT;      // Start of frame indicator
   eof     : SINT;      // End of frame indicator
   stuffch : SINT;      // Stuffing character
   timeout : DINT := -1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_OUTPUT
   led      : BOOL; | Indicates that a serial frame has been received
END_VAR;

PROGRAM test;

VAR
   RX       : serFrameReceiver;
   rxbuffer : ARRAY[0..63] of SINT; // Declare a buffer for receiving frames
from the serial port
   txbuffer : ARRAY[0..63] of SINT; // Declare a buffer for sending frames to
the serial port
   portNum  : SINT := 0;            // Select which port to use for the
communication
END_VAR;

// Open the serial port, set baudrate, parity and number of bits
serOpen(port:=portNum, baud:=9600, bit:=8, parity:=0);
// Enable receiving from the serial port, data will be stored in 'rxbuffer',
start of frame is a <CR> and end of frame is a <LF>
RX(port:=portNum, enable:=TRUE, frame:=ADDR(rxbuffer),
maxSize:=SIZEOF(rxbuffer), sof:=16#0D, eof:=16#0A);

// Send the string <CR>ABC<CR><LF> using a buffer
txbuffer[0]:=16#0D;  // <CR>
txbuffer[1]:=16#41;  // 'A'
txbuffer[2]:=16#42;  // 'B'
txbuffer[3]:=16#43;  // 'C'
txbuffer[4]:=16#0D;  // <CR>
txbuffer[5]:=16#0A;  // <LF>
serSendData(port:=portNum, data:=ADDR(txbuffer), size:=6);

// Send a <CR>
serSendChar(port:=portNum, ch:=16#0D);
// Send the string 'Hello world'
serSendString(port:=portNum, str:="Hello world");
// Send a <CR>
serSendChar(port:=portNum, ch:=16#0D);
// Send a <LF>
serSendChar(port:=portNum, ch:=16#0A);

BEGIN
   // Allow the receive functionblock to be updated
   RX();
   // Check if a frame has been received (at least a <CR> and a <LF> has been
received)
   IF RX.ready THEN
      // Indicate a frame has been received, data is available in 'rxbuffer',
```

```
length in 'RX.size'
      led:=NOT led;
      // Here we do something with the frame that has been received
      // ..
      // ..
      // ..
      // Release the buffer for the receiver as we are now finished with the
received data
      serFrameReceiveDone(port:=portNum);
   END_IF;
END;

END_PROGRAM;
```

4.2.40.8 serFrameReceiver (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

---

serFrameReceiver reads a frame of data from the serial port. A "frame of data" is identified by a frameStart character and a frameEnd character. The frameStart character starts the reception, and the frameEnd ends the reception. After the frameEnd character has been received, the buffer (given in "frame") will contain the received data (excluding the frameStart and frameEnd characters). This function block is able to restore the data transmitted by the serSendData() function. If the "sof" (start of frame) character is set to 0, the reception will start with the first character and be terminated with the "eof" character.
When the data has been processed by the application, the function serFrameReceiveDone() must be called to release the receive buffer.

All RTCU devices contain a 16 KByte receive buffer for each of the serial ports available.

Input:

port : SINT (0..127) ***(default 0)***
Selects which serial port to use.

enable : BOOL ***(Default FALSE)***
Set to true if receiving data should be enabled.

frame : PTR
Address of the buffer to receive the data in.

maxSize : INT
This is the maximum size of the receive buffer.

sof : SINT
Start of frame character (indicates when a frame begins). Set to 0 to ignore start character.

eof : SINT
End of frame character (indicates when a frame ends). If set to 0, "maxSize" indicates number of characters to read (excluding the "sof" character).

stuffch : SINT
A duplicated "stuffch" in the data stream will be translated to a single "stuffch". A "stuffch" followed by a "sof" or "eof" will be translated to a "sof"/"eof". If "stuffch" is 0, the stuffing is disabled.

Output:

ready : BOOL
True if a complete frame has been received (the minimum being that the frameStart and frameEnd characters have been received).

size : INT
Number of bytes received (excluding the frameStart and frameEnd characters).

Declaration:

```
FUNCTION_BLOCK serFrameReceiver;
VAR_INPUT
   port     : SINT := 0;     // Port number
   enable   : BOOL := FALSE; // enabled?
   frame    : PTR;           // ptr to frame memory
   maxSize  : INT;           // max size of frame.
   sof      : SINT;          // frame start indicator. Can be 0 for not used.
SOF is implicit on first byte.
   eof      : SINT;          // frame END indicator. Can be 0 for not used.
Will read maxSize chars.
   stuffch  : SINT;          // stuffing character
END_VAR;
VAR_OUTPUT
   ready    : BOOL;          // data is ready
   size     : INT;           // actual size of received frame.
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_OUTPUT
   led      : BOOL; | Indicates that a serial frame has been received
END_VAR;

PROGRAM test;

VAR
   RX       : serFrameReceiver;
   rxbuffer : ARRAY[0..63] of SINT; // Declare a buffer for receiving frames
from the serial port
   txbuffer : ARRAY[0..63] of SINT; // Declare a buffer for sending frames to
the serial port
   portNum  : SINT := 0;            // Select which port to use for the
communication
END_VAR;

// Open the serial port, set baudrate, parity and number of bits
serOpen(port:=portNum, baud:=9600, bit:=8, parity:=0);
// Enable receiving from the serial port, data will be stored in 'rxbuffer',
start of frame is a <CR> and end of frame is a <LF>
RX(port:=portNum, enable:=TRUE, frame:=ADDR(rxbuffer),
maxSize:=SIZEOF(rxbuffer), sof:=16#0D, eof:=16#0A);

// Send the string <CR>ABC<CR><LF> using a buffer
txbuffer[0]:=16#0D;  // <CR>
txbuffer[1]:=16#41;  // 'A'
txbuffer[2]:=16#42;  // 'B'
txbuffer[3]:=16#43;  // 'C'
txbuffer[4]:=16#0D;  // <CR>
txbuffer[5]:=16#0A;  // <LF>
serSendData(port:=portNum, data:=ADDR(txbuffer), size:=6);

// Send a <CR>
```

```
serSendChar(port:=portNum, ch:=16#0D);
// Send the string 'Hello world'
serSendString(port:=portNum, str:="Hello world");
// Send a <CR>
serSendChar(port:=portNum, ch:=16#0D);
// Send a <LF>
serSendChar(port:=portNum, ch:=16#0A);

BEGIN
   // Allow the receive functionblock to be updated
   RX();
   // Check if a frame has been received (at least a <CR> and a <LF> has been
received)
   IF RX.ready THEN
      // Indicate a frame has been received, data is available in 'rxbuffer',
length in 'RX.size'
      led:=NOT led;
      // Here we do something with the frame that has been received
      // ..
      // ..
      // ..
      // Release the buffer for the receiver as we are now finished with the
received data
      serFrameReceiveDone(port:=portNum);
   END_IF;
END;

END_PROGRAM;
```

4.2.40.9 serFrameReceiveDone (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

---

serFrameReceiveDone is called when the buffer from the serFrameReceiver function block is ready to receive new data. This is typically when the program is finished using the data and is awaiting a new frame on the serial port.
It is very important that this function is called with the same port number as the serFrameReceiver functionblock.

Input:

port : SINT (0..127) ***(default 0)***
This selects which serial port to use.

Returns:

None.

Declaration:

```
FUNCTION serFrameReceiveDone;
VAR_INPUT
   port : SINT := 0;    // Port number
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_OUTPUT
```

```
   led      : BOOL; | Indicates that a serial frame has been received
END_VAR;

PROGRAM test;

VAR
   RX       : serFrameReceiver;
   rxbuffer : ARRAY[0..63] of SINT; // Declare a buffer for receiving frames
from the serial port
   txbuffer : ARRAY[0..63] of SINT; // Declare a buffer for sending frames to
the serial port
   portNum  : SINT := 0;            // Select which port to use for the
communication
END_VAR;

// Open the serial port, set baudrate, parity and number of bits
serOpen(port:=portNum, baud:=9600, bit:=8, parity:=0);
// Enable receiving from the serial port, data will be stored in 'rxbuffer',
start of frame is a <CR> and end of frame is a <LF>
RX(port:=portNum, enable:=TRUE, frame:=ADDR(rxbuffer),
maxSize:=SIZEOF(rxbuffer), sof:=16#0D, eof:=16#0A);

// Send the string <CR>ABC<CR><LF> using a buffer
txbuffer[0]:=16#0D;  // <CR>
txbuffer[1]:=16#41;  // 'A'
txbuffer[2]:=16#42;  // 'B'
txbuffer[3]:=16#43;  // 'C'
txbuffer[4]:=16#0D;  // <CR>
txbuffer[5]:=16#0A;  // <LF>
serSendData(port:=portNum, data:=ADDR(txbuffer), size:=6);

// Send a <CR>
serSendChar(port:=portNum, ch:=16#0D);
// Send the string 'Hello world'
serSendString(port:=portNum, str:="Hello world");
// Send a <CR>
serSendChar(port:=portNum, ch:=16#0D);
// Send a <LF>
serSendChar(port:=portNum, ch:=16#0A);

BEGIN
   // Allow the receive functionblock to be updated
   RX();
   // Check if a frame has been received (at least a <CR> and a <LF> has been
received)
   IF RX.ready THEN
      // Indicate a frame has been received, data is available in 'rxbuffer',
length in 'RX.size'
      led:=NOT led;
      // Here we do something with the frame that has been received
      // ..
      // ..
      // ..
      // Release the buffer for the receiver as we are now finished with the
received data
      serFrameReceiveDone(port:=portNum);
   END_IF;
END;

END_PROGRAM;
```

4.2.40.10 serFlush (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

---

serFlush clears the system receive buffer for the chosen port.
The result is that any data not received into the application buffer, but present in the system buffer, will be cleared.

Input:

port : SINT (0..127) ***(default 0)***
Selects which serial port to use.

Returns:

None.

Declaration:

```
FUNCTION serFlush;
VAR_INPUT
   port : SINT := 0;    // Port number
end_var;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   portNum  : SINT := 0;            // Select which port to use for the
communication
END_VAR;

...

BEGIN
   ...
   // Reset buffer cmd received?
   IF .. THEN
      // Clear buffer
      serFlush(port:=portNum);
      ...
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.40.11 serForceDataReady (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

---

serForceDataReady will force that data is made ready (see serFrameReceiver) even if the criteria for a frame has not been met. This will set the "ready" output of the SerFrameReceiver function block to true immediately.

Note: this function will only work when operating in non-framed mode (eof=sof=0).

Input:

port : SINT (0..127) ***(default 0)***
Selects which serial port to use.

Returns:

None.

Declaration:

```
FUNCTION  serForceDataReady;
VAR_INPUT
   port : SINT := 0;    // Port number
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   RX       : serFrameReceiver;
   rxbuffer : ARRAY[0..5] of SINT; // Declare a buffer for receiving frames
from the serial port
   portNum  : SINT := 0;           // Select which port to use for the
communication
   rx_dummy : SINT := 16#FF;
END_VAR;

// Open the serial port, set baudrate, parity and number of bits
serOpen(port:=portNum, baud:=9600, bit:=8, parity:=0);
// Enable receiving from the serial port, data will be stored in 'rxbuffer',
start of frame is a <CR> and end of frame is a <LF>
RX(port:=portNum, enable:=TRUE, frame:=ADDR(rxbuffer),
maxSize:=SIZEOF(rxbuffer));

// Set birst byte in buffer to Dummy
rxBuffer[0] := rx_dummy;

BEGIN
   // Allow the receive functionblock to be updated
   RX();
   // Check if a ModBus frame has been received
   IF rxBuffer[0] <> rx_dummy THEN
      // Frame received
      serForceDataReady(port:=portNum);
      ...
      // Release the buffer for the receiver as we are now finished with the
received data
      serFrameReceiveDone(port:=portNum);
   END_IF;
END;

END_PROGRAM;
```

4.2.40.12 serSetHandshake (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, DX4 pro, AX9 pro, MX2 turbo, AX9 turbo, NX-200, NX-400, NX-900, NX-910, LX2, LX4
Firmware version: 1.00 / 1.00.00

---

serSetHandshake controls the handshaking on the serial port by using CTS and RTS signals.
This function is only supported on serial ports with control signals (typically serial port 1).
By using RTS/CTS handshake, the device will only transmit when CTS is active. RTS will be set to inactive when the receive buffer is more than 80% full, and it will be set back to active when the the receive buffer is less than 40% full.
When sending data it is possible to specify a timeout waiting for the handshake.

Input:

port : SINT (0..127) ***(default 1)***
Selects which serial port to use.

RtsCts : BOOL
If true, RTS and CTS handshaking is active.

Returns: INT

0 - Successful operation.
1 - Specified port does not support this operation.

Declaration:

```
FUNCTION  serSetHandshake : INT;
VAR_INPUT
   port   : SINT := 1;    // Port number, only port:=1 is supported
   RtsCts : BOOL := FALSE;// Enable RTS/CTS handshaking
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Open the serial port, set baudrate, parity and number of bits
serOpen(port:=1, baud:=9600, bit:=8, parity:=0);

// Enable RTS/CTS handshaking
serSetHandshake(port:=1, RtsCts:=TRUE);

BEGIN
   Sleep(Delay:=1000);
   // Send the string <CR>ABC<CR><LF> on the serial port, using handshake
   serSendString(port:=1, str:="$RABC$R$L");
END;

END_PROGRAM;
```

4.2.40.13 serGetBufferLevel (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This function will return the amount of space used in the serial buffer for the specific serial port. The returned value is on a scale between 0 and 1000 (permille). The exact size of the receive buffer is specified in the help section for serFrameReceiver().

Input:

port : SINT (0..127) ***(default 0)***
Selects which serial port to use.

Returns: INT

This is the amount of space used in the serial buffer for that port, 0..1000 per-mille.

Declaration:

```
FUNCTION serGetBufferLevel : INT;
VAR_INPUT
   port : SINT := 0;    // Port number
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   portNum  : SINT := 0;    // Select which port to use for the communication
   bufFull  : BOOL;
   i        : INT;
END_VAR;

BEGIN
   ...
   // Return amount of space used in serial buffer
   i := serGetBufferLevel(port:=portNum);
   IF i > 800 AND NOT bufFull THEN
      // Buffer is to full
      serSetRTS(port:=portNum,state:=ON);
      bufFull := TRUE;
      ...
   ELSIF i < 400 AND bufFull THEN
      // Buffer level OK again
      serSetRTS(port:=portNum,state:=OFF);
      bufFull := FALSE;
      ...
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.40.14 serGetCTS (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, DX4 pro, AX9 pro, MX2 turbo, AX9 turbo, NX-200, NX-400, NX-900, NX-910, LX2, LX4
Firmware version: 1.00 / 1.00.00

serGetCTS returns the state of the CTS signal. This function is only supported on serial ports with control signals (typically serial port 1).
The function does not work when hardware handshake is activated (see the serSetHandshake function).

Input:

port : SINT (0..127) ***(default 1)***
Selects which serial port to use.

Returns: BOOL

TRUE: If the CTS signal is logically asserted.
FALSE: If the CTS signal is logically de-asserted.

Declaration:

```
FUNCTION serGetCTS : BOOL;
VAR_INPUT
   port : SINT := 1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Test for CTS
   IF serGetCTS() THEN
      ...
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.40.15 serSetRTS (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, DX4 pro, AX9 pro, MX2 turbo, AX9 turbo, NX-200, NX-400, NX-900, NX-910, LX2, LX4
Firmware version: 1.00 / 1.00.00

serSetRTS sets the state of the RTS signal. This function is only supported on serial ports with control signals (typically serial port 1).
The function does not work when hardware handshake is activated (see the serSetHandshake function).

Input:

port : SINT (0..127) ***(default 1)***
Selects which serial port to use.

state : BOOL
The logical state of the RTS pin.

Returns:

None.

Declaration:

```
FUNCTION serSetRTS;
VAR_INPUT
   port  : SINT := 1;
   state : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Set RTS
   serSetRTS(state:=ON);
   ...
   // Clear RTS
   serSetRTS(state:=OFF);
   ...
END;

END_PROGRAM;
```

4.2.40.16 serSetDTR (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, DX4 pro, MX2 turbo, NX-200, NX-400, LX2
Firmware version: 4.70 / 1.30.00

---

serSetDTR sets the logical state of the DTR signal. This function is only supported on serial ports with control signals (typically serial port 1).

Input:

port : SINT (0..127) ***(default 1)***
Selects which serial port to use.

state : BOOL
The logical state the DTR signal.

Returns:

None.

Declaration:

```
FUNCTION serSetDTR;
VAR_INPUT
   port  : SINT := 1;
   state : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Set DTR
   serSetDTR(state:=ON);
   ...
   // Clear DTR
   serSetDTR(state:=OFF);
   ...
END;

END_PROGRAM;
```

4.2.40.17 serGetDCD (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, DX4 pro, MX2 turbo, NX-200, NX-400, LX2
Firmware version: 1.00 / 1.00.00

serGetDCD returns the state of the DCD signal. This function is only supported on serial ports with control signals (typically serial port 1).
The function does not work when hardware handshake is activated (see the serSetHandshake function).

Input:

port : SINT (0..127) ***(default 1)***
Selects which serial port to use.

Returns: BOOL

TRUE: If the DCD signal is logically asserted.
FALSE: If the DCD signal is logically de-asserted.

Declaration:

```
FUNCTION serGetDCD : BOOL;
VAR_INPUT
   port : SINT := 1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
```

```
   // Test for DCD
   IF serGetDCD() THEN
      ...
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.40.18 serGetDSR (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2, DX4 pro, MX2 turbo, NX-200, NX-400, LX2
Firmware version: 1.00 / 1.00.00

---

serGetDSR returns the state of the DSR signal. This function is only supported on serial ports with control signals (typically serial port 1).
The function does not work when hardware handshake is activated (see the serSetHandshake function).

Input:

port : SINT (0..127) ***(default 1)***
Selects which serial port to use.

Returns: BOOL

TRUE: If the DSR signal is logically asserted.
FALSE: If the DSR signal is logically de-asserted.

Declaration:

```
FUNCTION serGetDSR : BOOL;
VAR_INPUT
   port : SINT := 1;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Test for DSR
   IF serGetDSR() THEN
      ...
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.40.19 serSetWakeup (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.36.00

This function is used to configure the system to wake from suspend when data is received on a serial port.
It is similar to using the ser0 and ser1 parameters on pmWaitEvent.

pmSuspend return codes for this wake-up source:

Error	Type	Value	Description
True	3	-10	The serial port is not open.
False	3	1	The serial port has awoken the system.

The ID variable is set to the port number of the affected serial port.

Input:

Enable : BOOL (default: TRUE)
Set to true to enable the wake-up, set to false to disable it.

Port : SINT (default: 0)
The serial port to wake on.

Mode : SINT (default: 0)
The type of event to wake on. Not all ports have support for all modes. Mode 0 will wake when data is received. Mode 1 will wake on any change on the bus and the received data is not available from serFrameReceiver.

Returns:

1 - The system has been configured to wake.
0 - This function is not supported.
-1 - The port is not present.
-2 - The port is not open.
-3 - The port does not support wake-up.
-4 - The serial port does not support the mode.
-5 - Unknown mode.

Declaration:

```
FUNCTION ALIGN serSetWakeup:INT;
VAR_INPUT
   enable  : BOOL := TRUE;
   port    : SINT := 0;
   mode    : SINT := 0;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Enable wake on serial port 1
   serSetWakeup(port := 1);
   ...
END;

END_PROGRAM;
```

4.2.41 smtp: SMTP functions

4.2.41.1 smtp: SMTP functions

SMTP functions for sending emails (electronic mail) by using SMTP (Simple Mail Transfer Protocol).
SMTP is an internet standard for email transmission across IP networks that are typically only used for sending messages to a mail server for relaying.

The SMTP functions support two ways of sending emails.

smtpSend, for simple short messages:
Advantages:
- Only the smtpSend function is needed.
- The function waits for the mail transfer to complete.

Disadvantages:
- The function waits for the mail transfer to complete.
- Only a short text message.
- No attachments.

smtpSendX, for advanced messages:
Advantages:
- Large text messages.
- Message text can be built over time.
- File attachments.
- Asynchronous mail transfer (depending on the situation, this can also be a disadvantage).
- Mail transfer progress can be monitored.

Disadvantages:
- Several functions are required to build and send an email.

The following sequence is required to make and send the email:

1.	smtpNew	To create the email.
2.	smtpAddText	To add text to the email message (repeatedly for large messages).
3.	smtpAddAttachment	To attach file(s) to the email.
4.	smtpSendX	To start sending the email.
5.	smtpAwaitCompletion	To wait for the mail transfer to complete (optional).
6.	smtpStatus	To monitor mail transfer progress (optional).

The following functions are used to access the SMTP interface:

• smtpOpen	Opens the SMTP interface.
• smtpClose	Closes the SMTP interface.
• smtpSetConfig	Sets the SMTP configuration.
• smtpGetConfig	Gets the SMTP configuration.
• smtpSend	Sends an email.
• smtpNew	Creates a new email.
• smtpAddText	Adds text to an email.
• smtpAddAttachment	Attaches a file to an email.
• smtpSendX	Starts sending an email.
• smtpCancel	Cancels a mail transfer.
• smtpStatus	Gets the status of an email.
• smtpAwaitCompletion	Waits for mail transfer to be completed.

The implementation of SMTP has the following limitations:

- The outbox can hold two emails simultaneously.
- All email addresses are limited to 40 characters and a single address.
- The message text of an advanced email is restricted to a maximum length of 4095 characters.
- It is possible to attach a maximum of one file to an email.
- A network interface connection must be available.

4.2.41.2 smtpOpen (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.50 / 1.00.00

This function will open the SMTP interface. Calling this function is required before the SMTP interface can be used.
Also see smtpClose for closing the SMTP interface when finished with the operations.

Prior to a message sent through smtpSend or smtpSendX, a SMTP configuration must be a set by using smtpSetConfig, and the configured network interface must have been opened using netOpen.

Input:

None.

Returns: INT

0 - Success.
-1 - General error.
-2 - Already open.

Declaration:

```
FUNCTION smtpOpen : INT;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   alarm : BOOL;
END_VAR;

VAR
   fired   : BOOL;
END_VAR;

PROGRAM example;
VAR
   rc      : INT;
   Subject : STRING;
END_VAR;

   gsmPower(power := ON);
   netOpen(iface:=1);
   smtpOpen();

BEGIN
   // Is there an alarm
   IF alarm AND NOT fired THEN
      Subject := strFormat(format := "Device \4", v4 := boardSerialNumber());
```

```
      rc := smtpSend(Receiver := "devices@alarms.com",
                     Receiver_cc := "cc_devices@alarms.com",
                     Subject := Subject,
                     Message := "Alarm is triggered!");
      IF rc <> 0 THEN
         DebugFmt(message := "Error sending mail (errno=\1)", v1 := rc);
      ELSE
         DebugMsg(message := "Mail sent");
      END_IF;
   END_IF;
   fired := alarm;
END;
END_PROGRAM;
```

4.2.41.3 smtpClose (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.50 / 1.00.00

This function will close the SMTP interface. After the SMTP interface is closed, it cannot be used until opened again.
Please also see smtpOpen.

When closing the SMTP interface, all emails created with smtpNew and smtpSend will be canceled, and all waiting threads (using smtpAwaitCompletion) will be resumed.

Input:

None.

Returns: INT

0 - Success.
-1 - General error.

Declaration:

```
FUNCTION smtpClose : INT;
```

Example:

```
INCLUDE rtcu.inc

VAR
   ignition : BOOL;
END_VAR;

PROGRAM example;
VAR
   smtp_open : BOOL;
END_VAR;

   gsmPower(power := ON);
   netOpen(iface:=1);
BEGIN
   IF ignition THEN
      IF NOT smtp_open THEN
         IF smtpOpen() = 0 THEN
            smtp_open := TRUE;
         ELSE
            DebugMsg(message:="smtpOpen - Failed!");
         END_IF;
```

```
      END_IF;
   ELSE
      IF smtp_open THEN
         smtpClose();
         smtp_open := FALSE;
      END_IF;
   END_IF;
END;

END_PROGRAM;
```

4.2.41.4 smtpSetConfig (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.50 (4.34: iface) / 1.00.00

This function will set the configuration for the SMTP interface.
The SMTP configuration is stored in persistent memory and only needs to be set once.

Using secure connection
For NX32L devices a secure TLS (TLS v1.2, v1.1 or 1.0) connection can be established assuming that a matching X509 root certificate is present.

To use secure connection:
1. Ensure the root certificate needed to verify the mail server is present.
2. Set 'useTLS := TRUE' when calling smtpSetConfig.

Input:

IP : STRING
The IP address of the mail server (max. 40 characters).

Port : DINT *(default 25)*
The port of the mail server.

iface : SINT (*default 0*)
The network interface to use. 0 = Default network, 1 = Mobile network, 2 = LAN network, etc. (See Network)
Note: For backwards compatibility the Mobile network is used as default network.

Sender : STRING
The mail address of the sender. (max. 40 characters).

Authenticate : BOOL
TRUE: Use authentication.
FALSE: Do not use authentication.

Username : STRING
The user name the device should use in order to connect to the mail server (max. 40 characters).

Password : STRING
The password the device should use in order to connect to the mail server (max. 40 characters).

UseTLS : BOOT *(default FALSE)*
TRUE: Use secure connection.
FALSE: Use insecure connection

Returns: INT

0 - Success.
-3 - Invalid parameter

Declaration:

```
FUNCTION smtpSetConfig : INT;
VAR_INPUT
   IP                : STRING;
   Port              : DINT := 25;
   iface             : SINT := 0;
   Sender            : STRING;
   Authenticate      : BOOL;
   Username          : STRING;
   Password          : STRING;
   UseTLS            : BOOL := FALSE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;

// Set SMTP config
smtpSetConfig(IP := "mail.server.com",
              Port := 25,
              iface := 1,
              Sender := strFormat(format := "\4@server.com", v4 :=
boardSerialNumber()),
              Authenticate := ON,
              Username := "DEVICE",
              Password := "12345");

BEGIN
   ...
END;
END_PROGRAM;
```

4.2.41.5 smtpGetConfig (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.50 (4.34: iface) / 1.00.00

This function will read out the SMTP configuration previously set by using the smtpSetConfig function.

Input:

None.

Output:

Status : SINT

0 - SMTP parameters are valid.
-13 - SMTP parameters are not set.

IP : STRING

The IP address of the email server.

Port : DINT
The port of the email server.

iface : SINT
The network interface to use. 0 = Default network, 1 = Mobile network, 2 = LAN network, etc. (See Network)
Note: For backwards compatibility the Mobile network is used as default network.

Sender : STRING
The mail address of the email sender.

Authenticate : BOOL
TRUE: Authentication is enabled.
FALSE: Authentication is not enabled.

Username : STRING
The user name the device should use in order to connect to the mail server.

Password : STRING
The password the device should use in order to connect to the mail server.

UseTLS : BOOT
TRUE: Secure connection is enabled.
FALSE: Secure connection is not enabled.

Declaration:

```
FUNCTION_BLOCK smtpGetConfig;
VAR_OUTPUT
   Status             : SINT;
   IP                 : STRING;
   Port               : DINT;
   iface              : SINT;
   Sender             : STRING;
   Authenticate       : BOOL;
   Username           : STRING;
   Password           : STRING;
   UseTLS             : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   smtpParm  : smtpGetConfig;
END_VAR;

// Check SMTP settings
smtpParm();
DebugMsg(message := "smtpGetConfig:");
DebugFmt(message := "-status=\1", v1 := smtpParm.status);
DebugMsg(message := "-host=" + smtpParm.IP);
DebugFmt(message := "-port=\4", v4 := smtpParm.port);
DebugFmt(message := "-iface=\1", v1 := smtpParm.iface);
DebugFmt(message := "-UseTLS=\1", v1 := SINT(smtpParm.UseTLS));
DebugMsg(message := "-sender=" + smtpParm.sender);
DebugFmt(message := "-auth=\1", v1 := SINT(smtpParm.authenticate));
DebugMsg(message := "-user=" + smtpParm.username);
DebugMsg(message := "-pass=" + smtpParm.password);
```

```
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.41.6 smtpSend (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.50 / 1.00.00

This function will send an email.
The network interface must be connected and the SMTP interface must be configured to use an email server (with smtpSetConfig) before this function is called.
The function creates the email in the outbox and then waits for the mail transfer to complete before returning.
Because of this, the message text is limited to the size of STRING.

Note: the outbox can only contain two emails.

Input:

Receiver : STRING
The email address of the receiver (max. 40 characters).

Receiver_cc : STRING
The email address of an optional receiver (max. 40 characters).

Subject : STRING
The subject of the email (max. 40 characters).

Message : STRING
The text message of the email.

Returns: INT

0	- Success.
-1	- General error.
-2	- SMTP interface is not open.
-3	- Invalid parameter(s).
-4	- Outbox is full.
-7	- Network error.
-8	- Email server not found.
-10	- Email transfer error.
-13	- SMTP parameters are not set or are illegal.
-14	- The server certificate failed to validate.
-15	- The secure connection failed.

Declaration:

```
FUNCTION smtpSend : INT;
VAR_INPUT
   Receiver    : STRING;
   Receiver_cc : STRING;
   Subject     : STRING;
   Message     : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   alarm : BOOL;
END_VAR;

VAR
   fired : BOOL;
END_VAR;

PROGRAM example;
VAR
   rc      : INT;
   Subject : STRING;
END_VAR;

   gsmPower(power := ON);
   netOpen(iface:=1);
   smtpOpen();

   // Wait for network connection (after this, we are connected to the
Internet)
   WHILE NOT netConnected(iface:=1) DO
      DebugMsg(message:="Waiting for network connection");
      Sleep(delay:=2500);
   END_WHILE;

BEGIN
   // Is there an alarm
   IF alarm AND NOT fired THEN
      Subject := strFormat(format := "Device \4", v4 := boardSerialNumber());
      rc := smtpSend(Receiver := "devices@alarms.com",
                     Receiver_cc := "cc_devices@alarms.com",
                     Subject := Subject,
                     Message := "Alarm is triggered!");
      IF rc <> 0 THEN
         DebugFmt(message := "Error sending mail (errno=\1)", v1 := rc);
      ELSE
         DebugMsg(message := "Mail sent");
      END_IF;
   END_IF;
   fired := alarm;
END;
END_PROGRAM;
```

4.2.41.7 smtpNew (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.50 / 1.00.00

This function will create a new email.
After the email is created, use smtpAddText to add text to the message and smtpAddAttachment to attach files.
An email created by smtpNew can only be sent with the smtpSendX function.

Note: the outbox can only contain two emails.

Input:

Receiver : STRING
The email address of the receiver (max. 40 characters).

Receiver_cc : STRING
The email address of an optional receiver (max. 40 characters).

subject : STRING
The subject of the email (max. 40 characters).

Returns: INT

Handle to the new email.
-1 - General error.
-2 - SMTP interface is not open.
-3 - Invalid parameter(s).
-4 - Outbox is full.

Declaration:

```
FUNCTION smtpNew : INT;
VAR_INPUT
   Receiver    : STRING;
   Receiver_cc : STRING;
   Subject     : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   send  : DINT;
   timer : TON;
   gps   : gpsFix;
END_VAR;

FUNCTION gps_log;
VAR
   fd    : FILE;
   str   : STRING;
END_VAR;
   IF fsFileExists(name := "smtptest.txt") THEN
      fd := fsFileOpen(name := "smtptest.txt");
   ELSE
      fd := fsFileCreate(name := "smtptest.txt");
   END_IF;
   IF fsFileStatus(fd := fd) = 0 THEN
      gps();
      str := strFormat(format := "\1,", v1 := gps.mode);
      IF gps.mode = 0 THEN
         str := str + ",,,,,,,";
      ELSE
         str := str + strFormat(format := "\1.\2.\3,", v1 := gps.year, v2 :=
gps.month, v3 := gps.day);
         str := str + strFormat(format := "\1:\2:\3,", v1 := gps.hour, v2 :=
gps.minute, v3 := gps.second);
         IF gps.mode = 1 THEN
            str := str + ",,,,,";
         ELSE
            str := str + strFormat(format := "\4,", v4 := gps.latitude);
            str := str + strFormat(format := "\4,", v4 := gps.longitude);
            str := str + strFormat(format := "\4,", v4 := gps.speed);
            str := str + strFormat(format := "\4,", v4 := gps.course);
            str := str + strFormat(format := "\1,", v1 := gps.height);
```

```
         END_IF;
      END_IF;
      str := str + strFormat(format := "\1,", v1 := gps.inview);
      str := str + strFormat(format := "\1$n", v1 := gps.used);

      fsFileWriteString(fd := fd, str := str);
      fsFileClose(fd := fd);
   END_IF;
END_FUNCTION;

PROGRAM example;
VAR
   md    : INT;
   rc    : INT;
END_VAR;

   gsmPower(power := ON);
   gpsPower(power := ON);
   netOpen(iface:=1);
   fsMediaOpen(media := 0);
   smtpOpen();

   timer.pt := 10000;
   send     := clockNow() + 86400;
   DebugMsg(message := "Application running");

BEGIN
   timer(trig := ON);
   IF timer.q THEN
      gps_log();
      timer(trig := OFF);
   END_IF;

   IF clockNow() > send THEN
      // Build mail to send
      md := smtpNew(Receiver := "device@tracking.com", Subject :=
strFormat(format := "\4 trace", v4 := boardSerialNumber()));
      smtpAddText(Handle := md, Message := strFormat(format := "Device: \1$n",
v1 := boardType()));
      smtpAddText(Handle := md, Message := strFormat(format := "Fw:   \1.
\2$n", v1 := boardVersion() / 100, v2 := boardVersion() MOD 100));
      smtpAddText(Handle := md, Message := strFormat(format := "App:  " +
verGetAppName() + "$n"));
      smtpAddText(Handle := md, Message := strFormat(format := "Ver:  \1.
\2$n", v1 := verGetAppVersion() / 100, v2 := verGetAppVersion() MOD 100));
      smtpAddAttachment(Handle := md, Filename := "smtptest.txt");
      // Send mail
      rc := smtpSendX(Handle := md);
      IF rc = 0 THEN
         // Wait until mail is sent
         rc := smtpAwaitCompletion(Handle := md);
         IF rc = 0 THEN
            fsFileDelete(name := "smtptest.txt");
            DebugMsg(message := "Mail sent");
            send := clockNow() + 86400;
         ELSE
            DebugFmt(message := "Failed to send mail! (errno=\1)", v1 := rc);
            send := clockNow() + 300;
         END_IF;
      ELSE
         DebugFmt(message := "Failed to send mail! (errno=\1)", v1 := rc);
         send := clockNow() + 300;
      END_IF;
   END_IF;
```

```
END;
END_PROGRAM;
```

4.2.41.8 smtpAddText (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.50 / 1.00.00

This function will add text to the email message. An email must be created by smtpNew before text can be added.
Once the text is added, it cannot be removed again, and the only option is to cancel the email (with smtpCancel) and start over again.

Note: the email message can only contain 4095 characters of text.

Input:

Handle : INT
The handle for the email.

Message : STRING
The text message to add.

Returns: INT

The number of free characters remaining.
-1 - General error.
-2 - SMTP interface is not open.
-3 - Invalid parameter(s).
-5 - Invalid mail handle.
-6 - The mail is already sent.
-11 - There is no room for the text in the email.

Declaration:

```
FUNCTION smtpAddText : INT;
VAR_INPUT
   Handle  : INT;
   Message : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   send  : DINT;
   timer : TON;
   gps   : gpsFix;
END_VAR;

FUNCTION gps_log;
VAR
   fd    : FILE;
   str   : STRING;
END_VAR;
   IF fsFileExists(name := "smtptest.txt") THEN
      fd := fsFileOpen(name := "smtptest.txt");
   ELSE
      fd := fsFileCreate(name := "smtptest.txt");
   END_IF;
   IF fsFileStatus(fd := fd) = 0 THEN
```

```
      gps();
      str := strFormat(format := "\1,", v1 := gps.mode);
      IF gps.mode = 0 THEN
         str := str + ",,,,,,,";
      ELSE
         str := str + strFormat(format := "\1.\2.\3,", v1 := gps.year, v2 :=
gps.month, v3 := gps.day);
         str := str + strFormat(format := "\1:\2:\3,", v1 := gps.hour, v2 :=
gps.minute, v3 := gps.second);
         IF gps.mode = 1 THEN
            str := str + ",,,,,";
         ELSE
            str := str + strFormat(format := "\4,", v4 := gps.latitude);
            str := str + strFormat(format := "\4,", v4 := gps.longitude);
            str := str + strFormat(format := "\4,", v4 := gps.speed);
            str := str + strFormat(format := "\4,", v4 := gps.course);
            str := str + strFormat(format := "\1,", v1 := gps.height);
         END_IF;
      END_IF;
      str := str + strFormat(format := "\1,", v1 := gps.inview);
      str := str + strFormat(format := "\1$n", v1 := gps.used);

      fsFileWriteString(fd := fd, str := str);
      fsFileClose(fd := fd);
   END_IF;
END_FUNCTION;

PROGRAM example;
VAR
   md    : INT;
   rc    : INT;
END_VAR;

   gsmPower(power := ON);
   gpsPower(power := ON);
   netOpen(iface := 1);
   fsMediaOpen(media := 0);
   smtpOpen();

   timer.pt := 10000;
   send     := clockNow() + 86400;
   DebugMsg(message := "Application running");

BEGIN
   timer(trig := ON);
   IF timer.q THEN
      gps_log();
      timer(trig := OFF);
   END_IF;

   IF clockNow() > send THEN
      // Build mail to send
      md := smtpNew(Receiver := "device@tracking.com", Subject :=
strFormat(format := "\4 trace", v4 := boardSerialNumber()));
      smtpAddText(Handle := md, Message := strFormat(format := "Device: \1$n",
v1 := boardType()));
      smtpAddText(Handle := md, Message := strFormat(format := "Fw:   \1.
\2$n", v1 := boardVersion() / 100, v2 := boardVersion() MOD 100));
      smtpAddText(Handle := md, Message := strFormat(format := "App:  " +
verGetAppName() + "$n"));
      smtpAddText(Handle := md, Message := strFormat(format := "Ver:  \1.
\2$n", v1 := verGetAppVersion() / 100, v2 := verGetAppVersion() MOD 100));
      smtpAddAttachment(Handle := md, Filename := "smtptest.txt");
      // Send mail
```

```
      rc := smtpSendX(Handle := md);
      IF rc = 0 THEN
         // Wait until mail is sent
         rc := smtpAwaitCompletion(Handle := md);
         IF rc = 0 THEN
            fsFileDelete(name := "smtptest.txt");
            DebugMsg(message := "Mail sent");
            send := clockNow() + 86400;
         ELSE
            DebugFmt(message := "Failed to send mail! (errno=\1)", v1 := rc);
            send := clockNow() + 300;
         END_IF;
      ELSE
         DebugFmt(message := "Failed to send mail! (errno=\1)", v1 := rc);
         send := clockNow() + 300;
      END_IF;
   END_IF;
END;
END_PROGRAM;
```

4.2.41.9 smtpAddAttachment (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.50 / 1.00.00

This function will attach a file to the email. An email must be created by smtpNew before files can be attached.
Once the file is attached, it cannot be removed again, and the only option is to cancel the email (with smtpCancel) and start over again.

Note: only one file can be attached to the email.

Input:

Handle : INT
The handle for the email.

Filename : STRING
The name and path of the file to add.

Returns: INT

0	- Success.
-1	- General error.
-2	- SMTP interface is not open.
-3	- Invalid parameter(s).
-5	- Invalid email handle.
-6	- The email is already sent.
-11	- There is no room for more files in the email.
-12	- File not found.

Declaration:

```
FUNCTION smtpAddAttachment : INT;
VAR_INPUT
   Handle   : INT;
   Filename : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   send  : DINT;
   timer : TON;
   gps   : gpsFix;
END_VAR;

FUNCTION gps_log;
VAR
   fd    : FILE;
   str   : STRING;
END_VAR;
   IF fsFileExists(name := "smtptest.txt") THEN
      fd := fsFileOpen(name := "smtptest.txt");
   ELSE
      fd := fsFileCreate(name := "smtptest.txt");
   END_IF;
   IF fsFileStatus(fd := fd) = 0 THEN
      gps();
      str := strFormat(format := "\1,", v1 := gps.mode);
      IF gps.mode = 0 THEN
         str := str + ",,,,,,,";
      ELSE
         str := str + strFormat(format := "\1.\2.\3,", v1 := gps.year, v2 :=
gps.month, v3 := gps.day);
         str := str + strFormat(format := "\1:\2:\3,", v1 := gps.hour, v2 :=
gps.minute, v3 := gps.second);
         IF gps.mode = 1 THEN
            str := str + ",,,,,";
         ELSE
            str := str + strFormat(format := "\4,", v4 := gps.latitude);
            str := str + strFormat(format := "\4,", v4 := gps.longitude);
            str := str + strFormat(format := "\4,", v4 := gps.speed);
            str := str + strFormat(format := "\4,", v4 := gps.course);
            str := str + strFormat(format := "\1,", v1 := gps.height);
         END_IF;
      END_IF;
      str := str + strFormat(format := "\1,", v1 := gps.inview);
      str := str + strFormat(format := "\1$n", v1 := gps.used);

      fsFileWriteString(fd := fd, str := str);
      fsFileClose(fd := fd);
   END_IF;
END_FUNCTION;

PROGRAM example;
VAR
   md    : INT;
   rc    : INT;
END_VAR;

   gsmPower(power := ON);
   gpsPower(power := ON);
   netOpen(iface := 1);
   fsMediaOpen(media := 0);
   smtpOpen();

   timer.pt := 10000;
   send     := clockNow() + 86400;
   DebugMsg(message := "Application running");
```

```
BEGIN
   timer(trig := ON);
   IF timer.q THEN
      gps_log();
      timer(trig := OFF);
   END_IF;

   IF clockNow() > send THEN
      // Build mail to send
      md := smtpNew(Receiver := "device@tracking.com", Subject :=
strFormat(format := "\4 trace", v4 := boardSerialNumber()));
      smtpAddText(Handle := md, Message := strFormat(format := "Device: \1$n",
v1 := boardType()));
      smtpAddText(Handle := md, Message := strFormat(format := "Fw:   \1.
\2$n", v1 := boardVersion() / 100, v2 := boardVersion() MOD 100));
      smtpAddText(Handle := md, Message := strFormat(format := "App:  " +
verGetAppName() + "$n"));
      smtpAddText(Handle := md, Message := strFormat(format := "Ver:  \1.
\2$n", v1 := verGetAppVersion() / 100, v2 := verGetAppVersion() MOD 100));
      smtpAddAttachment(Handle := md, Filename := "smtptest.txt");
      // Send mail
      rc := smtpSendX(Handle := md);
      IF rc = 0 THEN
         // Wait until mail is sent
         rc := smtpAwaitCompletion(Handle := md);
         IF rc = 0 THEN
            fsFileDelete(name := "smtptest.txt");
            DebugMsg(message := "Mail sent");
            send := clockNow() + 86400;
         ELSE
            DebugFmt(message := "Failed to send mail! (errno=\1)", v1 := rc);
            send := clockNow() + 300;
         END_IF;
      ELSE
         DebugFmt(message := "Failed to send mail! (errno=\1)", v1 := rc);
         send := clockNow() + 300;
      END_IF;
   END_IF;
END;
END_PROGRAM;
```

4.2.41.10 smtpSendX (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.50 / 1.00.00

This function will start sending the email. An email must be created by smtpNew before it can be sent.
The network interface must be connected and the SMTP interface must be configured to use an email server (with smtpSetConfig) before this function is called.
The function will return after starting to send the email while the transfer will continue asynchronously.

Once the transfer is started, use smtpStatus to get transfer progress and smtpAwaitCompletion to wait for the transfer to complete.

When the asynchronous transfer is completed, or if an error is encountered, the email is removed from the outbox and all waiting threads will be resumed.

Input:

Handle : INT
The handle for the email.

Returns: INT

0	- Success.
-1	- General error.
-2	- SMTP interface is not open.
-5	- Invalid mail handle.
-6	- The mail is already sent.
-7	- Network error.
-13	- SMTP parameters are not set or are illegal.

Declaration:

```
FUNCTION smtpSendX : INT;
VAR_INPUT
   Handle : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   send  : DINT;
   timer : TON;
   gps   : gpsFix;
END_VAR;

FUNCTION gps_log;
VAR
   fd    : FILE;
   str   : STRING;
END_VAR;
   IF fsFileExists(name := "smtptest.txt") THEN
      fd := fsFileOpen(name := "smtptest.txt");
   ELSE
      fd := fsFileCreate(name := "smtptest.txt");
   END_IF;
   IF fsFileStatus(fd := fd) = 0 THEN
      gps();
      str := strFormat(format := "\1,", v1 := gps.mode);
      IF gps.mode = 0 THEN
         str := str + ",,,,,,,";
      ELSE
         str := str + strFormat(format := "\1.\2.\3,", v1 := gps.year, v2 :=
gps.month, v3 := gps.day);
         str := str + strFormat(format := "\1:\2:\3,", v1 := gps.hour, v2 :=
gps.minute, v3 := gps.second);
         IF gps.mode = 1 THEN
            str := str + ",,,,,";
         ELSE
            str := str + strFormat(format := "\4,", v4 := gps.latitude);
            str := str + strFormat(format := "\4,", v4 := gps.longitude);
            str := str + strFormat(format := "\4,", v4 := gps.speed);
            str := str + strFormat(format := "\4,", v4 := gps.course);
            str := str + strFormat(format := "\1,", v1 := gps.height);
         END_IF;
      END_IF;
      str := str + strFormat(format := "\1,", v1 := gps.inview);
      str := str + strFormat(format := "\1$n", v1 := gps.used);
```

```
      fsFileWriteString(fd := fd, str := str);
      fsFileClose(fd := fd);
   END_IF;
END_FUNCTION;

PROGRAM example;
VAR
   md    : INT;
   rc    : INT;
END_VAR;

   gsmPower(power := ON);
   gpsPower(power := ON);
   netOpen(iface := 1);
   fsMediaOpen(media := 0);
   smtpOpen();

   timer.pt := 10000;
   send     := clockNow() + 86400;
   DebugMsg(message := "Application running");

BEGIN
   timer(trig := ON);
   IF timer.q THEN
      gps_log();
      timer(trig := OFF);
   END_IF;

   IF clockNow() > send THEN
      // Build mail to send
      md := smtpNew(Receiver := "device@tracking.com", Subject :=
strFormat(format := "\4 trace", v4 := boardSerialNumber()));
      smtpAddText(Handle := md, Message := strFormat(format := "Device: \1$n",
v1 := boardType()));
      smtpAddText(Handle := md, Message := strFormat(format := "Fw:   \1.
\2$n", v1 := boardVersion() / 100, v2 := boardVersion() MOD 100));
      smtpAddText(Handle := md, Message := strFormat(format := "App:  " +
verGetAppName() + "$n"));
      smtpAddText(Handle := md, Message := strFormat(format := "Ver:  \1.
\2$n", v1 := verGetAppVersion() / 100, v2 := verGetAppVersion() MOD 100));
      smtpAddAttachment(Handle := md, Filename := "smtptest.txt");
      // Send mail
      rc := smtpSendX(Handle := md);
      IF rc = 0 THEN
         // Wait until mail is sent
         rc := smtpAwaitCompletion(Handle := md);
         IF rc = 0 THEN
            fsFileDelete(name := "smtptest.txt");
            DebugMsg(message := "Mail sent");
            send := clockNow() + 86400;
         ELSE
            DebugFmt(message := "Failed to send mail! (errno=\1)", v1 := rc);
            send := clockNow() + 300;
         END_IF;
      ELSE
         DebugFmt(message := "Failed to send mail! (errno=\1)", v1 := rc);
         send := clockNow() + 300;
      END_IF;
   END_IF;
END;
END_PROGRAM;
```

4.2.41.11 smtpCancel (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.50 / 1.00.00

This function will stop a mail transfer and remove the email from the outbox. An email must be created by smtpNew before it can be canceled.
An email can be canceled at any time in its lifetime - from its creation to the completion of the email transfer.

When the email is canceled, all waiting threads (using smtpAwaitCompletion) will be resumed.

Input:

Handle : INT
The handle for the email.

Returns: INT

0 - Success.
-1 - General error.
-2 - SMTP interface is not open.
-5 - Invalid mail handle.

Declaration:

```
FUNCTION smtpCancel : INT;
VAR_INPUT
   Handle : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   md    : INT;
   rc    : INT;
END_VAR;

PROGRAM example;

   gsmPower(power := ON);
   netOpen(iface := 1);
   smtpOpen();

   DebugMsg(message := "Application running");

BEGIN
   ...
   md := smtpNew(Receiver := "your@email.address", Subject := "example");
   ...
   rc := smtpSendX(Handle := md);
   IF rc = 0 THEN
      ...
      smtpCancel(Handle := md);
      ...
   END_IF;
   ...
```

```
END;
END_PROGRAM;
```

4.2.41.12 smtpStatus (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.50 / 1.00.00

This function will return the status and progress of an email. An email must be created by smtpNew before the status can be requested.

Note: if there are attachments to the email, then the email message will count as 50% of the progress and the attachments as the last 50%.

Input:

Handle : INT
The handle for the email.

Returns: INT

Percentage of the mail transfer completed.
-1 - General error.
-2 - SMTP interface is not open.
-5 - Invalid email handle.
-8 - Email server not found.
-10 - Email transfer error.
-14 - The server certificate failed to validate.
-15 - The secure connection failed.

Declaration:

```
FUNCTION smtpStatus : INT;
VAR_INPUT
   Handle : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   md    : INT;
   rc    : INT;
END_VAR;

PROGRAM example;

   gsmPower(power := ON);
   netOpen(iface:=1);
   smtpOpen();

   DebugMsg(message := "Application running");

BEGIN
   ...
   md := smtpNew(Receiver := "your@email.address", Subject := "example");
   ...
   rc := smtpSendX(Handle := md);
   IF rc = 0 THEN
      ...
```

```
      rc := smtpStatus(Handle := md);
      IF rc >= 0 THEN
         DebugFmt(message := "Completed: \1%", v1 := rc);
      END_IF;
      ...
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.41.13 smtpAwaitCompletion (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.50 / 1.00.00

This function will block the thread while waiting for an email transfer to complete. An email must be created by smtpNew before this function can wait for completion.
It is possible to start waiting at any point in the lifetime of the email - from its creation to the completion of the email transfer.

When the email transfer is completed, and if an error is encountered during the transfer or the email is canceled (with smtpCancel), the email is removed from the outbox and all waiting threads will be resumed.

Input:

Handle : INT
The handle for the email.

Returns: INT

0	- Success.
-1	- General error.
-2	- SMTP interface is not open.
-5	- Invalid mail handle.
-8	- Mail server not found.
-9	- Mail transfer canceled.
-10	- Mail transfer error.

Declaration:

```
FUNCTION smtpAwaitCompletion : INT;
VAR_INPUT
   Handle : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   send  : DINT;
   timer : TON;
   gps   : gpsFix;
END_VAR;

FUNCTION gps_log;
VAR
   fd    : FILE;
   str   : STRING;
END_VAR;
```

```
   IF fsFileExists(name := "smtptest.txt") THEN
      fd := fsFileOpen(name := "smtptest.txt");
   ELSE
      fd := fsFileCreate(name := "smtptest.txt");
   END_IF;
   IF fsFileStatus(fd := fd) = 0 THEN
      gps();
      str := strFormat(format := "\1,", v1 := gps.mode);
      IF gps.mode = 0 THEN
         str := str + ",,,,,,,";
      ELSE
         str := str + strFormat(format := "\1.\2.\3,", v1 := gps.year, v2 :=
gps.month, v3 := gps.day);
         str := str + strFormat(format := "\1:\2:\3,", v1 := gps.hour, v2 :=
gps.minute, v3 := gps.second);
         IF gps.mode = 1 THEN
            str := str + ",,,,,";
         ELSE
            str := str + strFormat(format := "\4,", v4 := gps.latitude);
            str := str + strFormat(format := "\4,", v4 := gps.longitude);
            str := str + strFormat(format := "\4,", v4 := gps.speed);
            str := str + strFormat(format := "\4,", v4 := gps.course);
            str := str + strFormat(format := "\1,", v1 := gps.height);
         END_IF;
      END_IF;
      str := str + strFormat(format := "\1,", v1 := gps.inview);
      str := str + strFormat(format := "\1$n", v1 := gps.used);

      fsFileWriteString(fd := fd, str := str);
      fsFileClose(fd := fd);
   END_IF;
END_FUNCTION;

PROGRAM example;
VAR
   md    : INT;
   rc    : INT;
END_VAR;

   gsmPower(power := ON);
   gpsPower(power := ON);
   netOpen(iface := 1);
   fsMediaOpen(media := 0);
   smtpOpen();

   timer.pt := 10000;
   send     := clockNow() + 86400;
   DebugMsg(message := "Application running");

BEGIN
   timer(trig := ON);
   IF timer.q THEN
      gps_log();
      timer(trig := OFF);
   END_IF;

   IF clockNow() > send THEN
      // Build mail to send
      md := smtpNew(Receiver := "device@tracking.com", Subject :=
strFormat(format := "\4 trace", v4 := boardSerialNumber()));
      smtpAddText(Handle := md, Message := strFormat(format := "Device: \1$n",
v1 := boardType()));
      smtpAddText(Handle := md, Message := strFormat(format := "Fw:   \1.
\2$n", v1 := boardVersion() / 100, v2 := boardVersion() MOD 100));
```

```
      smtpAddText(Handle := md, Message := strFormat(format := "App:  " +
verGetAppName() + "$n"));
      smtpAddText(Handle := md, Message := strFormat(format := "Ver:  \1.
\2$n", v1 := verGetAppVersion() / 100, v2 := verGetAppVersion() MOD 100));
      smtpAddAttachment(Handle := md, Filename := "smtptest.txt");
      // Send mail
      rc := smtpSendX(Handle := md);
      IF rc = 0 THEN
         // Wait until mail is sent
         rc := smtpAwaitCompletion(Handle := md);
         IF rc = 0 THEN
            fsFileDelete(name := "smtptest.txt");
            DebugMsg(message := "Mail sent");
            send := clockNow() + 86400;
         ELSE
            DebugFmt(message := "Failed to send mail! (errno=\1)", v1 := rc);
            send := clockNow() + 300;
         END_IF;
      ELSE
         DebugFmt(message := "Failed to send mail! (errno=\1)", v1 := rc);
         send := clockNow() + 300;
      END_IF;
   END_IF;
END;
END_PROGRAM;
```

4.2.42 snmp: Simple Network Management Protocol

4.2.42.1 snmp: Simple Network Management Protocol

Simple Network Management Protocol (SNMP) is an Internet Standard protocol for collecting and organizing information about managed devices on IP networks, and for modifying that information to change device behavior.

The manager can read and, when permitted, write to variables of managed devices.
The managed devices can send asynchronous notifications, called traps, to managing applications. Traps can contain variables with information about the event.

With an SNMP capable RTCU device, the application of the managed device can publish internal variables through SNMP. These variables can be read and possibly written to by the managing application.

The various variable types supported can be deduced from the offered variable read/write functions below.

All variables and traps are represented by an Object Identifier (OID).
Here the OID's must be used in a numerical dotted notation. For example: "1.3.6.1.2.1.1.5".
We only support OIDs in this notation, no OID translation is supported.

SNMP version 1, 2c and 3 are supported.

SNMP v3 security:

Both the USM (User-based Security Model) and TSM (Transport Security Model) methods are supported.

The USM method uses Users for identification and privacy.
A user is identified by username, and contains all the necessary parameters. (see the snmpUser structure)

The TSM method uses TLS/DTLS for identification and privacy.
The TLS/DTLS are based on certificates for security. (see certificates for more information)
Important: The certificates must not be password protected.

When working with traps, each managed device must be identified; this is done with two lists, a list of users for managed devices using the USM method, and a list of certificates for managed devices using the TSM method.
Both of the lists are persistent and will not be lost when the device is reset.
The lists are managed by the cert and user functions listed below.

The SNMP functionality offered here, can be divided into eight sections as:

Session handling
The following functions are used to setup a session to connect to a client.

- snmpConnect — Connect to a SNMP service.
- snmpDisonnect — Disconnect from a SNMP service.

Reading from clients
The following functions are used to read variables from a connected client:

- snmpGetDouble — Read a double floating point value.

- snmpGetFloat — Read a floating point value.
- snmpGetInteger — Read an integer value.
- snmpGetIP — Read an IP address value.
- snmpGetOID — Read an OID value.
- snmpGetString — Read a string value.
- snmpGetTimeticks — Read a timetick value.

Writing to clients

The following functions are used to write variables to a connected client:

- snmpSetDouble — Write a double floating point value.
- snmpSetFloat — Write a floating point value.
- snmpSetInteger — Write an integer value.
- snmpSetIP — Write an IP address value.
- snmpSetOID — Write an OID value.
- snmpSetString — Write a string value.
- snmpSetTimeticks — Write a timetick value.

Publishing variables

The following function are used for the application to publish internal variables as SNMP variables.

- snmpStartAgent — Start the publishing agent.
- snmpStopAgent — Stop the publishing agent.
- snmpPublishDouble — Publish a double-precision floating-point type variable.
- snmpPublishFloat — Publish a single-precision floating-point type variable.
- snmpPublishInteger — Publish an integer type variable.
- snmpPublishIP — Publish an IP-address type variable.
- snmpPublishOID — Publish an OID type variable.
- snmpPublishString — Publish a string type variable.
- snmpPublishTimeticks — Publish a timeticks type variable.
- snmpUnpublish — Remove published variables from the register.

Receiving traps

The following functions and structures are used to manage trap reception and handling.

- snmpStartListen — Start listening for traps on an IP port.
- snmpStopListen — Stop listening for traps on all IP ports.
- snmpRegisterTrap — Register a trap to the event handler.
- snmpUnregisterTrap — Unregister a trap from the event handler.

Sending traps

The following function are used to send traps to a managing application.

- snmpCreateTrap — Establish the basic trap.
- snmpDestroyTrap — Free resources from an establish trap.
- snmpAddTrapVariable — Add variables to an established trap.
- snmpSendTrap — Send a trap to a connected manager application.

Security handling

The following functions and structures are used to manage security for SNMP version 3.

- snmpSecurityConfig — Configure TSM security.
- snmpCertGet — Get the name of a certificate.
- snmpCertSet — Set the name of a certificate.
- snmpUser — Structure with USM user information.
- snmpUserGet — Get a USM user profile for the trap handler.
- snmpUserSet — Set a USM user profile for the trap handler.

- snmpAgentUsersGet — Get a USM user profile for the publishing agent.
- snmpAgentUsersSet — Set a USM user profile for the publishing agent.

Event handling
The following functions are used to handle SNMP events on manager and client.

- snmpWaitEvent — Wait for incoming trap events or optional timeout.
- snmpGetNextVar — Get the next variable of a received SNMP event.
- snmpVariable — Structure to receive SNMP variables.

Limitations:

- 10 simultaneous connections.
- It is only possible to have an RTCU device run as either a manager or a client (publishing agent).
- It is possible to listen on 4 IP ports simultaneously.
- There is a limit of 50 individual OIDs that can be published at a time.
- 30 trap OIDs to listen for can be registered at a time.
- The maximum size of a trap is 10 kB.
- 10 traps for sending can be established at a time.
- Each trap for sending can have at most 10 variables.
- The NX32L compilation mode is required. (See Project Settings in the RTCU IDE)

4.2.42.2 Session handling

4.2.42.2.1 snmpConnect (Function)

```
Architecture:       NX32L
Device support:     All NX devices
Firmware version:   1.50.00
```

This function will open a connection to a SNMP device.

Input:

Host : STRING
The address of the device to connect to.

Port : DINT [1..65535] (default 161)
The port of the device to connect to.

TCP : BOOL (default FALSE)
Set to TRUE if a TCP connection is required, otherwise an UDP connection is used.
The protocol uses UDP connections as default.

Version : INT (default _SNMP_VER_2C**)**
The SNMP version to use.
_SNMP_VER_ SNMP version 1. Using community names for security.
1
_SNMP_VER_ SNMP version 2. Using community names for security.
2C
_SNMP_VER_ SNMP version 3. More advanced security (USM/TLS), including certificates.
3

Community : STRING (*Max 63 characters*)
The community string. Used in version 1 and 2c.

Security : SINT (default _SNMP_SEC_USM**)**
The type of security to use. Used in version 3.
_SNMP_SEC_- The USM security model. (username and password)
USM
_SNMP_SEC_- The TSM security model. (certificates)
TLS

Level : SINT (default _SNMP_SEC_ENC**)**
The security level used in communication (TSM security only, see overview).
_SNMP_SEC_- No authentication or encryption
NONE
_SNMP_SEC_- Authentication and no encryption
AUTH
_SNMP_SEC_- Authentication and encryption
ENC

USMUser : PTR
Pointer to a snmpUser structure. (USM security only)

EngineID : STRING (*10..64 characters or empty*)
The engine ID of the peer agent (TSM) or the USM users engineID when sending traps to the manager.

Certlocal : STRING
The name of the certificate to identify the local device. (TSM security only, see overview)

Certclient : STRING
The name of the certificate to identify the peer agent. (TSM security only, see overview)

Context : STRING (*Max 63 characters*)
The security context name to use if needed.

Output:

Handle : SYSHANDLE
The handle to the connection.

Returns: INT

1- Success.
0- This function is not supported.
-2- Invalid parameter.
-3- Too many client sessions. See limitations on overview page.
-12- General error.

Declaration

```
FUNCTION snmpConnect : INT;
VAR_INPUT
   Host           : MANDATORY STRING;
   Port           : DINT := 161;
   TCP            : BOOL := FALSE;
   Version        : INT := _SNMP_VER_2C;
   Community      : STRING;
   Security       : SINT := _SNMP_SEC_USM;
   Level          : SINT := _SNMP_SEC_ENC;
   UsmUser        : PTR;
   EngineID       : STRING;
   Certlocal      : STRING;
   Certclient     : STRING;
   Context        : STRING;
   Handle         : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   iface    : SINT := 2;
END_VAR;

PROGRAM test;
VAR
   rc       : INT;
   var_str  : STRING;
   snmplabs : SYSHANDLE;
END_VAR;
   // Open net interface.
   rc := netOpen(iface := iface);
   DebugFmt(Message := "netOpen (rc=\1)", v1 := rc);
   WHILE NOT netConnected(iface := iface) DO
      Sleep(Delay := 2000);
   END_WHILE;

   // Establish connection with client host.
   rc := snmpConnect(
                     handle    := snmplabs,
                     community := "public",
                     host      := "demo.snmplabs.com",
                     version   := _SNMP_VER_2C
                    );
   IF rc = 1 THEN
      // Read 'sysName'
      rc := snmpGetString(handle := snmplabs, oid := "1.3.6.1.2.1.1.5", v :=
var_str);
      IF rc = 1 THEN
         DebugFmt(message := "snmpGetString (rc=\1): returned : '" + var_str +
"'", v1 := rc);
      ELSE
         DebugFmt(message := "snmpGetString error. (rc=\1)", v1 := rc);
      END_IF;
   ELSE
      DebugFmt(message := "snmpConnect failed (rc=\1)", v1 := rc);
   END_IF;
   ...
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.42.2.2 snmpDisconnect (Function)

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.50.00

This function will close a connection opened by snmpConnect.

Output:

Handle : SYSHANDLE
The handle to the connection to be closed. The handle will be cleared.

Returns: INT

1- Success.

0- This function is not supported.
-1- Invalid handle.
-12- General error.

Declaration

```
FUNCTION snmpDisconnect : INT;
VAR_INPUT
   Handle    : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc          : INT;
   snmplabs    : SYSHANDLE;
END_VAR;
   ...
   rc := snmpDisconnect(handle := snmplabs);
   DebugFmt(message := "snmpDisconnect (rc=\1)", v1 := rc);
   ...
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.42.3 Client variable reading

4.2.42.3.1 snmpGetDouble (Function)

```
Architecture:       NX32L
Device support:     All NX devices
Firmware version:   1.50.00
```

This function will read a double floating point value from a managed device. See DOUBLE.

Input:

Handle : SYSHANDLE
The handle to the connection established from snmpConnect.

OID : STRING
The OID for the value to read.

Output:

V : DOUBLE
The value read.

Returns: INT

1- Success.
0- This function is not supported.
-1- Invalid handle.
-8- Communication error.
-9- Invalid variable type received.
-10- Invalid OID.

Declaration

```
FUNCTION snmpGetDouble : INT;
VAR_INPUT
   Handle  : MANDATORY SYSHANDLE;
   OID     : MANDATORY STRING;
   V       : ACCESS DOUBLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   iface    : SINT := 2;
END_VAR;

PROGRAM example;
VAR
   rc       : INT;
   snmplabs : SYSHANDLE;
   val      : DOUBLE;
END_VAR;
   // Open net interface.
   rc := netOpen(iface := iface);
   DebugFmt(Message := "netOpen (rc=\1)", v1 := rc);
   WHILE NOT netConnected(iface := iface) DO
      Sleep(Delay := 2000);
   END_WHILE;

   // Establish connection with client host.
   rc := snmpConnect(
                     handle    := snmplabs,
                     community := "public",
                     host      := "demo.snmplabs.com"
                    );
   IF rc = 1 THEN
      // Read double variable.
      rc := snmpGetDouble(
                          handle := snmplabs,
                          oid    := "1.3.6.1.2.1.880.1",
                          v      := val
                         );
      IF rc = 1 THEN
         DebugMsg(message := "snmpGetDouble returned : " + doubleToStr(v :=
val));
      ELSE
         DebugFmt(message := "snmpGetDouble (rc=\1)", v1 := rc);
      END_IF;
   ELSE
      DebugFmt(message := "snmpConnect failed (rc=\1)", v1 := rc);
   END_IF;

BEGIN
END;
END_PROGRAM;
```

4.2.42.3.2 snmpGetFloat (Function)

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.50.00

This function will read a floating point value from a managed device. See FLOAT.

Input:

Handle : SYSHANDLE
The handle to the connection established from snmpConnect.

OID : STRING
The OID for the value to read.

Output:

V : FLOAT
The value read.

Returns: INT

1- Success.
0- This function is not supported.
-1- Invalid handle.
-8- Communication error.
-9- Invalid variable type received.
-10- Invalid OID.

Declaration

```
FUNCTION snmpGetFloat : INT;
VAR_INPUT
   Handle  : MANDATORY SYSHANDLE;
   OID     : MANDATORY STRING;
   V       : ACCESS FLOAT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   iface    : SINT := 2;
END_VAR;

PROGRAM example;
VAR
   rc       : INT;
   snmplabs : SYSHANDLE;
   val      : FLOAT;
END_VAR;
   // Open net interface.
   rc := netOpen(iface := iface);
   DebugFmt(Message := "netOpen (rc=\1)", v1 := rc);
   WHILE NOT netConnected(iface := iface) DO
      Sleep(Delay := 2000);
   END_WHILE;

   // Establish connection with client host.
   rc := snmpConnect(
                     handle    := snmplabs,
                     community := "public",
                     host      := "demo.snmplabs.com"
                    );
   IF rc = 1 THEN
      // Read float variable.
      rc := snmpGetFloat(
                         handle := snmplabs,
                         oid    := "1.3.6.1.999.1",
```

```
                        v       := val
                       );
      IF rc = 1 THEN
         DebugMsg(message := "snmpGetFloat returned : " + floatToStr(v :=
val));
      ELSE
         DebugFmt(message := "snmpGetFloat (rc=\1)", v1 := rc);
      END_IF;
   ELSE
      DebugFmt(message := "snmpConnect failed (rc=\1)", v1 := rc);
   END_IF;

BEGIN
END;
END_PROGRAM;
```

4.2.42.3.3 snmpGetInteger (Function)

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.50.00

This function will read an integer value from a managed device.

Input:

Handle : SYSHANDLE
The handle to the connection established from snmpConnect.

OID : STRING
The OID for the value to read.

Output:

V : DINT
The value read.

Returns: INT

1- Success.
0- This function is not supported.
-1- Invalid handle.
-8- Communication error.
-9- Invalid variable type received.
-10- Invalid OID.

Declaration

```
FUNCTION snmpGetInteger : INT;
VAR_INPUT
   Handle  : MANDATORY SYSHANDLE;
   OID     : MANDATORY STRING;
   V       : ACCESS DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   iface    : SINT := 2;
END_VAR;

PROGRAM test;
```

```
VAR
   rc       : INT;
   snmplabs : SYSHANDLE;
   var_int  : DINT;
END_VAR;
   // Open net interface.
   rc := netOpen(iface := iface);
   DebugFmt(Message := "netOpen (rc=\1)", v1 := rc);
   WHILE NOT netConnected(iface := iface) DO
      Sleep(Delay := 2000);
   END_WHILE;

   // Establish connection with client host.
   rc := snmpConnect(
                     handle    := snmplabs,
                     community := "public",
                     host      := "demo.snmplabs.com"
                    );
   IF rc = 1 THEN
      // Read 'sysServices'
      rc := snmpGetInteger(
                           handle := snmplabs,
                           oid    := "1.3.6.1.2.1.1.7",
                           v      := var_int
                          );
      IF rc = 1 THEN
         DebugFmt(message := "snmpGetInteger returned : \4", v4 := var_int);
      ELSE
         DebugFmt(message := "snmpGetInteger (rc=\1)", v1 := rc);
      END_IF;
   ELSE
      DebugFmt(message := "snmpConnect failed (rc=\1)", v1 := rc);
   END_IF;
   ...
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.42.3.4 snmpGetIP (Function)

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.50.00

This function will read an IP address value from a managed device.

Input:

Handle : SYSHANDLE
The handle to the connection established from snmpConnect.

OID : STRING
The OID for the value to read.

Output:

V : STRING
The value read.

Returns: INT

1- Success.
0- This function is not supported.

-1- Invalid handle.
-8- Communication error.
-9- Invalid variable type received.
-10- Invalid OID.

Declaration

```
FUNCTION snmpGetIP : INT;
VAR_INPUT
   Handle  : MANDATORY SYSHANDLE;
   OID     : MANDATORY STRING;
   V       : ACCESS STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   iface    : SINT := 2;
END_VAR;

PROGRAM test;
VAR
   rc       : INT;
   snmplabs : SYSHANDLE;
   var_str  : STRING;
END_VAR;
   // Open net interface.
   rc := netOpen(iface := iface);
   DebugFmt(Message := "netOpen (rc=\1)", v1 := rc);
   WHILE NOT netConnected(iface := iface) DO
      Sleep(Delay := 2000);
   END_WHILE;

   // Establish connection with client host.
   rc := snmpConnect(
                     handle    := snmplabs,
                     community := "public",
                     host      := "demo.snmplabs.com"
                    );
   IF rc = 1 THEN
      // Read IP address
      rc := snmpGetIP(
                      handle := snmplabs,
                      oid    := "1.3.6.1.2.1.3.1.1.3.2.1.195.218.254.97",
                      v      := var_str
                     );
      IF rc = 1 THEN
         DebugMsg(message := "snmpGetIP : returned : '" + var_str + "'");
      ELSE
         DebugFmt(message := "snmpGetIP (rc=\1)", v1 := rc);
      END_IF;
   ELSE
      DebugFmt(message := "snmpConnect failed (rc=\1)", v1 := rc);
   END_IF;
   ...
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.42.3.5 snmpGetOID (Function)

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.50.00

This function will read an OID value from a managed device.

Input:

Handle : SYSHANDLE
The handle to the connection established from snmpConnect.

OID : STRING
The OID for the value to read.

Output:

V : STRING
The value read.

Returns: INT

1- Success.
0- This function is not supported.
-1- Invalid handle.
-8- Communication error.
-9- Invalid variable type received.
-10- Invalid OID.

Declaration

```
FUNCTION snmpGetOID : INT;
VAR_INPUT
   Handle  : MANDATORY SYSHANDLE;
   OID     : MANDATORY STRING;
   V       : ACCESS STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   iface    : SINT := 2;
END_VAR;

PROGRAM test;
VAR
   rc       : INT;
   snmplabs : SYSHANDLE;
   var_str  : STRING;
END_VAR;
   // Open net interface.
   rc := netOpen(iface := iface);
   DebugFmt(Message := "netOpen (rc=\1)", v1 := rc);
   WHILE NOT netConnected(iface := iface) DO
      Sleep(Delay := 2000);
   END_WHILE;

   // Establish connection with client host.
   rc := snmpConnect(
                    handle    := snmplabs,
```

```
                    community := "public",
                    host      := "demo.snmplabs.com"
                   );
   IF rc = 1 THEN
      // Read 'sysObjectID'
      rc := snmpGetOID(
                     handle := snmplabs,
                     oid    := "1.3.6.1.2.1.1.2",
                     v      := var_str
                    );
      IF rc = 1 THEN
         DebugMsg(message := "snmpGetOID returned : '" + var_str + "'");
      ELSE
         DebugFmt(message := "snmpGetOID (rc=\1)", v1 := rc);
      END_IF;
   ELSE
      DebugFmt(message := "snmpConnect failed (rc=\1)", v1 := rc);
   END_IF;
   ...
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.42.3.6 snmpGetString (Function)

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.50.00

This function will read a string value from a managed device.

Input:

Handle : SYSHANDLE
The handle to the connection established from snmpConnect.

OID : STRING
The OID for the value to read.

Output:

V : STRING
The value read.

Returns: INT

1- Success.
0- This function is not supported.
-1- Invalid handle.
-8- Communication error.
-9- Invalid variable type received.
-10- Invalid OID.

Declaration

```
FUNCTION snmpGetString : INT;
VAR_INPUT
   Handle  : MANDATORY SYSHANDLE;
   OID     : MANDATORY STRING;
   V       : ACCESS STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   iface    : SINT := 2;
END_VAR;

PROGRAM test;
VAR
   rc       : INT;
   snmplabs : SYSHANDLE;
   var_str  : STRING;
END_VAR;
   // Open net interface.
   rc := netOpen(iface := iface);
   DebugFmt(Message := "netOpen (rc=\1)", v1 := rc);
   WHILE NOT netConnected(iface := iface) DO
      Sleep(Delay := 2000);
   END_WHILE;

   // Establish connection with client host.
   rc := snmpConnect(
                     handle    := snmplabs,
                     community := "public",
                     host      := "demo.snmplabs.com"
                    );
   IF rc = 1 THEN
      // Read 'sysName'
      rc := snmpGetString(
                          handle := snmplabs,
                          oid    := "1.3.6.1.2.1.1.5",
                          v      := var_str
                         );
      IF rc = 1 THEN
         DebugMsg(message := "snmpGetString returned : '" + var_str + "'");
      ELSE
         DebugFmt(message := "snmpGetString (rc=\1)", v1 := rc);
      END_IF;
   ELSE
      DebugFmt(message := "snmpConnect failed (rc=\1)", v1 := rc);
   END_IF;
   ...
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.42.3.7 snmpGetTimeticks (Function)

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.50.00

This function will read a timetick value from a managed device.

Input:

Handle : SYSHANDLE
The handle to the connection established from snmpConnect.

OID : STRING

The OID for the value to read.

Output:

V : DINT
The value read.

Returns: INT

1- Success.
0- This function is not supported.
-1- Invalid handle.
-8- Communication error.
-9- Invalid variable type received.
-10- Invalid OID.

Declaration

```
FUNCTION snmpGetTimeticks : INT;
VAR_INPUT
   Handle  : MANDATORY SYSHANDLE;
   OID     : MANDATORY STRING;
   V       : ACCESS DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   iface    : SINT := 2;
END_VAR;

PROGRAM test;
VAR
   rc         : INT;
   snmplabs   : SYSHANDLE;
   timeticks  : DINT;
END_VAR;
   // Open net interface.
   rc := netOpen(iface := iface);
   DebugFmt(Message := "netOpen (rc=\1)", v1 := rc);
   WHILE NOT netConnected(iface := iface) DO
      Sleep(Delay := 2000);
   END_WHILE;

   // Establish connection with client host.
   rc := snmpConnect(
                     handle    := snmplabs,
                     community := "public",
                     host      := "demo.snmplabs.com"
                    );
   IF rc = 1 THEN
      // Read 'sysUpTime'
      rc := snmpGetTimeticks(
                             handle := snmplabs,
                             oid    := "1.3.6.1.2.1.1.3",
                             v      := timeticks
                            );
      IF rc = 1 THEN
         DebugFmt(message := "snmpGetTimeticks returned : \4", v4 :=
timeticks);
      ELSE
         DebugFmt(message := "snmpGetTimeticks error. (rc=\1)", v1 := rc);
      END_IF;
```

```
   ELSE
      DebugFmt(message := "snmpConnect failed (rc=\1)", v1 := rc);
   END_IF;
   ...
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.42.4 Client variable writing

4.2.42.4.1 snmpSetDouble (Function)

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.50.00

This function will write a double-precision floating point value to a managed device. See DOUBLE.

Input:

Handle : SYSHANDLE
The handle to the connection established from snmpConnect.

OID : STRING
The OID for the value to write.

V : DOUBLE
The value to write.

Returns: **INT**

1- Success.
0- This function is not supported.
-1- Invalid handle.
-2- Invalid parameter.
-8- Communication error.
-10- Invalid OID.

Declaration

```
FUNCTION snmpSetDouble : INT;
VAR_INPUT
   Handle  : MANDATORY SYSHANDLE;
   OID     : MANDATORY STRING;
   V       : MANDATORY DOUBLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   iface      : SINT := 2;
END_VAR;

PROGRAM test;
VAR
   rc         : INT;
   snmplabs   : SYSHANDLE;
   var_double : DOUBLE;
```

```
END_VAR;
   // Open net interface.
   rc := netOpen(iface := iface);
   DebugFmt(Message := "netOpen (rc=\1)", v1 := rc);
   WHILE NOT netConnected(iface := iface) DO
      Sleep(Delay := 2000);
   END_WHILE;

   // Establish connection with client host.
   rc := snmpConnect(
                     handle    := snmplabs,
                     community := "public",
                     host      := "demo.snmplabs.com"
                    );
   IF rc = 1 THEN
      // Write to double variable.
      rc := snmpSetDouble(
                          handle := snmplabs,
                          oid    := "1.3.6.1.2.1.880.1",
                          v      := var_double
                         );
      DebugFmt(message := "snmpSetDouble (rc=\1)", v1 := rc);
   ELSE
      DebugFmt(message := "snmpConnect failed (rc=\1)", v1 := rc);
   END_IF;
   ...
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.42.4.2 snmpSetFloat (Function)

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.50.00

This function will write a floating point value to a managed device. See FLOAT.

Input:

Handle : SYSHANDLE
The handle to the connection established from snmpConnect.

OID : STRING
The OID for the value to write.

V : FLOAT
The value to write.

Returns: INT

1- Success.
0- This function is not supported.
-1- Invalid handle.
-2- Invalid parameter.
-8- Communication error.
-10- Invalid OID.

Declaration

```
FUNCTION snmpSetFloat : INT;
VAR_INPUT
   Handle  : MANDATORY SYSHANDLE;
```

```
   OID      : MANDATORY STRING;
   V        : MANDATORY FLOAT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   iface     : SINT := 2;
END_VAR;

PROGRAM test;
VAR
   rc        : INT;
   snmplabs  : SYSHANDLE;
   var_float : FLOAT;
END_VAR;
   // Open net interface.
   rc := netOpen(iface := iface);
   DebugFmt(Message := "netOpen (rc=\1)", v1 := rc);
   WHILE NOT netConnected(iface := iface) DO
      Sleep(Delay := 2000);
   END_WHILE;

   // Establish connection with client host.
   rc := snmpConnect(
                     handle    := snmplabs,
                     community := "public",
                     host      := "demo.snmplabs.com"
                    );
   IF rc = 1 THEN
      // Write to float variable.
      rc := snmpSetFloat(
                         handle := snmplabs,
                         oid    := "1.3.6.1.999.1",
                         v      := var_float
                        );
      DebugFmt(message := "snmpSetFloat (rc=\1)", v1 := rc);
   ELSE
      DebugFmt(message := "snmpConnect failed (rc=\1)", v1 := rc);
   END_IF;
   ...
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.42.4.3 snmpSetInteger (Function)

Architecture: **NX32L**
Device support: **All NX devices**
Firmware version: **1.50.00**

This function will write an integer value to a managed device.

Input:

Handle : SYSHANDLE
The handle to the connection established from snmpConnect.

OID : STRING

The OID for the value to write.

V : DINT
The value to write.

Subtype : SINT [0..1] (default 0)
The type of integer to write.
0: Signed integer type
1: Unsigned integer type

Returns: INT

1- Success.
0- This function is not supported.
-1- Invalid handle.
-2- Invalid parameter.
-8- Communication error.
-10- Invalid OID.

Declaration

```
FUNCTION snmpSetInteger : INT;
VAR_INPUT
   Handle  : MANDATORY SYSHANDLE;
   OID     : MANDATORY STRING;
   V       : MANDATORY DINT;
   Subtype : SINT := 0;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   iface    : SINT := 2;
END_VAR;

PROGRAM test;
VAR
   rc       : INT;
   snmplabs : SYSHANDLE;
   var_int  : DINT;
END_VAR;
   // Open net interface.
   rc := netOpen(iface := iface);
   DebugFmt(Message := "netOpen (rc=\1)", v1 := rc);
   WHILE NOT netConnected(iface := iface) DO
      Sleep(Delay := 2000);
   END_WHILE;

   // Establish connection with client host.
   rc := snmpConnect(
                     handle    := snmplabs,
                     community := "public",
                     host      := "demo.snmplabs.com"
                    );
   IF rc = 1 THEN
      // Write to integer variable.
       rc := snmpSetInteger(
                            handle  := snmplabs,
                            oid     := "1.3.6.1.2.1.4.26.0",
                            v       := var_int,
                            subtype := 0
```

```
                              );
      DebugFmt(message := "snmpSetInteger (rc=\1)", v1 := rc);
   ELSE
      DebugFmt(message := "snmpConnect failed (rc=\1)", v1 := rc);
   END_IF;
   ...
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.42.4.4 snmpSetIP (Function)

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.50.00

This function will write an IP address value to a managed device.

Input:

Handle : SYSHANDLE
The handle to the connection established from snmpConnect.

OID : STRING
The OID for the value to write.

V : STRING
The value to write.

Returns: INT

1- Success.
0- This function is not supported.
-1- Invalid handle.
-2- Invalid parameter.
-8- Communication error.
-10- Invalid OID.

Declaration

```
FUNCTION snmpSetIP : INT;
VAR_INPUT
   Handle  : MANDATORY SYSHANDLE;
   OID     : MANDATORY STRING;
   V       : MANDATORY STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   iface    : SINT := 2;
END_VAR;

PROGRAM test;
VAR
   rc       : INT;
   snmplabs : SYSHANDLE;
   var_ip   : STRING;
END_VAR;
   // Open net interface.
```

```
   rc := netOpen(iface := iface);
   DebugFmt(Message := "netOpen (rc=\1)", v1 := rc);
   WHILE NOT netConnected(iface := iface) DO
      Sleep(Delay := 2000);
   END_WHILE;

   // Establish connection with client host.
   rc := snmpConnect(
                     handle    := snmplabs,
                     community := "public",
                     host      := "demo.snmplabs.com"
                    );
   IF rc = 1 THEN
      var_ip := "172.17.10.1";
      // Write to IP address variable.
      rc := snmpSetIP(
                      handle := snmplabs,
                      oid    := "1.3.6.1.2.1.3.1.1.3.4.1.192.168.0.1",
                      v      := var_ip
                     );
      DebugFmt(message := "snmpSetIP (rc=\1)", v1 := rc);
   ELSE
      DebugFmt(message := "snmpConnect failed (rc=\1)", v1 := rc);
   END_IF;
   ...
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.42.4.5 snmpSetOID (Function)

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.50.00

This function will write an OID value to a managed device.

Input:

Handle : SYSHANDLE
The handle to the connection established from snmpConnect.

OID : STRING
The OID for the value to write.

V : STRING
The value to write.

Returns: INT

1- Success.
0- This function is not supported.
-1- Invalid handle.
-2- Invalid parameter.
-8- Communication error.
-10- Invalid OID.

Declaration

```
FUNCTION snmpSetOID : INT;
VAR_INPUT
   Handle  : MANDATORY SYSHANDLE;
   OID     : MANDATORY STRING;
```

```
   V       : MANDATORY STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   iface     : SINT := 2;
END_VAR;

PROGRAM test;
VAR
   rc        : INT;
   snmplabs  : SYSHANDLE;
   var_oid   : STRING;
END_VAR;
   // Open net interface.
   rc := netOpen(iface := iface);
   DebugFmt(Message := "netOpen (rc=\1)", v1 := rc);
   WHILE NOT netConnected(iface := iface) DO
      Sleep(Delay := 2000);
   END_WHILE;

   // Establish connection with client host.
   rc := snmpConnect(
                     handle    := snmplabs,
                     community := "public",
                     host      := "demo.snmplabs.com"
                    );
   IF rc = 1 THEN
      var_oid := "0.1.9";
      // Write to OID variable.
      rc := snmpSetOID(
                       handle := snmplabs,
                       oid    := "1.3.6.1.2.1.4.21.1.13.10.20.41.0",
                       v      := var_oid
                      );
      DebugFmt(message := "snmpSetOID (rc=\1)", v1 := rc);
   ELSE
      DebugFmt(message := "snmpConnect failed (rc=\1)", v1 := rc);
   END_IF;
   ...
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.42.4.6 snmpSetString (Function)

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.50.00

This function will write a string value to a managed device.

Input:

Handle : SYSHANDLE
The handle to the connection established from snmpConnect.

OID : STRING

The OID for the value to write.

V : STRING
The value to write.

Subtype : SINT [0..2] (default 0)
The type of string to write.
- 0: Text string type.
- 1: Hex string type.
- 2: Decimal string type.

Returns: INT

- 1- Success.
- 0- This function is not supported.
- -1- Invalid handle.
- -2- Invalid parameter.
- -8- Communication error.
- -10- Invalid OID.

Declaration

```
FUNCTION snmpSetString : INT;
VAR_INPUT
   Handle  : MANDATORY SYSHANDLE;
   OID     : MANDATORY STRING;
   V       : MANDATORY DINT;
   Subtype : SINT := 0;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   iface      : SINT := 2;
END_VAR;

PROGRAM test;
VAR
   rc         : INT;
   snmplabs   : SYSHANDLE;
   var_string : STRING;
END_VAR;
   // Open net interface.
   rc := netOpen(iface := iface);
   DebugFmt(Message := "netOpen (rc=\1)", v1 := rc);
   WHILE NOT netConnected(iface := iface) DO
      Sleep(Delay := 2000);
   END_WHILE;

   // Establish connection with client host.
   rc := snmpConnect(
                     handle    := snmplabs,
                     community := "public",
                     host      := "demo.snmplabs.com"
                    );
   IF rc = 1 THEN
      // Write to string variable.
      rc := snmpSetString(
                          handle  := snmplabs,
                          oid     := "1.3.6.1.2.1.2.2.1.6.1",
                          v       := var_string,
```

```
                        subtype := 0
                       );
      DebugFmt(message := "snmpSetString (rc=\1)", v1 := rc);
   ELSE
      DebugFmt(message := "snmpConnect failed (rc=\1)", v1 := rc);
   END_IF;
   ...
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.42.4.7 snmpSetTimeticks (Function)

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.50.00

This function will write a timetick value to a managed device.

Input:

Handle : SYSHANDLE
The handle to the connection established from snmpConnect.

OID : STRING
The OID for the value to write.

V : DINT
The value to write.

Returns: INT

1- Success.
0- This function is not supported.
-1- Invalid handle.
-2- Invalid parameter.
-8- Communication error.
-10- Invalid OID.

Declaration

```
FUNCTION snmpSetTimeticks : INT;
VAR_INPUT
   Handle   : MANDATORY SYSHANDLE;
   OID      : STRING;
   V        : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   iface     : SINT := 2;
END_VAR;

PROGRAM test;
VAR
   rc        : INT;
   snmplabs  : SYSHANDLE;
   timeticks : DINT;
```

```
END_VAR;
   // Open net interface.
   rc := netOpen(iface := iface);
   DebugFmt(Message := "netOpen (rc=\1)", v1 := rc);
   WHILE NOT netConnected(iface := iface) DO
      Sleep(Delay := 2000);
   END_WHILE;

   // Establish connection with client host.
   rc := snmpConnect(
                     handle    := snmplabs,
                     community := "public",
                     host      := "demo.snmplabs.com"
                    );
   IF rc = 1 THEN
      // Write to timeticks variable.
      rc := snmpSetTimeticks(
                             handle := snmplabs,
                             oid    := "1.3.6.1.2.1.4.31.1.1.46.1",
                             v      := timeticks
                            );
      DebugFmt(message := "snmpSetTimeticks (rc=\1)", v1 := rc);
   ELSE
      DebugFmt(message := "snmpConnect failed (rc=\1)", v1 := rc);
   END_IF;
   ...
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.42.5 Publishing Variables

4.2.42.5.1 snmpStartAgent (Function)

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.60.00

Start the publishing agent needed to use the publishing functions: snmpPublishTimeticks, snmpPublishInteger, snmpPublishString, snmpPublishOID, snmpPublishIP, snmpPublishFloat and snmpPublishDouble.

For USM connections, the read and write capable users must be set with snmpAgentUsersSet before calling this function.
For TLS connections, the local certificate must be set with snmpSecurityConfig and certificates for remote devices with snmpCertSet before calling this function.

Input:

EngineID : STRING
The engineID of the master agent. If not set, the master agent will generate a random engineID. This is only used in SNMPv3. It is a good practice to set it, as connecting SNMPv3 agents will often need it.

Pub : STRING
The public community string for read-only authentication (disabled if secure is set to TRUE). This is only used in SNMPv1/2c.

Priv : STRING

The private community string for read and writable permissions (disabled if secure is set to TRUE).
This is only used in SNMPv1/2c.

Network1 : STRING (Default to "")
The primary network mask setting. Devices on this network have possible access to the SNMP facilities on the RTCU device.
The format of this string is in the notation : x1.x2.x3.x4/y, x1 in]0..255], x2,x3,x4 in [0..255] and y in [0..32]. F.ex.: '192.168.1.0/24'.
The default empty string results in INANY_ADDR=0.0.0.0/0 being used, thus allowing all incoming requests.

Network2 : STRING (Default to "")
The network mask of a second optional network to enable for possible SNMP access. Disabled if set to the empty string.

Network3 : STRING (Default to "")
The network mask of a third optional network to enable for possible SNMP access. Disabled if set to the empty string.

CACert : STRING
The certificate authority root certificate, used to sign the local and remote certificate. Needed for TLS connections.

UDP : DINT [1..65535] (default 161)
The UDP port to operate on. Disabled if set to 0.

TCP : DINT [1..65535] (default 0)
The TCP port to operate on. Disabled if set to 0.

DTLS : DINT [1..65535] (default 10161)
The DTLS port to operate on. Disabled if set to 0.

TLSTCP : DINT [1..65535] (default 0)
The TLSTCP port to operate on. Disabled if set to 0.

Secure : BOOL (default TRUE)
If TRUE, only allow secure connections, i.e. SNMPv3.

Output:
None.

Returns: INT
1- Success.
0- This function is not supported.
-2- Invalid input parameter.
-12- General error.
-18- Already running as a manager. See limitations on overview page.

Declaration
```
FUNCTION snmpStartAgent : INT;
VAR_INPUT
   engineid : STRING;
   pub      : STRING;
   priv     : STRING;
   network1 : STRING := "";
   network2 : STRING;
   network3 : STRING;
```

```
   cacert   : STRING;
   udp      : DINT := 161;
   tcp      : DINT := 0;
   dtls     : DINT := 10161;
   tlstcp   : DINT := 0;
   secure   : BOOL := TRUE;
END_VAR;
```

Example:

```
// Please see the example at Examples - SNMP Client.
```

4.2.42.5.2 snmpStopAgent (Function)

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.60.00

Stop the publishing agent and unpublish all published SNMP variables.

Input:
None.

Output:
None.

Returns: INT
1- Success.
0- This function is not supported.
-19- Publishing agent is not running.

Declaration
```
FUNCTION snmpStopAgent : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
...
    snmpStopAgent();
...
END
```

4.2.42.5.3 snmpPublishDouble (Function)

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.60.00

Publish a double precision floating point variable.
The published variable can be read from any authenticated SNMP device on an authorized network.
Use snmpGetDouble to read the variable from an RTCU device.

Input:

V : DOUBLE
The double value to publish.

OID : STRING
The OID representing the variable to publish.

Writable : BOOL (Default false)
Give permissions to write to this variable with snmpSetDouble.

Output:

None.

Returns: INT

1- Success.
0- This function is not supported.
-10- Invalid input OID.
-12- General error.
-14- OID was published with another type.
-15- Maximum amount of published OIDs reached. See limitations on overview page.

Declaration

```
FUNCTION snmpPublishDouble : INT;
VAR_INPUT
   v         : MANDATORY DOUBLE;
   oid       : MANDATORY STRING;
   writable  : BOOL := FALSE;
END_VAR;
```

Example:

```
// Please see the example at Examples - SNMP Client.
```

4.2.42.5.4 snmpPublishFloat (Function)

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.60.00

Publish a single precision floating point variable.
The published variable can be read from any authenticated SNMP device on an authorized network.
Use snmpGetFloat to read the variable from an RTCU device.

Input:

V : FLOAT
The float value to publish.

OID : STRING
The OID representing the variable to publish.

Writable : BOOL (Default false)
Give permissions to write to this variable with snmpSetFloat.

Output:

None.

Returns: INT

1- Success.
0- This function is not supported.
-10- Invalid input OID.
-12- General error.
-14- OID was published with another type.
-15- Maximum amount of published OIDs reached. See limitations on overview page.

Declaration

```
FUNCTION snmpPublishFloat : INT;
VAR_INPUT
   v        : MANDATORY FLOAT;
   oid      : MANDATORY STRING;
   writable : BOOL := FALSE;
END_VAR;
```

Example:

```
// Please see the example at Examples - SNMP Client.
```

4.2.42.5.5 snmpPublishInteger (Function)

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.60.00

Publish an integer type variable.
The published variable can be read from any authenticated SNMP device on an authorized network.
Use snmpGetInteger to read the variable from an RTCU device.

Input:

V : DINT
The value to publish.

OID : STRING
The OID representing the variable to publish.

Subtype : USINT [0..2]
The type of integer to publish.
0: Signed integer type
1: Unsigned integer type COUNTER32.
2: Unsigned integer type GAUGE32.

Writable : BOOL (Default false)
Give permissions to write to this variable with snmpSetInteger.

Output:

None.

Returns: INT

1- Success.
0- This function is not supported.
-10- Invalid input OID.
-12- General error.
-14- OID was published with another type.
-15- Maximum amount of published OIDs reached. See limitations on overview page.

Declaration

```
FUNCTION snmpPublishInteger : INT;
VAR_INPUT
   v         : MANDATORY DINT;
   oid       : MANDATORY STRING;
   subtype   : USINT := 0;
   writable  : BOOL := FALSE;
END_VAR;
```

Example:

```
// Please see the example at Examples - SNMP Client.
```

4.2.42.5.6 snmpPublishIP (Function)

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.60.00

Publish an IP address variable.
The published variable can be read from any authenticated SNMP device on an authorized network.
Use snmpGetIP to read the variable from an RTCU device.

Input:

V : STRING
The IP address as a (string) to publish.

OID : STRING
The OID representing the variable to publish.

Writable : BOOL (Default false)
Give permissions to write to this variable with snmpSetIP.

Output:

None.

Returns: INT

1- Success.
0- This function is not supported.
-10- Invalid input OID.
-12- General error.
-14- OID was published with another type.
-15- Maximum amount of published OIDs reached. See limitations on overview page.

Declaration

```
FUNCTION snmpPublishIP : INT;
VAR_INPUT
   v         : MANDATORY STRING;
   oid       : MANDATORY STRING;
   writable  : BOOL := FALSE;
END_VAR;
```

Example:

```
// Please see the example at Examples - SNMP Client.
```

4.2.42.5.7 snmpPublishOID (Function)

```
Architecture:       NX32L
Device support:     All NX devices
Firmware version:   1.60.00
```

Publish an OID variable.
The published variable can be read from any authenticated SNMP device on an authorized network.
Use snmpGetOID to read the variable from an RTCU device.

Input:

V : STRING
The OID value as a (string) to publish.

OID : STRING
The OID representing the variable to publish.

Writable : BOOL (Default false)
Give permissions to write to this variable with snmpSetOID.

Output:

None.

Returns: INT

1- Success.
0- This function is not supported.
-10- Invalid input OID.
-12- General error.
-14- OID was published with another type.
-15- Maximum amount of published OIDs reached. See limitations on overview page.

Declaration

```
FUNCTION snmpPublishOID : INT;
VAR_INPUT
   v        : MANDATORY STRING;
   oid      : MANDATORY STRING;
   writable : BOOL := FALSE;
END_VAR;
```

Example:

```
// Please see the example at Examples - SNMP Client.
```

4.2.42.5.8 snmpPublishString (Function)

```
Architecture:       NX32L
Device support:     All NX devices
Firmware version:   1.60.00
```

Publish a string variable.
The published variable can be read from any authenticated SNMP device on an authorized network.
Use snmpGetString to read the variable from an RTCU device.

Input:

V : STRING

The value (string) to publish.

OID : STRING
The OID representing the variable to publish.

Writable : BOOL (Default false)
Give permissions to write to this variable with snmpSetString.

Output:
None.

Returns: INT
1- Success.
0- This function is not supported.
-10- Invalid input OID.
-12- General error.
-14- OID was published with another type.
-15- Maximum amount of published OIDs reached. See limitations on overview page.

Declaration
```
FUNCTION snmpPublishString : INT;
VAR_INPUT
   v         : MANDATORY STRING;
   oid       : MANDATORY STRING;
   subtype   : USINT := 1;
   writable  : BOOL := FALSE;
END_VAR;
```

Example:
```
// Please see the example at Examples - SNMP Client.
```

4.2.42.5.9 snmpPublishTimeticks (Function)

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.60.00

Publish a time ticks variable. System uptime (sysUpTime) is a standard time ticks variable with OID: "1.3.6.1.2.1.1.3".
The published variable can be read from any authenticated SNMP device on an authorized network.
Use snmpGetTimeticks to read the variable from an RTCU device.

Input:
V : DINT [1..65535]
The value to publish.

OID : STRING
The OID representing the variable to publish.

Writable : BOOL (Default false)
Give permissions to write to this variable with snmpSetTimeticks.

Output:
None.

Returns: INT

1- Success.
0- This function is not supported.
-10- Invalid input OID.
-12- General error.
-14- OID was published with another type.
-15- Maximum amount of published OIDs reached. See limitations on overview page.

Declaration

```
FUNCTION snmpPublishTimeticks : INT;
VAR_INPUT
   v        : MANDATORY DINT;
   oid      : MANDATORY STRING;
   writable : BOOL := FALSE;
END_VAR;
```

Example:

```
// Please see the example at Examples - SNMP Client.
```

4.2.42.5.10 snmpUnpublish (Function)

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.60.00

Remove one or all published variables from the register.

Input:

OID : STRING
The OID representing the variable to unpublish.

All : BOOL (Default false)
Remove all published variables if set to true.

Output:

None.

Returns: INT

1- Success.
0- This function is not supported.
-5- OID is not published.
-10- Invalid input OID.

Declaration

```
FUNCTION snmpUnpublish : INT;
VAR_INPUT
   oid : STRING;
   all : BOOL := FALSE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
...
```

```
    snmpUnpublish(oid:="1.3.6.1.4.1.999.1.1");
...
END
```

4.2.42.6 Receiving Traps

4.2.42.6.1 snmpStartListen (Function)

```
Architecture:       NX32L
Device support:     All NX devices
Firmware version:   1.50.00
```

This function will start a listening operation for incoming traps on the specified port number.

To receive traps using SNMPv3, the security must be configured first.
When using the USM method, the list of users must be configured on all connecting devices (Using the snmpUserSet and snmpUserGet functions).
When using the TSM method, the list of certificates must be configured on all connecting devices (Using the snmpCertSet and snmpCertGet functions).
If a managed device cannot be identified by one of these lists, then traps will not be received.

Input:

Port : DINT [1..65535] (default 162)
The port to listen on.

TCP : BOOL (default FALSE)
Set to TRUE if a TCP connection is required, otherwise an UDP connection is used.
The protocol uses UDP connections as default.

Version : SINT (default _SNMP_VER_2C**)**
The SNMP version to use. (obsoleted from firmware version 1.52.00, as all versions are accepted)
_SNMP_VER_ SNMP version 1. Using community names for security.
1
_SNMP_VER_ SNMP version 2. Using community names for security.
2C
_SNMP_VER_ SNMP version 3. Advanced security, including certificates.
3

Community : STRING
The community string. Used in Version 1 and 2c.

Security : SINT (default _SNMP_SEC_USM**)**
The type of security to use. Used in version 3.
_SNMP_SEC_- The USM security model. (username and password)
USM
_SNMP_SEC_- The TSM security model. (certificates)
TLS

Output:

Handle : SYSHANDLE
The handle to the connection.

Returns: **INT**

1- Success.
0- This function is not supported.
-2- Invalid parameter.

-3- Too many server sessions. See limitations on overview page.
-6- The port is in use.
-8- Failed to open port.
-12- General error.
-17- Publishing service is running. See limitations on overview page.

Declaration

```
FUNCTION snmpStartListen : INT;
VAR_INPUT
   Port      : DINT := 162;
   TCP       : BOOL := FALSE;
   Version   : SINT := _SNMP_VER_2C;
   Community : STRING;
   Security  : SINT := _SNMP_SEC_USM;
   Handle    : ACCESS SYSHANDLE;
END_VAR;
```

Example:

Please see the example at Examples - SNMP Mananger

4.2.42.6.2 **snmpStopListen (Function)**

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.50.00

This function will stop a listening operation.

Input:

All : BOOL (default FALSE)
Set to TRUE when all listening operations are to be closed.

Output:

Handle : SYSHANDLE
The handle to the connection. The handle will be cleared.

Returns: INT

1- Success.
0- This function is not supported.
-1- Invalid handle.
-12- General error.
-20- Manager is not running.

Declaration

```
FUNCTION snmpStopListen : INT;
VAR_INPUT
   All      : BOOL := FALSE;
   Handle   : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc          : INT;
   default_udp : SYSHANDLE;
```

```
END_VAR;
   ...
   rc := snmpStopListen(handle := default_udp, all := TRUE);
   DebugFmt(message := "snmpStopListen (rc=\1)", v1 := rc);
   ...
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.42.6.3 snmpRegisterTrap (Function)

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.50.00

This function will register a trap. Only registered traps will generate a trap event in the snmpWaitEvent function.

Input:

OID : STRING
The OID of the trap.

Returns: INT

1- Success.
0- This function is not supported.
-3- Too many traps are registered. See limitations on overview page.
-4- The trap is already registered.
-10- Illegal OID
-20- Manager not running.

Declaration

```
FUNCTION snmpRegisterTrap : INT;
VAR_INPUT
   OID : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   iface    : SINT := 2;
END_VAR;

PROGRAM test;
VAR
   rc       : INT;
END_VAR;
   // Open net interface.
   rc := netOpen(iface := iface);
   DebugFmt(Message := "netOpen (rc=\1)", v1 := rc);
   WHILE NOT netConnected(iface := iface) DO
      Sleep(Delay := 2000);
   END_WHILE;
   ...
   rc := snmpStartListen(handle := handle, community := "public");
   DebugFmt(message := "snmpStartListen (rc=\1)", v1 := rc);
   // Listen for some trap OID on community net 'public'
   rc := snmpRegisterTrap(oid := "1.3.6.1.4.1.6101.1.8.8.2.6");
   DebugFmt(message := "snmpRegisterTrap (rc=\1)", v1 := rc);
```

```
   ...
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.42.6.4 snmpUnregisterTrap (Function)

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.50.00

This function will remove a trap registered with the snmpRegisterTrap function.
After a trap is removed, no new trap events will be added to the event buffer.

Input:

OID : STRING
The OID of the trap.

Returns: **INT**

1- Success.
0- This function is not supported.
-5- The trap is not found.
-10- Illegal OID.
-20- Manager not running.

Declaration

```
FUNCTION snmpUnregisterTrap : INT;
VAR_INPUT
   OID : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc       : INT;
END_VAR;
...
// Remove listener for some trap OID.
rc := snmpUnregisterTrap(oid := "1.3.6.1.4.1.6101.1.8.8.2.6");
DebugFmt(message := "snmpUnregisterTrap (rc=\1)", v1 := rc);
...
BEGIN
   ...
END;
END_PROGRAM;
```

4.2.42.7 Sending Traps

4.2.42.7.1 snmpCreateTrap (Function)

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.60.00

Establish the basic trap to which SNMP variables can be added with snmpAddTrapVariable. Send the trap with snmpSendTrap.

Note: SNMPv1 traps are only for SNMPv1 connections.

Input:

OID : STRING
The trap OID.

Uptime : DINT [0..65535]
The sysUpTime that comes with the trap.

Generic : DINT
SNMPv1 generic type trap specifier.

Specific : DINT
SNMPv1 specific type trap specifier.

Output:

Trap : SYSHANDLE
The handle to the created trap.

Returns: INT

1- Success.
0- This function is not supported.
-12- General error.
-16- Maximum amount of traps created. See limitations on overview page.

Declaration

```
FUNCTION snmpCreateTrap : INT;
VAR_INPUT
   trap     : ACCESS SYSHANDLE;
   oid      : MANDATORY STRING;
   uptime   : MANDATORY DINT;
   generic  : DINT := 0;
   specific : DINT := 0;
END_VAR;
```

Example:

```
// Please see the example at Examples - SNMP Client.
```

4.2.42.7.2 snmpDestroyTrap (Function)

```
Architecture:       NX32L
Device support:     All NX devices
Firmware version:   1.60.00
```

Free all resources involved with a trap. That is trap variables that have been added, and the SYSHANDLE is reset so that it can be used again.

Input:

None.

Output:

Trap : SYSHANDLE
The handle to the trap, created with snmpCreateTrap, to free.

Returns: INT

1- Success.
0- This function is not supported.
-5- Trap is not registered.
-12- General error.

Declaration

```
FUNCTION snmpDestroyTrap : INT;
VAR_INPUT
   trap     : ACCESS SYSHANDLE;
END_VAR;
```

Example:

```
// Please see the example at Examples - SNMP Client.
```

4.2.42.7.3 snmpAddTrapVariable (Function)

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.60.00

Add a SNMP variable to the variable list of a trap. When sending the trap, the list of trap variables is sent with the trap.

Input:

Trap : SYSHANDLE
The handle to the trap created with snmpCreateTrap.

Data : snmpVariable
Trap variable to add. See snmpVariable.

Returns: INT

1- Success.
0- This function is not supported.
-3- Cannot add more variables, limit reached. See limitations on overview page.
-5- Trap is not registered.
-9- Illegal input variable type.
-10- Invalid input OID.
-12- General error.

Declaration

```
FUNCTION snmpAddTrapVariable : INT;
VAR_INPUT
   trap     : MANDATORY SYSHANDLE;
   data     : ACCESS snmpVariable;
END_VAR;
```

Example:

```
// Please see the example at Examples - SNMP Client.
```

4.2.42.7.4 snmpSendTrap (Function)

```
Architecture:       NX32L
Device support:     All NX devices
Firmware version:   1.60.00
```

Send a trap with a list of accompanying SNMP variables to a SNMP manager indicating an event on the sending RTCU device.

Input:

Trap : SYSHANDLE
The handle to the trap created with snmpCreateTrap.

Connection : SYSHANDLE
The handle to the connection set with snmpConnect.

Output:

None.

Returns: INT

1- Success.
0- This function is not supported.
-1- Invalid connection handle.
-5- Trap is not registered.
-12- General error.

Declaration

```
FUNCTION snmpSendTrap : INT;
VAR_INPUT
   trap       : MANDATORY SYSHANDLE;
   connection : MANDATORY SYSHANDLE;
END_VAR;
```

Example:

```
// Please see the example at Examples - SNMP Client.
```

4.2.42.8 Security Handling

4.2.42.8.1 snmpCertGet (Function)

```
Architecture:       NX32L
Device support:     All NX devices
Firmware version:   1.52.00
```

This function will read out a certificate name from the list of certificates for the allowed peer devices, previously set by the snmpCertSet function.

Input:

Index : SINT (1..25)
The index of the certificate.

Output:

Cert : STRING
The name of the certificate. Empty if no certificate is present

Returns: INT

1- Success.
0- This function is not supported.
-2- Illegal index.

Declaration

```
FUNCTION snmpCertGet : INT;
VAR_INPUT
   index : MANDATORY SINT;
   cert  : ACCESS STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   iface    : SINT := 2;
END_VAR;

FUNCTION show_cert
VAR_INPUT
   index : SINT;
END_VAR;
VAR
   str    : STRING;
   name   : STRING;
   rc     : INT;
END_VAR;

   // Prefix
   str := strFormat(format := "Certificate \1: ", v1 := index);

   // Get certificate
   rc := snmpCertGet(index := index, cert := name);
   IF rc < 1 THEN
      DebugFmt(message := str + "snmpCertGet=\1", v1 := rc);
      RETURN;
   END_IF;

   // Show
   IF strLen(str := name) = 0 THEN
      DebugMsg(message := str + "<EMPTY>");
   ELSE
      DebugMsg(message := str + name);
   END_IF;

END_FUNCTION;

PROGRAM example;
VAR
   i        : SINT;
   rc       : INT;
   handle   : SYSHANDLE;
END_VAR;

   // Iterate certificates
   DebugMsg(message := "-------------------------------------------------");
   FOR i := 1 TO 10 DO
      show_cert(index := i);
   END_FOR;
```

```
    // Set certificate
    rc := snmpCertSet(index := 1, cert := "snmp_agent");
    IF rc < 1 THEN
       DebugFmt(message := "snmpCertSet=\1", v1 := rc);
    END_IF;

    // Configure
    rc := snmpSecurityConfig(
                             localcert := "snmp_manager",
                             engineid  := "8000000001020304"
                            );
    IF rc < 1 THEN
       DebugFmt(message := "snmpSecurityConfig=\1", v1 := rc);
    END_IF;

    // Open net interface.
    rc := netOpen(iface := iface);
    DebugFmt(Message := "netOpen (rc=\1)", v1 := rc);
    WHILE NOT netConnected(iface := iface) DO
       Sleep(Delay := 2000);
    END_WHILE;
    DebugMsg(Message := "Network connected");

    // Start to listen for traps
    rc := snmpStartListen(
                          handle    := handle,
                          port      := 10162,
                          community := "public",
                          security  := _SNMP_SEC_TLS
                         );
    IF rc < 1 THEN
       DebugFmt(message := "snmpStartListen=\1", v1 := rc);
    END_IF;
    rc := snmpRegisterTrap(oid := "1.3.6.1.4.1.6101.1.8.8.2.6");
    IF rc < 1 THEN
       DebugFmt(message := "snmpRegisterTrap=\1", v1 := rc);
    END_IF;
    DebugMsg(Message := "Ready");

BEGIN
END;
END_PROGRAM;
```

4.2.42.8.2 snmpCertSet (Function)

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.52.00

This function is used to insert or remove a certificate in the list of allowed peer devices. (see security in overview)
The list of certificates are persistent and will not be lost when the device is reset.

Input:

Index : SINT (1..25)
The index of the certificate.

Cert : STRING
The name of the certificate. Leave empty to remove a certificate.

Returns: INT

1- Success.
0- This function is not supported.
-2- Illegal parameter
-4- The certificate is already present.

Declaration

```
FUNCTION snmpCertSet : INT;
VAR_INPUT
   index : MANDATORY SINT;
   cert  : MANDATORY STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   iface    : SINT := 2;
END_VAR;

FUNCTION show_cert
VAR_INPUT
   index : SINT;
END_VAR;
VAR
   str    : STRING;
   name   : STRING;
   rc     : INT;
END_VAR;

   // Prefix
   str := strFormat(format := "Certificate \1: ", v1 := index);

   // Get certificate
   rc := snmpCertGet(index := index, cert := name);
   IF rc < 1 THEN
      DebugFmt(message := str + "snmpCertGet=\1", v1 := rc);
      RETURN;
   END_IF;

   // Show
   IF strLen(str := name) = 0 THEN
      DebugMsg(message := str + "<EMPTY>");
   ELSE
      DebugMsg(message := str + name);
   END_IF;

END_FUNCTION;

PROGRAM example;
VAR
   i        : SINT;
   rc       : INT;
   handle   : SYSHANDLE;
END_VAR;

   // Iterate certificates
   DebugMsg(message := "-------------------------------------------------");
   FOR i := 1 TO 10 DO
      show_cert(index := i);
   END_FOR;

   // Set certificate
```

```
   rc := snmpCertSet(index := 1, cert := "snmp_agent");
   IF rc < 1 THEN
      DebugFmt(message := "snmpCertSet=\1", v1 := rc);
   END_IF;

   // Configure
   rc := snmpSecurityConfig(
                            localcert := "snmp_manager",
                            engineid  := "8000000001020304"
                           );
   IF rc < 1 THEN
      DebugFmt(message := "snmpSecurityConfig=\1", v1 := rc);
   END_IF;

   // Open net interface.
   rc := netOpen(iface := iface);
   DebugFmt(Message := "netOpen (rc=\1)", v1 := rc);
   WHILE NOT netConnected(iface := iface) DO
      Sleep(Delay := 2000);
   END_WHILE;
   DebugMsg(Message := "Network connected");

   // Start to listen for traps
   rc := snmpStartListen(
                         handle    := handle,
                         port      := 10162,
                         community := "public",
                         security  := _SNMP_SEC_TLS
                        );
   IF rc < 1 THEN
      DebugFmt(message := "snmpStartListen=\1", v1 := rc);
   END_IF;
   rc := snmpRegisterTrap(oid := "1.3.6.1.4.1.6101.1.8.8.2.6");
   IF rc < 1 THEN
      DebugFmt(message := "snmpRegisterTrap=\1", v1 := rc);
   END_IF;
   DebugMsg(Message := "Ready");

BEGIN
END;
END_PROGRAM;
```

4.2.42.8.3 snmpSecurityConfig (Function)

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.52.00

This function is needed for configuring TSM security, both when listening for incoming SNMP traps (manager) and when publishing SNMP variables.
The TSM security also requires a certificate to identify a peer device. The snmpCertSet and snmpCertGet functions are used to manage this list of certificates.

Input:

Localcert : STRING
The name of the certificate to identify the RTCU device. (see overview)

EngineID : STRING (*10..64 characters or empty*)
The engine ID value as a Hex value (Manager only).

Level : STRING
The security level used in communication.
_SNMP_SEC_- No authentication or encryption
NONE
_SNMP_SEC_- Authentication and no encryption
AUTH
_SNMP_SEC_- Authentication and encryption
ENC

Returns: INT

1- Success.
0- This function is not supported.
-2- Illegal parameter

Declaration

```
FUNCTION snmpSecurityConfig : INT;
VAR_INPUT
   localcert : STRING;
   engineid  : STRING;
   level     : SINT := _SNMP_SEC_ENC;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   iface    : SINT := 2;
END_VAR;

FUNCTION show_cert
VAR_INPUT
   index : SINT;
END_VAR;
VAR
   str    : STRING;
   name   : STRING;
   rc     : INT;
END_VAR;

   // Prefix
   str := strFormat(format := "Certificate \1: ", v1 := index);

   // Get certificate
   rc := snmpCertGet(index := index, cert := name);
   IF rc < 1 THEN
      DebugFmt(message := str + "snmpCertGet=\1", v1 := rc);
      RETURN;
   END_IF;

   // Show
   IF strLen(str := name) = 0 THEN
      DebugMsg(message := str + "<EMPTY>");
   ELSE
      DebugMsg(message := str + name);
   END_IF;

END_FUNCTION;

PROGRAM example;
VAR
   i        : SINT;
```

```
   rc       : INT;
   handle   : SYSHANDLE;
END_VAR;

   // Iterate certificates
   DebugMsg(message := "-------------------------------------------------");
   FOR i := 1 TO 10 DO
      show_cert(index := i);
   END_FOR;

   // Set certificate
   rc := snmpCertSet(index := 1, cert := "snmp_agent");
   IF rc < 1 THEN
      DebugFmt(message := "snmpCertSet=\1", v1 := rc);
   END_IF;

   // Configure
   rc := snmpSecurityConfig(
                            localcert := "snmp_manager",
                            engineid  := "8000000001020304"
                           );
   IF rc < 1 THEN
      DebugFmt(message := "snmpSecurityConfig=\1", v1 := rc);
   END_IF;

   // Open net interface.
   rc := netOpen(iface := iface);
   DebugFmt(Message := "netOpen (rc=\1)", v1 := rc);
   WHILE NOT netConnected(iface := iface) DO
      Sleep(Delay := 2000);
   END_WHILE;
   DebugMsg(Message := "Network connected");

   // Start to listen for traps
   rc := snmpStartListen(
                         handle    := handle,
                         port      := 10162,
                         community := "public",
                         security  := _SNMP_SEC_TLS
                        );
   IF rc < 1 THEN
      DebugFmt(message := "snmpStartListen=\1", v1 := rc);
   END_IF;
   rc := snmpRegisterTrap(oid := "1.3.6.1.4.1.6101.1.8.8.2.6");
   IF rc < 1 THEN
      DebugFmt(message := "snmpRegisterTrap=\1", v1 := rc);
   END_IF;
   DebugMsg(Message := "Ready");

BEGIN
END;
END_PROGRAM;
```

4.2.42.8.4 snmpUser (Structure)

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.52.00

The snmpUser structure contains the information of a USM security user.

The structure is used with the snmpConnect, snmpUserGet, snmpUserSet, snmpAgentUsersGet and snmpAgentUsersSet functions.

Fields:

Username : STRING (*Max 63 characters*)
The name of the user

Authentication : SINT
The method used to authenticate the user.
_SNMP_USM_ - MD5
MD5
_SNMP_USM_ - SHA
SHA

Password : STRING (8..*63 characters*)
The password to authenticate the user.

Encryption : SINT
The method used to encrypt communication
_SNMP_USM_ - AES
AES
_SNMP_USM_ - DES
DES

Cryptkey : STRING (8..*63 characters*)
The text used to encrypt communication.

EngineID : STRING (*10..64 characters or empty*)
The engine ID value as a Hex value.

Level : SINT
The security level used in communication.
_SNMP_SEC_ - No authentication or encryption
NONE
_SNMP_SEC_ - Authentication and no encryption
AUTH
_SNMP_SEC_ - Authentication and encryption
ENC

Declaration:

```
STRUCT_BLOCK snmpUser;
   username       : STRING;
   authentication : SINT;
   password       : STRING;
   encryption     : SINT;
   cryptkey       : STRING;
   engineid       : STRING;
   level          : SINT;
END_STRUCT_BLOCK;
```

4.2.42.8.5 snmpUserGet (Function)

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.52.00

This function will read out a user from the list of USM users on the manager, previously set by the snmpUserSet function.
The list contains the USM user profiles of the users that are allowed to send traps to the manager.

Input:
Index : SINT (1..25)
The index of the user.

Output:
User : snmpUser
The information about the user. See snmpUser.

Returns: INT
1- Success.
0- This function is not supported.
-2- Illegal index.

Declaration
```
FUNCTION snmpUserGet : INT;
VAR_INPUT
   index : MANDATORY SINT;
   user  : ACCESS snmpUser;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

VAR
   user     : snmpUser;
   iface    : SINT := 2;
END_VAR;

FUNCTION show_user
VAR_INPUT
   index       : SINT;
END_VAR;
VAR
   str         : STRING;
   rc          : INT;
END_VAR;

   // Get user
   rc := snmpUserGet(index := index, user := user);
   IF rc < 1 THEN
      DebugFmt(message := "  snmpUserGet=\1", v1 := rc);
      RETURN;
   END_IF;

   // Show
   DebugMsg(message := "-------------------------------------------------");
   DebugFmt(message := "User \1:", v1 := index);
   IF strLen(str := user.username) = 0 THEN
      DebugMsg(message := "  <EMPTY>");
   ELSE
      DebugMsg(message := "  Name               = " + user.username);
      DebugMsg(message := "  Password           = " + user.password);
      DebugMsg(message := "  Encrypt key        = " + user.cryptkey);
      DebugMsg(message := "  Engine ID          = " + user.engineid);
      CASE user.level OF
         0: DebugMsg(message := "  Level                = None");
         1: DebugMsg(message := "  Level                = Authenticate");
         2: DebugMsg(message := "  Level                = Encrypt");
      END_CASE;
      IF user.authentication = _SNMP_USM_MD5 THEN
```

```
            DebugMsg(message := "  Authentication     = MD5");
         ELSE
            DebugMsg(message := "  Authentication     = SHA");
         END_IF;
         IF user.encryption = _SNMP_USM_AES THEN
            DebugMsg(message := "  Encryption         = AES");
         ELSE
            DebugMsg(message := "  Encryption         = DES");
         END_IF;
      END_IF;

END_FUNCTION;

PROGRAM example;
VAR
   i        : SINT;
   rc       : INT;
   handle   : SYSHANDLE;
END_VAR;

   // Iterate Users
   FOR i := 1 TO 10 DO
      show_user(index := i);
   END_FOR;

   // Build user
   user.username        := "Administrator";
   user.password        := "Pass phrase";
   user.cryptkey        := "Pass phrase";
   user.engineid        := "8000000001020304";
   user.level           := _SNMP_SEC_ENC;
   user.authentication  := _SNMP_USM_MD5;
   user.encryption      := _SNMP_USM_AES;

   // Set user
   rc := snmpUserSet(index := 1, user := user);
   IF rc < 1 THEN
      DebugFmt(message := "snmpUserSet=\1", v1 := rc);
   END_IF;

   // Open net interface.
   rc := netOpen(iface := iface);
   DebugFmt(Message := "netOpen (rc=\1)", v1 := rc);
   WHILE NOT netConnected(iface := iface) DO
      Sleep(Delay := 2000);
   END_WHILE;

   // Start to listen for traps
   rc := snmpStartListen(
                         handle    := handle,
                         community := "public"
                        );
   IF rc < 1 THEN
      DebugFmt(message := "snmpStartListen=\1", v1 := rc);
   END_IF;
   rc := snmpRegisterTrap(oid := "1.3.6.1.4.1.6101.1.8.8.2.6");
   IF rc < 1 THEN
      DebugFmt(message := "snmpRegisterTrap=\1", v1 := rc);
   END_IF;

BEGIN
END;
END_PROGRAM;
```

4.2.42.8.6 snmpUserSet (Function)

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.52.00

Insert or remove a USM user profile on the manager.
The list of users are persistent and will not be lost when the device is reset.

Input:

Index : SINT (1..25)
The index of the user.

User : snmpUser
The information about the user. See snmpUser.
Leave the username empty to remove a user.

Returns: INT

1- Success.
0- This function is not supported.
-2- Illegal parameter
-4- Another user with this name is already present.

Declaration

```
FUNCTION snmpUserSet : INT;
VAR_INPUT
   index : MANDATORY SINT;
   user  : ACCESS snmpUser;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   user     : snmpUser;
   iface    : SINT := 2;
END_VAR;

FUNCTION show_user
VAR_INPUT
   index       : SINT;
END_VAR;
VAR
   str         : STRING;
   rc          : INT;
END_VAR;

   // Get user
   rc := snmpUserGet(index := index, user := user);
   IF rc < 1 THEN
      DebugFmt(message := "  snmpUserGet=\1", v1 := rc);
      RETURN;
   END_IF;

   // Show
   DebugMsg(message := "-------------------------------------------------");
   DebugFmt(message := "User \1:", v1 := index);
   IF strLen(str := user.username) = 0 THEN
```

```
      DebugMsg(message := "  <EMPTY>");
   ELSE
      DebugMsg(message := "  Name               = " + user.username);
      DebugMsg(message := "  Password           = " + user.password);
      DebugMsg(message := "  Encrypt key        = " + user.cryptkey);
      DebugMsg(message := "  Engine ID          = " + user.engineid);
      CASE user.level OF
         0: DebugMsg(message := "  Level              = None");
         1: DebugMsg(message := "  Level              = Authenticate");
         2: DebugMsg(message := "  Level              = Encrypt");
      END_CASE;
      IF user.authentication = _SNMP_USM_MD5 THEN
         DebugMsg(message := "  Authentication     = MD5");
      ELSE
         DebugMsg(message := "  Authentication     = SHA");
      END_IF;
      IF user.encryption = _SNMP_USM_AES THEN
         DebugMsg(message := "  Encryption         = AES");
      ELSE
         DebugMsg(message := "  Encryption         = DES");
      END_IF;
   END_IF;

END_FUNCTION;

PROGRAM example;
VAR
   i        : SINT;
   rc       : INT;
   handle   : SYSHANDLE;
END_VAR;

   // Iterate Users
   FOR i := 1 TO 10 DO
      show_user(index := i);
   END_FOR;

   // Build user
   user.username        := "Administrator";
   user.password        := "Pass phrase";
   user.cryptkey        := "Pass phrase";
   user.engineid        := "8000000001020304";
   user.level           := _SNMP_SEC_ENC;
   user.authentication  := _SNMP_USM_MD5;
   user.encryption      := _SNMP_USM_AES;

   // Set user
   rc := snmpUserSet(index := 1, user := user);
   IF rc < 1 THEN
      DebugFmt(message := "snmpUserSet=\1", v1 := rc);
   END_IF;

   // Open net interface.
   rc := netOpen(iface := iface);
   DebugFmt(Message := "netOpen (rc=\1)", v1 := rc);
   WHILE NOT netConnected(iface := iface) DO
      Sleep(Delay := 2000);
   END_WHILE;

   // Start to listen for traps
   rc := snmpStartListen(
                         handle    := handle,
                         community := "public"
                        );
```

```
   IF rc < 1 THEN
      DebugFmt(message := "snmpStartListen=\1", v1 := rc);
   END_IF;
   rc := snmpRegisterTrap(oid := "1.3.6.1.4.1.6101.1.8.8.2.6");
   IF rc < 1 THEN
      DebugFmt(message := "snmpRegisterTrap=\1", v1 := rc);
   END_IF;

BEGIN
END;
END_PROGRAM;
```

4.2.42.8.7 snmpAgentUsersGet (Function)

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.60.00

This function will read out a user from the list of USM users for the publishing agent, previously set by the snmpUserSet function.
Foreign entities must be able to identify themselves with these users to be able to access published variables on an USM SNMPv3 enabled RTCU device.

Input:

None

Output:

Readonly : snmpUser
USM user with read permissions.

Writable : snmpUser
USM user with read and write permissions.

Returns: INT

1- Success.
0- This function is not supported.
-2- Invalid input parameter.
-12- General error.

Declaration

```
FUNCTION snmpAgentUsersGet : INT;
VAR_INPUT
   readonly : ACCESS snmpUser;
   writable : ACCESS snmpUser;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

FUNCTION authToStr : STRING;
VAR_INPUT
   auth : SINT;
END_VAR;
   CASE auth OF
      _SNMP_USM_MD5 : authToStr := "MD5";
      _SNMP_USM_SHA : authToStr := "SHA";
   ELSE
      authToStr := "Unknown???";
   END_CASE;
```

```
END_FUNCTION

FUNCTION encToStr : STRING;
VAR_INPUT
   enc : SINT;
END_VAR;
   CASE enc OF
      _SNMP_USM_AES : encToStr := "AES";
      _SNMP_USM_DES : encToStr := "DES";
   ELSE
      encToStr := "Unknown???";
   END_CASE;
END_FUNCTION

FUNCTION levelToStr : STRING;
VAR_INPUT
   level : SINT;
END_VAR;
   CASE level OF
      _SNMP_SEC_NONE : levelToStr := "None";
      _SNMP_SEC_AUTH : levelToStr := "AuthOnly";
      _SNMP_SEC_ENC  : levelToStr := "Auth+Enc";
   ELSE
      levelToStr := "Unknown???";
   END_CASE;
END_FUNCTION

PROGRAM test;
VAR
   rc     : INT;
   rouser : snmpUser;
   rwuser : snmpUser;
END_VAR
rc := snmpAgentUsersGet(readonly:=rouser, writable:=rwuser);
DebugFmt(message:="snmpAgentUsersGet (rc=\1)", v1:=rc);
DebugMsg(message:="----------------------------------------");
DebugMsg(message:="Read-Only user:");
DebugMsg(message:="-----------USM user profile-------------");
DebugMsg(message:="Username is  : "+rouser.username);
DebugMsg(message:="Auth is      : "+authToStr(auth:=rouser.authentication));
DebugMsg(message:="Auth pass is : "+rouser.password);
DebugMsg(message:="Enc is       : "+encToStr(enc:=rouser.encryption));
DebugMsg(message:="Cryptkey is  : "+rouser.cryptkey);
DebugMsg(message:="Sec level is : "+levelToStr(level:=rouser.level));
DebugMsg(message:="----------------------------------------");
DebugMsg(message:="");
DebugMsg(message:="Read-Write capable user:");
DebugMsg(message:="-----------USM user profile-------------");
DebugMsg(message:="Username is  : "+rwuser.username);
DebugMsg(message:="Auth is      : "+authToStr(auth:=rwuser.authentication));
DebugMsg(message:="Auth pass is : "+rwuser.password);
DebugMsg(message:="Enc is       : "+encToStr(enc:=rwuser.encryption));
DebugMsg(message:="Cryptkey is  : "+rwuser.cryptkey);
DebugMsg(message:="Sec level is : "+levelToStr(level:=rwuser.level));
DebugMsg(message:="----------------------------------------");
END_PROGRAM;
```

4.2.42.8.8 snmpAgentUsersSet (Function)

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.60.00

Write USM users to the persistently stored database for the publishing agent.
Foreign entities must be able to identify themselves with these users to be able to access published variables on an USM SNMPv3 enabled RTCU device.

Input:

Readonly : snmpUser
USM user with read permissions.

Writable : snmpUser
USM user with read and write permissions.

Output:

None

Returns: INT

1- Success.
0- This function is not supported.
-2- Invalid input parameter.
-12- General error.

Declaration

```
FUNCTION snmpAgentUsersSet : INT;
VAR_INPUT
   readonly : ACCESS snmpUser;
   writable : ACCESS snmpUser;
END_VAR;
```

Example:

```
// Please see the example at Examples - SNMP Client.
```

4.2.42.9 Event Handling

4.2.42.9.1 snmpWaitEvent (Function)

```
Architecture:       NX32L
Device support:     All NX devices
Firmware version:   1.50.00
```

This function will until a trap event is received or an optional timeout. Only traps registered with the snmpRegisterTrap function will generate an event.
When a SNMP event is received, the variables associated with the event can be read by calling snmpGetNextVar once for each variable.

Note: Calling snmpWaitEvent again will clear the variables from the previous event!

Input:

Timeout : DINT (default -1)
Time in milliseconds to wait for trap event before returning. (-1: wait forever.)

Output:

Event : USINT
The event type of the returned event.
_SNMP_EVENT_NO If the function returns with a failure, check the return code.
EVENT

_SNMP_EVENT_TIMWhen a teímeout occured according to the Timeout parameter.
EOUT
_SNMP_EVENT_TR A registered trap was received.
AP
_SNMP_EVENT_SE A published variable was changed with a snmpSet call.
T

OID : STRING
The OID of the received SNMP event.

Vars : INT
The number of variables received with the SNMP event.

Returns: INT

1- Event received.
0- This function is not supported.
-2- Invalid parameter.
-3- Events have been lost due to too many events in queue.
-7- Timeout.
-12- General error.

Declaration

```
FUNCTION snmpWaitEvent : INT;
VAR_INPUT
   Timeout : DINT := -1;
   Event   : ACCESS USINT;
   OID     : ACCESS STRING;
   Vars    : ACCESS INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

THREAD_BLOCK snmpMonitor;
VAR
   count   : INT;
   oid     : STRING;
   rc      : INT;
   val     : snmpVariable;
   str     : STRING;
   event_t : USINT;
END_VAR;

WHILE TRUE DO
   rc := snmpWaitEvent(timeout := -1, event:=event_t, oid:=oid, vars:=count);
   CASE event_t OF
      _SNMP_EVENT_NOEVENT :
         CASE rc OF
             1: DebugMsg(message:="ERROR: NOEVENT should not give return code
1!");
             0: DebugMsg(message:="ERROR: The function is not supported!");
            -2: DebugMsg(message:="ERROR: Invalid input");
            -3: DebugMsg(message := "(Trap events lost)");
           -12: DebugMsg(message := "General error!");
         ELSE
            DebugFmt(message:="ERROR: unknown return code from snmpReadEvent
(rc=\1)", v1 := rc);
         END_CASE;
      _SNMP_EVENT_TIMEOUT :
          DebugMsg(message:="snmpWaitEvent: Timeout!");
      _SNMP_EVENT_TRAP :
```

```
            DebugMsg(message := "Trap [" + oid + "] received");
            DebugFmt(message := "Variables: \1", v1 := count);
         _SNMP_EVENT_SET :
            DebugMsg(message:="Set request received on OID " + oid);
      ELSE
         DebugFmt(message:="ERROR: snmpWaitEvent - unknown event type \1!", v1 :=
event_t);
      END_CASE;
      IF count > 0 THEN
         // Iterate variables
         rc := 1;
         WHILE rc = 1 DO
            rc := snmpGetNextVar(data := val);
            IF rc = 1 THEN
               // Build output
               str := "[" + val.oid + "] = ";
               CASE val.type OF
                  1: str := str + dintToStr(v := val.time_value);
                  2: str := str + dintToStr(v := val.int_value);
                  3: str := str + dintToStr(v := val.uint_value);
                  4: str := str + "[" + val.str_value + "]";
                  5: str := str + "[" + val.hex_value + "]";
                  6: str := str + "[" + val.oid_value + "]";
                  7: str := str + "[" + val.ip_value + "]";
                  8: str := str + floatToStr(v := val.float_value);
                  9: str := str + doubleToStr(v := val.double_value);
               ELSE
                  str := str + "Unknown variable type (" + intToStr(v :=
val.type) + ")";
               END_CASE;
            END_IF;
         END_WHILE;
         IF rc < 1 AND rc <> -11 THEN
            DebugFmt(message := "snmpGetNextVar=\1", v1 := rc);
         END_IF;
      END_IF;
   END_WHILE;
END_THREAD_BLOCK;

PROGRAM example;
VAR
   mon      : snmpMonitor;
   handle   : SYSHANDLE;
   iface    : SINT := 2;
   rc       : INT;
END_VAR;

   // Initialize network
   rc := netOpen(iface := iface);
   DebugFmt(message := "netOpen (rc=\1)", v1 := rc);
   WHILE NOT netConnected(iface := iface) DO
      Sleep(Delay := 2000);
   END_WHILE;

   // Initialize SNMP
   rc := snmpStartListen(
                         handle    := handle,
                         community := "public",
                         port      := 162
                        );
   DebugFmt(message := "snmpStartListen = \1", v1 := rc);
   rc := snmpRegisterTrap(oid := "1.3.6.1.4.1.6101.1.8.8.2.6");
   DebugFmt(message := "snmpRegisterTrap = \1", v1 := rc);
   mon();
```

```
BEGIN
END;
END_PROGRAM;
```

4.2.42.9.2 snmpGetNextVar (Function)

Architecture: NX32L
Device support: All NX devices
Firmware version: 1.50.00

Get the next SNMP trap variable associated with the last received event. See snmpWaitEvent. Call this function as many times as snmpWaitEvent reported that there was trap variables associated with the event to get all variables.

Input:

None.

Output:

Data : snmpVariable
The received value contained in a general structure. See snmpVariable.

Returns: INT

1- Data available
0- This function is not supported.
-2- Invalid parameter
-9- Invalid variable type.
-11- No more data

Declaration

```
FUNCTION snmpGetNextVar : INT;
VAR_INPUT
   Data     : ACCESS snmpVariable;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

THREAD_BLOCK snmpMonitor;
VAR
   count : INT;
   oid   : STRING;
   rc    : INT;
   val   : snmpVariable;
   str   : STRING;
END_VAR;

WHILE TRUE DO
   rc := snmpWaitEvent(timeout := -1, oid := oid, vars := count);
   IF rc = 1 OR rc = -5 THEN
      DebugMsg(message := "Trap [" + oid + "] received");
      IF rc = -5 THEN
         DebugMsg(message := "(Trap events lost)");
         rc := 1;
      END_IF;
      DebugFmt(message := "Variables: \1", v1 := count);
      IF count > 0 THEN
         // Iterate variables
         WHILE rc = 1 DO
```

```
            rc := snmpGetNextVar(data := val);
            IF rc = 1 THEN
               // Build output
               str := "[" + val.oid + "] = ";
               CASE val.type OF
                  1: str := str + dintToStr(v := val.time_value);
                  2: str := str + dintToStr(v := val.int_value);
                  3: str := str + dintToStr(v := val.uint_value);
                  4: str := str + "[" + val.str_value + "]";
                  5: str := str + "[" + val.hex_value + "]";
                  6: str := str + "[" + val.oid_value + "]";
                  7: str := str + "[" + val.ip_value + "]";
                  8: str := str + floatToStr(v := val.float_value);
                  9: str := str + doubleToStr(v := val.double_value);
               ELSE
                  str := str + "Unknown variable type (" + intToStr(v :=
val.type) + ")";
               END_CASE;
            END_IF;
         END_WHILE;
         IF rc < 1 AND rc <> -11 THEN
            DebugFmt(message := "snmpGetNextVar=\1", v1 := rc);
         END_IF;
      END_IF;
   ELSE
      DebugFmt(message := "snmpWaitEvent=\1", v1 := rc);
   END_IF;
END_WHILE;
END_THREAD_BLOCK;

PROGRAM example;
VAR
   mon      : snmpMonitor;
   handle   : SYSHANDLE;
   iface    : SINT := 2;
   rc       : INT;
END_VAR;

   // Initialize network
   rc := netOpen(iface := iface);
   DebugFmt(message := "netOpen (rc=\1)", v1 := rc);
   WHILE NOT netConnected(iface := iface) DO
      Sleep(Delay := 2000);
   END_WHILE;

   // Initialize SNMP
   rc := snmpStartListen(
                         handle    := handle,
                         community := "public",
                         port      := 162
                        );
   DebugFmt(message := "snmpStartListen = \1", v1 := rc);
   rc := snmpRegisterTrap(oid := "1.3.6.1.4.1.6101.1.8.8.2.6");
   DebugFmt(message := "snmpRegisterTrap = \1", v1 := rc);
   mon();

BEGIN
END;
END_PROGRAM;
```

4.2.42.9.3 snmpVariable (Structure)

```
Architecture:       NX32L
Device support:     All NX devices
Firmware version:   1.50.00
```

The snmpVariable structure contains the value of a received SNMP variable.

The structure is used with the snmpGetNextVar function.

Fields:

Type : SINT
The type of the variable.

OID : STRING
The OID of the variable.

Time_value : DINT
Timeticks, 32bit unsigned signed integer value. Type = 1.

INT_value : DINT
32bit signed integer value. Type = 2.

UINT_value : DINT
32bit unsigned integer value. Type = 3.

STR_value : STRING
String value. Type = 4.

Hex_value : STRING
Hex string value. Type= 5.

OID_value : STRING
OID variable value in string format. Type = 6.

IP_value : STRING
IP address value as a string. Type = 7.

FLOAT_value : FLOAT
Floating point value. Type = 8.

DOUBLE_value : DOUBLE
Double floating point value. Type = 9.

Declaration:

```
STRUCT_BLOCK snmpVariable;
   Type         : SINT;
   OID          : STRING;
   Time_value   : DINT;
   INT_value    : DINT;
   UINT_value   : DINT;
   STR_value    : STRING;
   Hex_value    : STRING;
   OID_value    : STRING;
   IP_value     : STRING;
   FLOAT_value  : FLOAT;
```

```
    DOUBLE_value : DOUBLE;
END_STRUCT_BLOCK;
```

4.2.43 so: Advanced socket functions (NX32L)

4.2.43.1 so: socket functions

The socket functions are an API used for transferring (sending and receiving) data across a network.

A socket is an abstract representation for the local endpoint of a communication path. The communication path can be across a network to a PC (or similar) or back to the device itself. This abstraction is introduced so that a single interface can be used for multiple network protocols. (e.g. TCP or UDP)

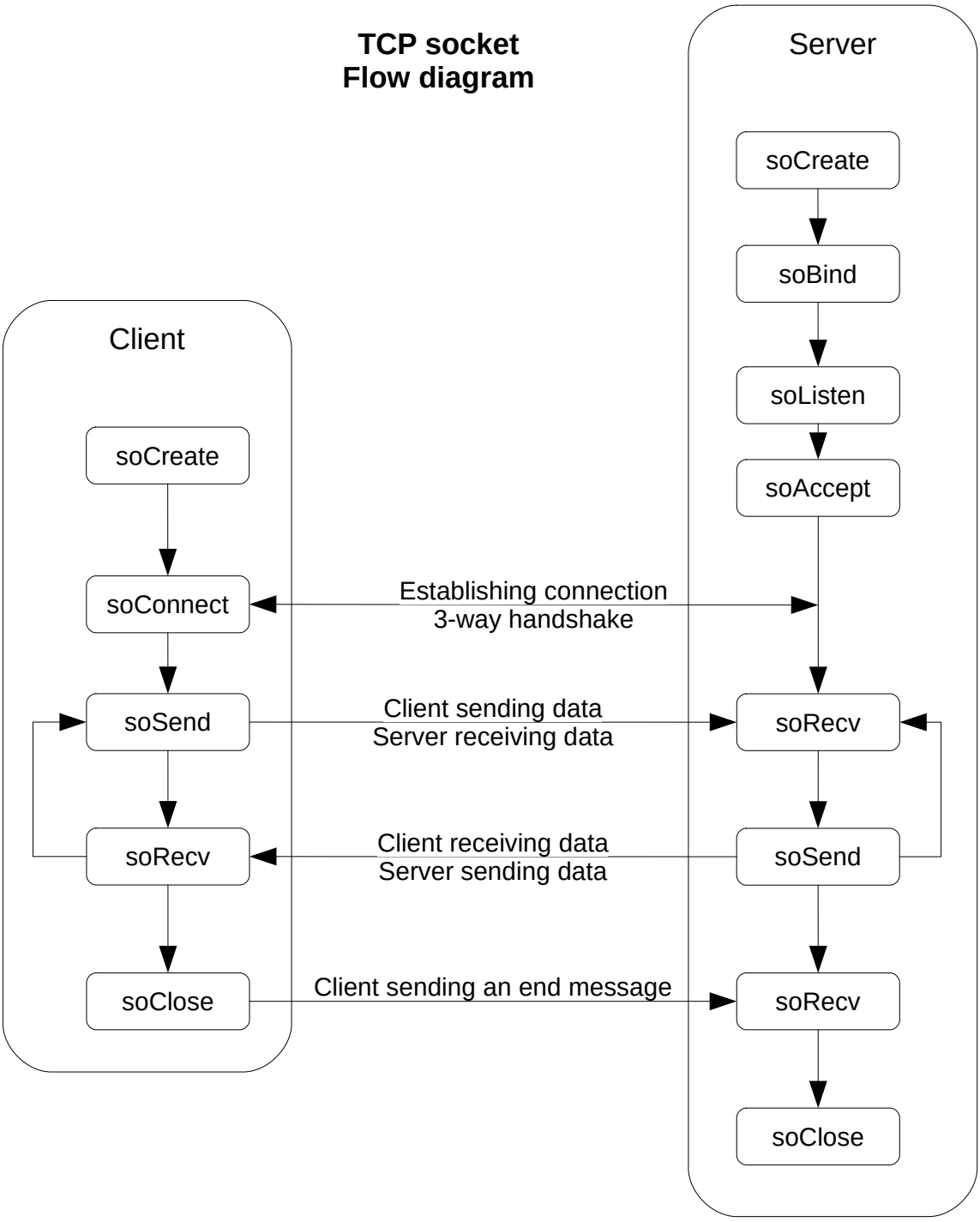
There are two types of sockets:

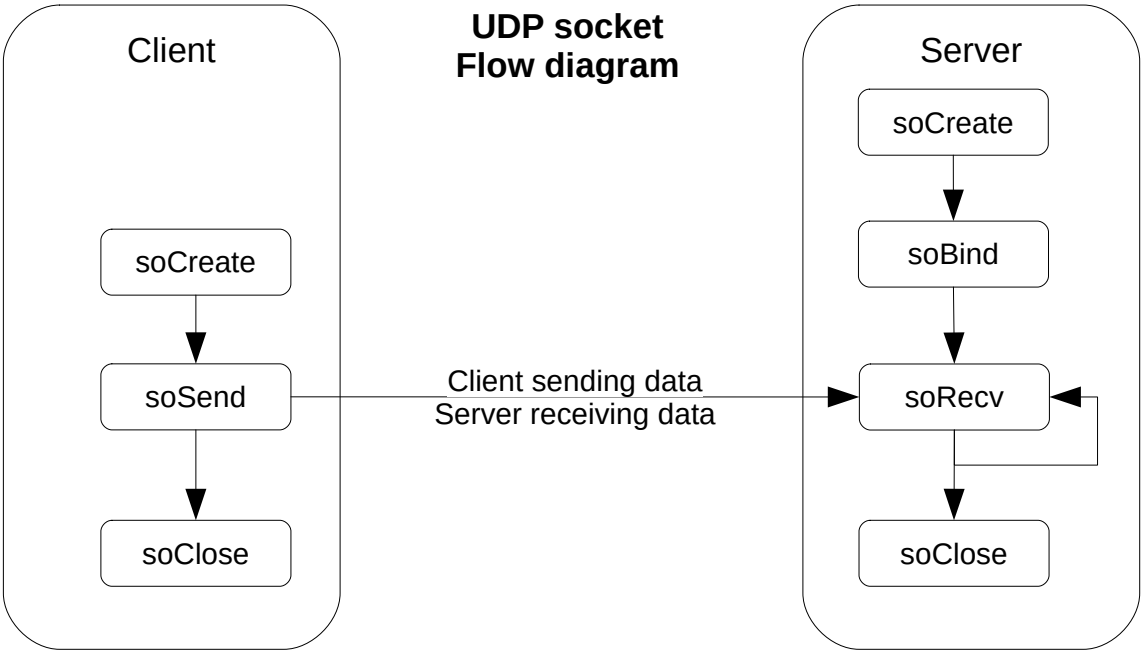
- Stream sockets
  Delivery in a networked environment is guaranteed.
  If you send through the stream socket three items "A, B, C", they will arrive in the same order − "A, B, C". These sockets use TCP (Transmission Control Protocol) for data transmission. If delivery is impossible, the sender receives an error indicator. Data records do not have any boundaries.

- Datagram sockets
  Delivery in a networked environment is not guaranteed.
  They're connectionless because you don't need to have an open connection as in Stream Sockets − you build a packet with the destination information and send it out. They use UDP (User Datagram Protocol).

The images below shows the use of a Stream socket and a Datagram socket:

TCP socket
Flow diagram



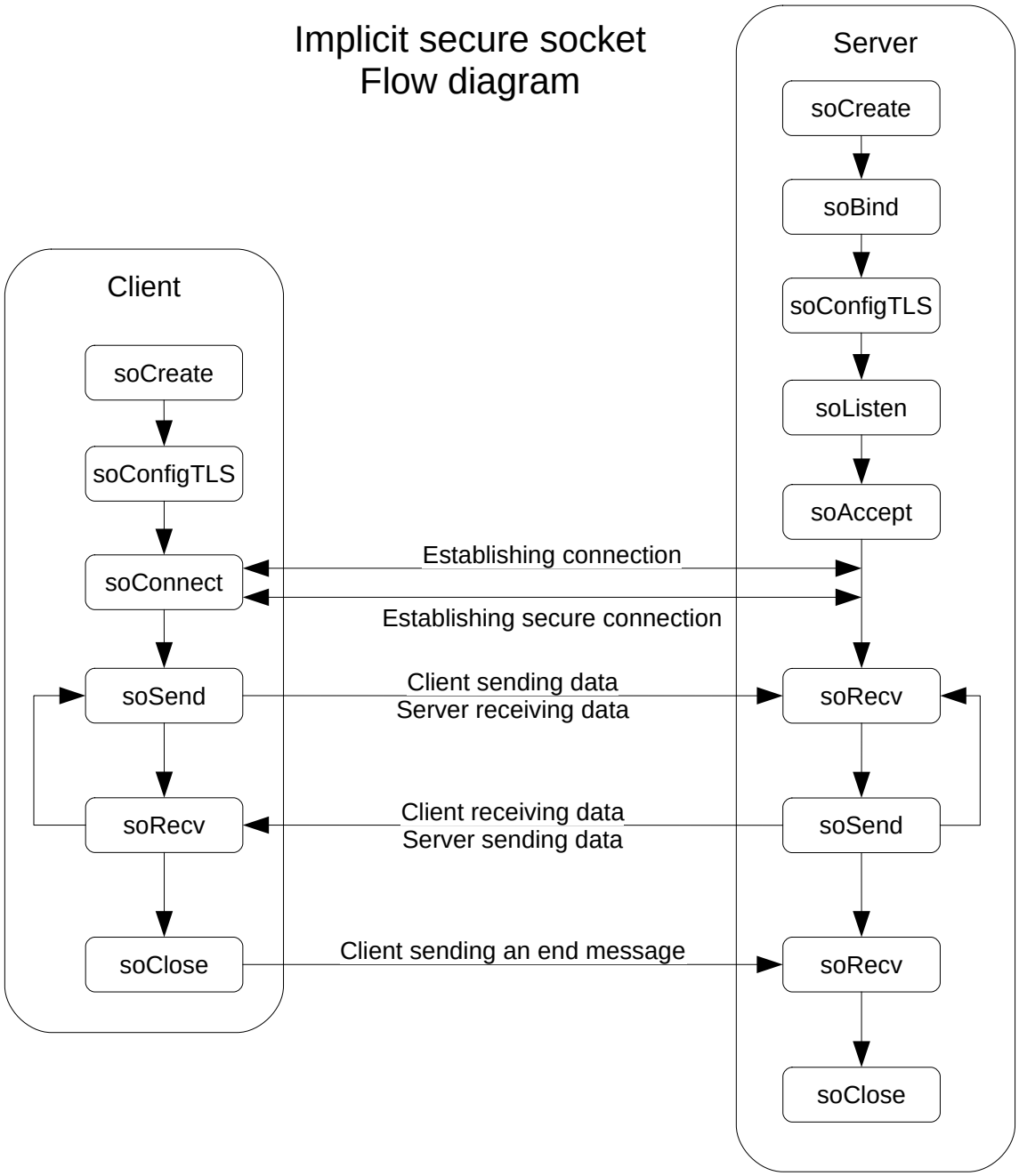


The default behavior of the socket functions is to block until the function is completed.
It is possible to configure a socket to be non-blocking, which means that the socket functions return immediately with an error (-4, would block), while the function is completed asynchronous.

Secure connections:
The socket API have support for secure connections using the Transport Layer Security (TLS) protocol.
This is built into the socket functions, so that no other functions are necessary.
A secure connection can be created both implicit and explicit.
(Implicit being that TLS is negotiated immediately after the connection is established, and Explicit being that TLS is negotiated after communicating on the socket)

Secure connections are only available for TCP sockets.
While secure connections can be both blocking and non-blocking, for implicit security the socket MUST be blocking.
(The socket can be switched to non-blocking after the secure connection is established)

The image below shows an example of a secure connection:



Functions for sockets:

• soOpen	Create a new socket
• soClose	Close a socket
• soBind	Bind socket to local address
• soConnect	Connect a socket to a remote address
• soListen	Start listening for incoming connections on socket
• soAccept	Accept an incoming connection
• soSend	Send data over a socket
• soRecv	Read data from a socket
• soRecvFrom	Read data from a socket
• soStatus	Get the status of a socket
• soConfigGet	Read configuration of a socket
• soConfigSet	Set configuration of a socket

- soConfigTLS — Configure TLS security of a socket
- soConfigNonblock — Configure non-blocking state of a socket

Option structures:

- soOptionLinger — The parameters for the linger option.

Functions for socket address:

- soAddrToIP — Convert a socket address to a 32 bit IP address
- soAddrFromIP — Convert an IP address to a socket address
- soAddrInetGet — Get the IP address and port from a socket address
- soAddrInetSet — Create a socket address with an IP address and port
- soAddrToInterface — Get the network interface from a socket address
- soAddrFromInterface — Get the socket address of a network interface
- soAddrLookup — Resolve a domain name or IP address

The implementation of sockets have the following limitations:
- The maximum number of sockets is limited to 512. The sockets are shared with FTP.

4.2.43.2 soCreate (Function)

```
Architecture:       NX32L
Device support:     All
Firmware version:   1.08.00
```

The soCreate function creates a socket for communication using a specific transport service provider.

Input:

domain : INT *(default _SO_DOMAIN_INET)*
The protocol family which will be used for communication on the socket.
- _SO_DOMAIN_INET - The Internet Protocol version 4 (IPv4) protocol family.

type : INT *(default _SO_TYPE_STREAM)*
The communication semantics of socket.
- _SO_TYPE_STREAM - Provides sequenced, reliable, two-way, connection-based byte streams.
- _SO_TYPE_DGRAM - Provides datagrams, connectionless, unreliable buffers of a fixed (typically small) maximum length.

protocol : INT *(default _SO_PROT_TCP)*
The protocol to be used by the socket.
- _SO_PROT_TCP - The Transmission Control Protocol (TCP).
- _SO_PROT_UDP - The User Datagram Protocol (UDP).

Output:

socket : SYSHANDLE
Handle to the socket.

Returns: INT

1 - Success.
0 - The function is not supported.
-2 - One or more parameters are illegal.
-15 - It is not possible to open a new socket.
-17 - Generic error.

Declaration:

```
FUNCTION soCreate : INT;
VAR_INPUT
   domain   : SINT := _SO_DOMAIN_INET;
   type     : SINT := _SO_TYPE_STREAM;
   protocol : SINT := _SO_PROT_TCP;
   socket   : ACCESS SYSHANDLE;
END_VAR;
```

Example:

Please see the "Examples - Telnet server2"

4.2.43.3 soClose (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.08.00

The soClose function closes a socket opened by the soCreate function. After calling this function the handle is no longer valid and can no longer be used with the socket functions.

Input:

socket : SYSHANDLE
Handle to the socket.

Returns: INT

1 - Success.
0 - The function is not supported.
-1 - The handle is not a valid socket.

Declaration:

```
FUNCTION soClose : INT;
VAR_INPUT
   socket : ACCESS SYSHANDLE;
END_VAR;
```

Example:

Please see the "Examples - Telnet server2"

4.2.43.4 soBind (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.08.00

The soBind function assigns a local address to a socket.

This function is required on an unconnected socket before subsequent calls to the soListen function, to set the port and optionally the interface to listen on.
It may be used on an unconnected socket before subsequent calls to the soConnect function, to set the interface and optionally the port.

If the address refer to a network interface, then the socket will only use this interface for communication.

Input:

socket : SYSHANDLE
Handle to the socket.

address : STRING
The address to bind to.
See also soAddrFromInterface.

Returns: INT

1	- Success.
0	- The function is not supported.
-1	- The handle is not a valid socket.
-2	- One or more parameters are illegal.
-3	- The socket is closed.
-8	- The local address is already used by another socket.
-16	- The socket is already bound.
-17	- Generic error.
-18	- Illegal address.

Declaration:

```
FUNCTION soBind : INT;
VAR_INPUT
   socket  : SYSHANDLE;
   address : STRING;
END_VAR;
```

Example:

Please see the "Examples - Telnet server2"

4.2.43.5 soConnect (Function)

```
Architecture:       NX32L
Device support:     All
Firmware version:   1.08.00
```

The soConnect function establishes a connection to the specified address.

On a blocking socket, the return value indicates success or failure of the connection attempt. With a non-blocking socket, the connection attempt cannot be completed immediately. In this case soConnect will return -4; however the connection attempt continues.

When a connection between sockets is broken, the socket that was connected should be discarded and a new socket should be created. When a problem develops on a connected

socket, it must be discarded and a new socket should be created, in order to return to a stable point.

For a connectionless socket (type _SO_TYPE_DGRAM), the operation performed by this function is merely to establish a default destination address that can be used on subsequent calls to the soSend function.

Input:

socket : SYSHANDLE
Handle to the socket.

address : STRING
The address to connect to.
See also soAddrInetSet.

Returns: INT

1	- Success.
0	- The function is not supported.
-1	- The handle is not a valid socket.
-2	- One or more parameters are illegal.
-3	- The socket is closed.
-4	- The socket is marked non-blocking and the operation would block.
-5	- The socket does not support this action.
-7	- The socket already have a connection.
-8	- The local address is already used by another socket.
-9	- The connection attempt was aborted.
-10	- Failed to connect to remote address.
-15	- It is not possible to open a new socket.
-17	- Generic error.
-101	- The certificate can not be validated.
-102	- The certificate is expired or not yet valid.
-103	- The root certificate cannot be validated.
-105	- The TLS handshake failed.

Declaration:

```
FUNCTION soConnect : INT;
VAR_INPUT
   socket  : SYSHANDLE;
   address : STRING;
END_VAR;
```

Example:

Please see the "Examples - Web client"

4.2.43.6 soListen (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.08.00

The soListen function marks the socket as listening for incoming connection attempts.
Any incoming connection attempts will be queued pending acceptance by the soAccept function.
If the queue is full, subsequent connection attempts will be refused.

The socket must be bound to a local address with the soBind function before calling this function.

Input:

socket : SYSHANDLE
Handle to the socket.

backlog : INT *(default 1)*
The maximum number of pending connections allowed.

Returns: INT

1	- Success.
0	- The function is not supported.
-1	- The handle is not a valid socket.
-2	- One or more parameters are illegal.
-3	- The socket is closed.
-5	- The socket does not support this action.
-8	- The local address is already used by another socket.
-17	- Generic error.

Declaration:

```
FUNCTION soListen : INT;
VAR_INPUT
   socket  : SYSHANDLE;
   backlog : INT := 1;
END_VAR;
```

Example:

Please see the "Examples - Telnet server2"

4.2.43.7 soAccept (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.08.00

The soAccept function will accept the first pending connection attempt, if present, in the queue of a listening socket.
A new socket will be created for the resulting connection.

On a blocking socket, the function will wait until an incoming connection attempt is received before returning.
With a non-blocking socket, the function will return immediately if no pending connection attempts are present. In this case soAccept will return -4.

The socket must be marked as listening for incoming connections with the soListen function before calling this function.

Input:

socket : SYSHANDLE
Handle to the socket.

Output:

remote : SYSHANDLE
Handle to the new socket.

Returns: INT

1	- Success.
0	- The function is not supported.
-1	- The handle is not a valid socket.
-2	- One or more parameters are illegal.
-3	- The socket is closed.
-4	- The socket is marked non-blocking and the operation would block.
-9	- The connection attempt was aborted.
-11	- Memory error.
-15	- It is not possible to open a new socket.
-17	- Generic error.
-101	- The certificate can not be validated.
-102	- The certificate is expired or not yet valid.
-103	- The root certificate cannot be validated.
-104	- The client did not send a certificate.
-105	- The TLS handshake failed.

Declaration:

```
FUNCTION soAccept : INT;
VAR_INPUT
   socket : SYSHANDLE;
   remote : ACCESS SYSHANDLE;
END_VAR;
```

Example:

Please see the "Examples - Telnet server2"

4.2.43.8 soSend (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.08.00

The soSend function transmits data on a socket.
When the function returns success, this does not mean that the data is delivered and received by the recipient; only that the data was sent.

On a blocking socket, the function will not return until the data is sent.
With a non-blocking socket, if the data do not fit into the send buffer, the function will return -4.

For a connection-based socket (type _SO_TYPE_STREAM), the socket must first be connected by the soConnect or soAccept functions before this function can be called.
For a connectionless socket (type _SO_TYPE_DGRAM), the destination given by the address parameter. The address parameter can be ignored if the soConnect function is used to set the destination.
Also, for a connectionless socket, care must be taken to not exceed the maximum packet size of the underlying protocol. If the data is too large to pass through atomically, no data is is sent and an error is returned.

Input:

socket : SYSHANDLE
Handle to the socket.

data : PTR
Address of the buffer that contains the data to be sent.

size : DINT
Number of bytes to send from the buffer.

address : STRING
The destination address. The address is ignored if a connection is established.

Output:

sent : DINT
The number of bytes sent.

Returns: INT

1	- Success.
0	- The function is not supported.
-1	- The handle is not a valid socket.
-2	- One or more parameters are illegal.
-3	- The socket is closed.
-4	- The socket is marked non-blocking and the operation would block.
-6	- The socket require a connection and is not connected.
-7	- The socket already have a connection.
-11	- Memory error.
-12	- The action requires an address.
-13	- The data size is too large.
-14	- One or more flags are invalid.
-17	- Generic error.
-110	- The secure transfer failed.
-111	- The secure connection is closed.

Declaration:

```
FUNCTION soSend : INT;
VAR_INPUT
   socket  : SYSHANDLE;
   data    : PTR;
   size    : DINT;
   address : STRING;
   sent    : ACCESS DINT;
END_VAR;
```

Example:

Please see the "Examples - Telnet server2"

4.2.43.9 soRecv (Function)

```
Architecture:       NX32L
Device support:     All
Firmware version:   1.08.00
```

The soRecv function receives data from a socket.
The function return any data available, up to the requested amount, rather than waiting for receipt of the full amount requested.

On a blocking socket, the function will not return until data is received.
With a non-blocking socket, if no data is available, the function will return -4.

For a connection-based socket (type _SO_TYPE_STREAM), the socket must first be connected by the soConnect or soAccept functions before this function can be called.
For a connectionless socket (type _SO_TYPE_DGRAM), the socket must be bound to a local address with the soBind function before this function can be called.

Input:

socket : SYSHANDLE
Handle to the socket.

data : PTR
Address of the buffer that contains the received data.

maxsize : DINT
Maximum number of bytes that can be received (size of "data").

Output:

size : DINT
The number of bytes received.

Returns: INT

1	- Success.
0	- The function is not supported.
-1	- The handle is not a valid socket.
-2	- One or more parameters are illegal.
-3	- The socket is closed.
-4	- The socket is marked non-blocking and the operation would block.
-6	- The socket require a connection and is not connected.
-11	- Memory error.
-17	- Generic error.
-110	- The secure transfer failed.
-111	- The secure connection is closed.

Declaration:

```
FUNCTION soRecv : INT;
VAR_INPUT
   socket  : SYSHANDLE;
   data    : PTR;
   maxsize : DINT;
   size    : ACCESS DINT;
END_VAR;
```

Example:

Please see the "Examples - Telnet server2"

4.2.43.10 soRecvFrom (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.08.00

The soRecvFrom function is identical to the soRecv function, except it also returns the local address and the address the data originated from.
The soRecvFrom function is typically used with connectionless sockets, where this is the only way to get the source address (and local address).

Input:

socket : SYSHANDLE
Handle to the socket.

data : PTR
Address of the buffer that contains the received data.

maxsize : DINT
Maximum number of bytes that can be received (size of "data").

Output:

size : DINT
The number of bytes received.

local : STRING
The socket address of the network interface where the data was received.

remote : STRING
The socket address of the source of the data.

Returns: INT

1	- Success.
0	- The function is not supported.
-1	- The handle is not a valid socket.
-2	- One or more parameters are illegal.
-3	- The socket is closed.
-4	- The socket is marked non-blocking and the operation would block.
-6	- The socket require a connection and is not connected.
-11	- Memory error.
-17	- Generic error.
-110	- The secure transfer failed.
-111	- The secure connection is closed.

Declaration:

```
FUNCTION soRecvFrom : INT;
VAR_INPUT
   socket  : SYSHANDLE;
   data    : PTR;
   maxsize : DINT;
   size    : ACCESS DINT;
   local   : ACCESS STRING;
```

```
   remote  : ACCESS STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM udp_example;
VAR
  handle  : SYSHANDLE;
  address : STRING;

  buf     : ARRAY [1..230] OF SINT;
  size    : DINT;
  local   : STRING;
  remote  : STRING;
  rc      : INT;

  port    : DINT;
  host    : STRING;
  str     : STRING;
END_VAR;

// Open network
netOpen(iface := _NET_IFACE_LAN1);

// Open UDP socket
rc := soCreate(type := _SO_TYPE_DGRAM, protocol := _SO_PROT_UDP, socket :=
handle);
IF rc < 1 THEN
  // Error handling
  DebugFmt(message := "soCreate=\1", v1 := rc);
END_IF;

// Listen to port 5022 on any network interface
soAddrInetSet(address := address, port := 5022);
rc := soBind(socket := handle, address := address);
IF rc < 1 THEN
  // Error handling
  DebugFmt(message := "soBind=\1", v1 := rc);
END_IF;

BEGIN
  // Read data (blocks until data is available)
  rc  := soRecvFrom(
                    socket  := handle,
                    data    := ADDR(buf),
                    maxsize := SIZEOF(buf),
                    size    := size,
                    local   := local,
                    remote  := remote
                   );
  IF rc < 1 THEN
    // Error handling
  END_IF;

  // Get information
  soAddrInetGet(address := remote, host := host, port := port);
  str := strFromMemory(src := ADDR(buf), len := INT(size));

  // Show information
  DebugFmt(message := " soRecvFrom    = \1", v1 := rc);
```

```
  DebugFmt(message := "    interface  = \1", v1 := soAddrToInterface(address
:= local));
  DebugMsg(message := "    IP address = " + host);
  DebugFmt(message := "    IP port    = \4", v4 := port);
  DebugMsg(message := "    data       = [" + str + "]");
END;
END_PROGRAM;
```

4.2.43.11 soStatus (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.08.00

The soStatus function will return information about the socket.

While this function can be used on a connectionless socket (type _SO_TYPE_DGRAM), the information returned is rarely useful.
(The status returned will always be 1 (not connected) and the remote address will only be valid if set by the soConnect function)

Input:

socket : SYSHANDLE
Handle to the socket.

Output:

local : STRING
The socket address of the local end of the connection.

remote : STRING
The socket address of the remote end of the connection.

Returns: INT

6 - The socket is connected and secured with TLS.
5 - The socket is connected.
4 - The socket is connecting.
3 - The socket is listening.
2 - The socket is closed.
1 - The socket is not connected.
0 - The function is not supported.
-1 - The handle is not a valid socket.
-2 - One or more parameters are illegal.

Declaration:

```
FUNCTION soStatus : INT;
VAR_INPUT
   socket  : SYSHANDLE;
   local   : ACCESS STRING;
   remote  : ACCESS STRING;
END_VAR;
```

Example:

Please see the "Examples - Telnet server2"

4.2.43.12 soConfigGet (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.08.00

The soConfigGet function retrieves the current value of a socket option that can be set with soConfigSet.

The available options are:

option	data type	description
_SO_OPTION_ACCEPTCONN	BOOL	Query if the socket has been marked to accept incoming connections.
_SO_OPTION_BROADCAST	BOOL	Query if the socket are allowed to send data to a broadcast address. (_SO_PROT_UDP only)
_SO_OPTION_KEEPALIVE	BOOL	Query if keep-alive messages are enabled in the socket (_SO_PROT_TCP only)
_SO_OPTION_LINGER	soOptionLinger	Query if the socket should linger when closed. (_SO_PROT_TCP only)

Input:

socket : SYSHANDLE
Handle to the socket.

option : INT
The option to query.

data : PTR
A pointer to a buffer where the option data is stored.

Returns: INT

1	- Success.
0	- The function is not supported.
-1	- The handle is not a valid socket.
-2	- One or more parameters are illegal.
-3	- The socket is closed.
-5	- The socket does not support this action.
-17	- Generic error.

Declaration:

```
FUNCTION soConfigGet : INT;
VAR_INPUT
   socket : SYSHANDLE;
   option : INT;
   data   : PTR;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   handle : SYSHANDLE;
   val    : BOOL;
   rc     : INT;
END_VAR;

BEGIN
  ...
  // Get socket option
  rc := soConfigGet(
                    socket := handle,
                    option := _SO_OPTION_ACCEPTCONN,
                    data   := ADDR(val)
                   );
  DebugFmt(message := "soConfigGet=\1 accept=\2", v1 := rc, v2 := INT(val));
  ...
END;
END_PROGRAM;
```

4.2.43.13 soConfigSet (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.08.00

The soConfigSet function sets a socket option.

The available options are:

option	data type	description
_SO_OPTION_BROADCAST	BOOL	Configure the socket to allow sending data to a broadcast address. (_SO_PROT_UDP only)
_SO_OPTION_KEEPALIVE	BOOL	Enable sending keep-alive messages in the socket (_SO_PROT_TCP only)
_SO_OPTION_LINGER	soOptionLinger	Configure the linger time in the socket. (_SO_PROT_TCP only)

Input:

socket : SYSHANDLE
Handle to the socket.

option : INT
The option to set.

data : PTR
A pointer to a buffer where the option data is stored.

Returns: INT

1 - Success.
0 - The function is not supported.
-1 - The handle is not a valid socket.
-2 - One or more parameters are illegal.

-3 - The socket is closed.
-5 - The socket does not support this action.
-17 - Generic error.

Declaration:

```
FUNCTION soConfigSet : INT;
VAR_INPUT
   socket : SYSHANDLE;
   option : INT;
   data   : PTR;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
   handle : SYSHANDLE;
   val    : soOptionLinger;
   rc     : INT;
END_VAR;

BEGIN
  ...
  // Setup
  val.enable := ON;
  val.linger := 10;

  // Set option
  rc := soConfigSet(
                    socket := handle,
                    option := _SO_OPTION_LINGER,
                    data   := ADDR(val)
                   );
  DebugFmt(message := "soConfigGet=\1", v1 := rc);
  ...
END;
END_PROGRAM;
```

4.2.43.14 soConfigTLS (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.08.00

The soConfigTLS function enables or disables the use of a secure TLS connection for a socket. The secure TLS (TLS v1.2, v1.1 or 1.0) connection can be established assuming that a matching X509 certificate(s) is present.

If a connection is present when the function is called, the secure connection will be established immediately.
Otherwise the secure connection will be established together with the socket connection. (in the soConnect or soAccept functions)

Using TLS as a client:

1. Ensure the root certificate needed to verify the server is present.

2. Set enable := TRUE' when calling soConfigTLS.

If client verification is required by server then:
1. Ensure the client certificate needed to verify the device is present.
2. Set 'certificate' and 'password' according to the installed client certificate when calling soConfigTLS.

Using TLS as a server:

1. Ensure the server certificate used to create the secure connection, and identify the device, is present.
2. Set enable := TRUE' when calling soConfigTLS.
3. Set 'certificate' and 'password' according to the installed server certificate when calling soConfigTLS.

To enable verification of the connecting client(s):
1. Ensure the root certificate needed to verify the connecting client is present.
2. Set 'peer_cert := TRUE' when calling soConfigTLS.

Input:

socket : SYSHANDLE
Handle to the socket.

enable : BOOL
TRUE: Enable secure TLS connection.
FALSE: Disable secure TLS connection.

peer_cert : BOOL
TRUE: A client certificate is required by incoming connections.
FALSE: Incoming connections do not require a client certificate..
Only used if listening for incoming connections.

certificate : STRING
The certificate to use for the secure connections.
Only include if listening for incoming connections, or if the server require clients to include a certificate.

password : STRING
The password for the certificate if required.

Returns: INT

1	- Success.
0	- The function is not supported.
-1	- The handle is not a valid socket.
-2	- One or more parameters are illegal.
-3	- The socket is closed.
-5	- The socket does not support this action.
- 10	- Timeout establishing the secure connection.
-17	- Generic error.
-19	- TLS is already enabled in socket.
-101	- The certificate can not be validated.
-102	- The certificate is expired or not yet valid.
-103	- The root certificate cannot be validated.
-104	- The client did not send a certificate.
-105	- The TLS handshake failed.
-106	- Configuration of CA certificate failed.
-107	- Configuration of certificate failed.

-108 - Configuration of certificate key failed.
-109 - Configuration of TLS failed.

Declaration:

```
FUNCTION soConfigTLS : INT;
VAR_INPUT
   socket      : SYSHANDLE;
   enable      : BOOL;
   peer_cert   : BOOL;
   certificate : STRING;
   password    : STRING;
END_VAR;
```

Example:

Please see the "Examples - Web client"

4.2.43.15 soConfigNonblock (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.08.00

The soConfigNonblock function configure the non-blocking state of a socket.

When a socket is marked as blocking, the socket functions will block and wait until their function is completed.
For example the soConnect function will block until the socket connection is established.

When a socket is marked as non-blocking, the socket functions will return immediately with error -4 (would block) instead of blocking.
It is then up to the application to determine when the function is ready to complete.
(In case of the example above, by calling the soStatus function to determine when the socket is connected)

Input:

socket : SYSHANDLE
Handle to the socket.

enable : BOOL
Set TRUE to make the socket non-blocking and FALSE to make the socket blocking.

Returns: INT

1 - Success.
0 - The function is not supported.
-1 - The handle is not a valid socket.
-2 - One or more parameters are illegal.
-3 - The socket is closed.
-5 - The socket does not support this action.
-17 - Generic error.

Declaration:

```
FUNCTION soConfigNonblock : INT;
VAR_INPUT
   socket : SYSHANDLE;
   enable : BOOL;
END_VAR;
```

Example:

Please see the "Examples - Web client"

4.2.43.16 soAddrToIP (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.08.00

Convert a socket address to a 32 bit IP address.

Input:

address : STRING
The socket address.

Returns: DINT

The 32 bit IP address. 0 will be returned if address cannot be resolved.

Declaration:

```
FUNCTION soAddrToIP : DINT;
VAR_INPUT
   address : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
  address : STRING;
  ip      : DINT;
END_VAR;

BEGIN
  ...
  DebugMsg(message := "Socket address = [" + address + "]");

  // Convert a socket address
  ip := soAddrToIP(address := address);

  DebugFmt(message := "IP address = \4", v4 := ip);
  ...
END;
END_PROGRAM;
```

4.2.43.117 soAddrFromIP (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.08.00

Convert a 32 bit IP address to a socket address.

Input:

ip : DINT
The 32 bit IP address.

Returns: STRING

The socket address.

Declaration:

```
FUNCTION soAddrFromIP : STRING;
VAR_INPUT
   ip : DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
  address : STRING;
  ip      : DINT;
END_VAR;

BEGIN
  ...
  DebugFmt(message := "IP address = \4", v4 := ip);

  // Convert a socket address
  address := soAddrFromIP(ip := ip);

  DebugMsg(message := "Socket address = [" + address + "]");
  ...
END;
END_PROGRAM;
```

4.2.43.118 soAddrInetGet (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.08.00

The soAddrInetGet function will return the IP address and port from a socket address.

Input:
address : STRING
The socket address.

Output:
host : STRING
The IP address.

port : DINT
The IP port number.

Returns: INT
1 - Success.
0 - The function is not supported.

Declaration:
```
FUNCTION soAddrInetGet : INT;
VAR_INPUT
   address : STRING;
   host    : ACCESS STRING;
   port    : ACCESS DINT;
END_VAR;
```

Example:
```
INCLUDE rtcu.inc

PROGRAM udp_example;
VAR
  handle  : SYSHANDLE;
  address : STRING;

  buf     : ARRAY [1..230] OF SINT;
  size    : DINT;
  local   : STRING;
  remote  : STRING;
  rc      : INT;

  port    : DINT;
  host    : STRING;
  str     : STRING;
END_VAR;

// Open network
netOpen(iface := _NET_IFACE_LAN1);

// Open UDP socket
rc := soCreate(type := _SO_TYPE_DGRAM, protocol := _SO_PROT_UDP, socket :=
handle);
IF rc < 1 THEN
  // Error handling
  DebugFmt(message := "soCreate=\1", v1 := rc);
END_IF;

// Listen to port 5022 on any network interface
soAddrInetSet(address := address, port := 5022);
rc := soBind(socket := handle, address := address);
IF rc < 1 THEN
```

```
  // Error handling
  DebugFmt(message := "soBind=\1", v1 := rc);
END_IF;

BEGIN
  // Read data (blocks until data is available)
  rc  := soRecvFrom(
                    socket  := handle,
                    data    := ADDR(buf),
                    maxsize := SIZEOF(buf),
                    size    := size,
                    local   := local,
                    remote  := remote
                   );
  IF rc < 1 THEN
    // Error handling
  END_IF;

  // Get information
  soAddrInetGet(address := remote, host := host, port := port);
  str := strFromMemory(src := ADDR(buf), len := INT(size));

  // Show information
  DebugFmt(message := " soRecvFrom    = \1", v1 := rc);
  DebugFmt(message := "    interface  = \1", v1 := soAddrToInterface(address
:= local));
  DebugMsg(message := "    IP address = " + host);
  DebugFmt(message := "    IP port    = \4", v4 := port);
  DebugMsg(message := "    data       = [" + str + "]");
END;
END_PROGRAM;
```

4.2.43.19 soAddrInetSet (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.08.00

The soAddrInetSet function will create a socket address with an IP address and port.
It is possible to modify an existing socket address with this function, for example to change the port.

Input:

host : STRING
The IP address. If empty, the IP address will not be set in the socket address.

port : DINT (*0..65535, default 0*)
The IP port number. If zero, the port will not be set in the socket address.

Output:

address : STRING
The socket address.

Returns: INT

1 - Success.
0 - The function is not supported.
-2 - One or more parameters are illegal.

Declaration:

```
FUNCTION soAddrInetSet : INT;
VAR_INPUT
   address : ACCESS STRING;
   host    : STRING;
   port    : DINT;
END_VAR;
```

Example:

Please see the "Examples - Web client"

4.2.43.20 soAddrToInterface (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.08.00

Get the network interface from a socket address.

Input:

address : STRING
The socket address.

Returns: SINT

The network interface. 1 = Mobile network, 2 = LAN network, etc. (See Network).
 0 - The function is not supported.
 -2 - One or more parameters are illegal.
 -17 - Generic error.

Declaration:

```
FUNCTION soAddrToInterface : SINT;
VAR_INPUT
   address : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM udp_example;
VAR
  handle  : SYSHANDLE;
  address : STRING;

  buf     : ARRAY [1..230] OF SINT;
  size    : DINT;
  local   : STRING;
  remote  : STRING;
  rc      : INT;

  port    : DINT;
```

```
  host    : STRING;
  str     : STRING;
END_VAR;

// Open network
netOpen(iface := _NET_IFACE_LAN1);

// Open UDP socket
rc := soCreate(type := _SO_TYPE_DGRAM, protocol := _SO_PROT_UDP, socket :=
handle);
IF rc < 1 THEN
  // Error handling
  DebugFmt(message := "soCreate=\1", v1 := rc);
END_IF;

// Listen to port 5022 on any network interface
soAddrInetSet(address := address, port := 5022);
rc := soBind(socket := handle, address := address);
IF rc < 1 THEN
  // Error handling
  DebugFmt(message := "soBind=\1", v1 := rc);
END_IF;

BEGIN
  // Read data (blocks until data is available)
  rc  := soRecvFrom(
                    socket  := handle,
                    data    := ADDR(buf),
                    maxsize := SIZEOF(buf),
                    size    := size,
                    local   := local,
                    remote  := remote
                   );
  IF rc < 1 THEN
    // Error handling
  END_IF;

  // Get information
  soAddrInetGet(address := remote, host := host, port := port);
  str := strFromMemory(src := ADDR(buf), len := INT(size));

  // Show information
  DebugFmt(message := " soRecvFrom    = \1", v1 := rc);
  DebugFmt(message := "    interface  = \1", v1 := soAddrToInterface(address
:= local));
  DebugMsg(message := "    IP address = " + host);
  DebugFmt(message := "    IP port    = \4", v4 := port);
  DebugMsg(message := "    data       = [" + str + "]");
END;
END_PROGRAM;
```

4.2.43.21 soAddrFromInterface (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.08.00

Get the socket address from a network interface.

Input:

iface : SINT
The network interface. 1 = Mobile network, 2 = LAN network, etc. (See Network).

Returns: STRING
The socket address.

Declaration:
```
FUNCTION soAddrFromInterface : STRING;
VAR_INPUT
   iface : SINT;
END_VAR;
```

Example:

Please see the "Examples - Web client"

4.2.43.22 soAddrLookup (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.08.00

Resolve an IP address, like "80.62.33.111", or a symbolic name, like "domain.com", to a socket address.

Input:
str : STRING
IP address (in dotted format "aaa.bbb.ccc.ddd" or symbolic "domain.xyz").

iface : SINT *(default 0)*
The network interface to use. 0 = Default, 1 = Mobile network, 2 = LAN network, etc. (See Network)

Output:
address : STRING
The socket address.

Returns: INT
```
   1   - Success.
   0   - The function is not supported.
  -2   - One or more parameters are illegal.
 -17   - Generic error.
```

Declaration:
```
FUNCTION soAddrLookup : INT;
VAR_INPUT
   str     : STRING;
   iface   : SINT := 0;
   address : ACCESS STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
VAR
  address : STRING;
  port    : DINT;
  host    : STRING;
  rc      : INT;
END_VAR;

  netOpen(iface := 2);

BEGIN
  ...
  // Look up address
  rc := soAddrLookup(str := "www.example.com", iface := 2, address :=
address);
  DebugFmt(message := "soAddrLookup   = \1", v1 := rc);

  // Show address
  soAddrInetGet(address := address, host := host, port := port);
  DebugMsg(message := "    IP address = " + host);
  ...
END;
END_PROGRAM;
```

4.2.43.23 soOptionLinger (Structure)

Architecture: NX32L
Device support: All
Firmware version: 1.00.00

The soOptionLinger structure contains the parameters for the _SO_OPTION_LINGER option in a socket.
The _SO_OPTION_LINGER option determine how a socket should behave when there are unsent data and the soClose function is called.

When this option is enabled, the socket will remain open for the number of seconds set, or until the data is sent.
When this option is not enabled the socket will be closed immediately and the unsent data will be lost.

The structure is used by the soConfigGet and and soConfigSet functions to get and set the _SO_OPTION_LINGER option in a socket.

Fields:

enable : BOOL
Enable (TRUE) or Disable (FALSE) the linger function.

linger : DINT
The number of seconds the socket should linger.

Declaration:

```
STRUCT_BLOCK soOptionLinger;
   enable     : BOOL;
   linger     : DINT;
END_STRUCT_BLOCK;
```

4.2.44 sock: TCP/IP socket functions

4.2.44.1 sock: TCP/IP socket functions

Socket functions are used for communicating over a mobile network or LAN. The sock() functions uses TCP/IP. For UDP/IP support, please see UDP/IP socket functions.

Please see the example program (shown for each function).

- sockConnect — Initiates a socket connection
- sockConnection — Returns the current status of a connection
- sockDisconnect — Disconnects a socket
- sockGetLocalIP — Returns the local IP address
- sockIPFromName — Converts a string to an IP address
- sockIPToName — Converts an IP address to a string
- sockListen — Listen for a connection
- sockReceive — Reads data from the connection
- sockSend — Sends data over the connection
- sockSetTCPIPParm — Sets connection parameters
- sockGetTCPIPParm — Reads the connection parameters

Please also see the example application section for examples.

For NX32L based devices it is recommended to use the Advanced so socket API.

IP address format:
In all places where an IP address aaa.bbb.ccc.ddd is stored in a DINT, the following format is used:

ddd is stored in bit 24..31.
ccc is stored in bit 16..23.
bbb is stored in bit 8..15.
aaa is stored in bit 0..7.

4.2.44.2 sockIPFromName (Function)

```
Architecture:       X32 / NX32 / NX32L
Device support:     All
Firmware version:   1.00 / 1.00.00
```

Makes a 32 bit IP address out of a dotted format like "80.62.33.111". It can also resolve a symbolic name like "logicio.com".

Note: when the default network interface is selected, the sockIPFromName function will try to resolve the symbolic name using the first connected network. The sockIPFromName function will look for a connected network, starting with Mobile network and working up. (LAN1 network, then LAN 2 network, etc.)

Input:

str : STRING
IP address (in dotted format "aaa.bbb.ccc.ddd" or symbolic "domain.xyz").

iface : SINT *(default 0)*
The network interface to use. 0 = Default, 1 = Mobile network, 2 = LAN network, etc. (See Network)

Returns: **DINT**
32 bit IP address. 0 will be returned if address cannot be resolved.

Declaration:

```
FUNCTION sockIPFromName : DINT;
VAR_INPUT
   str   : STRING;    // IP address (aaa.bbb.ccc.ddd or "domain.xxx")
   iface : SINT := 0; // Interface to resolve on
END_VAR;
```

Example:

Please see the "Examples - Socket Communication"

4.2.44.3 sockIPToName (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

Converts a 32 bit IP address into dotted format like "80.62.33.111".

Input:
ip : DINT
32 bit IP address.

Returns: **STRING**
The IP address as a dotted format string "aaa.bbb.ccc.ddd".

Declaration:

```
FUNCTION sockIPToName : STRING;
VAR_INPUT
   ip : DINT;  // IP address
END_VAR;
```

Example:

Please see the "Examples - Socket Communication"

4.2.44.4 sockConnect (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

Initiates a TCP/IP socket connection to the specified IP address/port number. When this function returns, the connection has not been finally established but only initiated. The establishing of the connection must be determined by using the sockConnection() function block.
It is possible to establish a maximum of 8 simultaneous socket connections.

Please note that for each sockConnect() there must be a sockDisconnect(), even if the connection is closed from the "outside" (see example below).

Input:

ip : DINT
32 bit IP address to connect to.

port : DINT
Port number to connect to an IP address.

iface : SINT *(default 0)*
The network interface to use. 0 = Default network, 1 = Mobile network, 2 = LAN network, etc. (See Network)
Note: For backwards compatibility the Mobile network is used as default network.

Returns: SINT

ID of the connection. If return value is 0, the connection could not be initiated (possibly because there are too many sockConnect() without corresponding sockDisconnect()).

Declaration:

```
FUNCTION sockConnect : SINT;
VAR_INPUT
   ip    : DINT;       // IP address to connect to
   port  : DINT;       // Port number to connect to
   iface : SINT := 0;  // Interface to connect on
END_VAR;
```

Example:

Please see the "Examples - Socket Communication"

4.2.44.5 sockListen (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

Initiates a TCP/IP socket listen operation on the specified port number. The establishing of the connection must be determined by using the sockConnection() function block.
It is possible to establish a maximum of 8 simultaneous socket connections.

It must be observed that typically the GSM-network utilizes a firewall that blocks inbound connections to the RTCU.

Input:

ip : DINT
Remote IP address accepted. 0 if all is allowed.

port : DINT
Port number listening on.

iface : SINT *(default 0)*
The network interface to listen on. 0 = all networks (default), 1 = Mobile network, 2 = LAN network, etc. (See Network)

Returns: SINT

ID of the connection. If return value is 0, the connection could not be initiated (possibly because there are too many sockListen() without corresponding sockDisconnect()).

Declaration:

```
FUNCTION sockListen : SINT;
VAR_INPUT
   ip    : DINT;       // Accept from this remote IP only. 0=all.
   port  : DINT;       // Port number to listen on
   iface : SINT := 0;  // Interface to listen on
END_VAR;
```

Example:

Please see the "Examples - Telnet server"

4.2.44.6 sockConnection (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This function block returns the current status of a connection initiated by using either the sockConnect() or sockListen() function.

Input:

id : SINT
ID of the connection.

Output:

Changed : BOOL
Connection info has changed.

Connected : BOOL
True if connected, false if disconnected.

localport : DINT
Local port number.

remoteip : DINT
IP address of peer.

remoteport : DINT
Port number of peer.

iface : SINT
The network interface used by the connection. 0 = Not Available, 1 = Mobile network, 2 = LAN network, etc. (See Network)

Declaration:

```
FUNCTION_BLOCK sockConnection;
VAR_INPUT
   id          : SINT; // ID of the connection.
END_VAR;
VAR_OUTPUT
   Changed     : BOOL; // New connection info
```

```
   Connected   : BOOL; // true if connected, false if disconnected
   localPort   : DINT; // Local portnumber
   remoteIP    : DINT; // IP address of peer
   remotePort  : DINT; // Portnumber on peer
   iface       : SINT; // Interface used by the connection
END_VAR;
```

Example:

Please see the "Examples - Socket Communication"

4.2.44.7 sockDisconnect (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

Disconnects a socket and the connection established. All references to the connection identification (ID) after this call are illegal.

Please note that for each sockConnect(), there must be a sockDisconnect(), even if the connection is closed from the "outside" (see example below).

Input:
id : SINT
ID of the connection.

Returns: **INT**
0 - Success.
1 - Illegal ID.

Declaration:

```
FUNCTION sockDisconnect : INT;
VAR_INPUT
   id : SINT;  // ID of the connection.
END_VAR;
```

Example:

Please see the "Examples - Socket Communication"

4.2.44.8 sockSend (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

Sends data on a socket. A connection must be established before any data can be sent.

Input:
id : SINT
ID of the connection.

data : PTR
Address of the buffer that contains the data to be sent.

size : INT
Number of bytes to send from the buffer.

Returns: INT
Number of bytes that were sent on the socket. This can be less than the requested number of bytes if the data has only partially been sent.

Declaration:
```
FUNCTION sockSend : INT;
VAR_INPUT
   id   : SINT; // ID of the connection.
   data : PTR;  // Address of the buffer to send
   size : INT;  // Number of bytes to send
END_VAR;
```

Example:

Please see the "Examples - Socket Communication"

4.2.44.9 sockReceive (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This function block reads data from the specified connection and will indicate when data is ready and placed into the receive buffer.
The data received will be available when indicated so by the "ready" output and until the next call to sockReceive.

Input:
id : SINT
ID of the connection.

data : PTR
Address of the buffer that contains the received data.

maxsize : INT
Maximum number of bytes that can be received (size of "data").

Output:
ready : BOOL
True if data has been received

size : INT
Number of bytes received (if "ready" is true).

Declaration:
```
FUNCTION_BLOCK sockReceive;
VAR_INPUT
   id      : SINT; // ID of the connection.
   data    : PTR;  // Address of receive buffer
```

```
   maxsize : INT;  // Maximum number of bytes we can receive (size of 'data')
END_VAR;
VAR_OUTPUT
   ready   : BOOL; // data is ready
   size    : INT;  // Number of bytes received
END_VAR;
```

Example:

Please see the "Examples - Socket Communication"

4.2.44.10 sockGetLocalIP (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

Returns the 32 bit IP address of the local IP address (the IP address of the RTCU).

Note: For backwards compatibility, if no interface is selected the function returns the IP address of the Mobile network.

Input:

iface : SINT ***(default 1)***
The network interface to query. 1 = Mobile network, 2 = LAN network, etc. (See Network)
Note: For backwards compatibility the Mobile network is used as default network.

Returns: DINT

Returns the 32 bit local IP address (the IP address of the RTCU).

Declaration:

```
FUNCTION sockGetLocalIP : DINT;
VAR_INPUT
   iface : SINT := 1;
END_VAR;
```

Example:

Please see the "Examples - Telnet server"

4.2.44.11 sockSetTCPIPParm (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

Sets TCP/IP-specific parameters for making connection via mobile networks. It has the same effect as setting the parameters via Device: Network: Network Settings or netSetMobileParam.

The device must be reset before the changes can take effect (for example by using boardReset).

Note: The IP address byte order for this function is ddd.ccc.bbb.aaa which is reversed compared to the standard format.

This function is obsolete and it is recommended to use netSetMobileParam.

Input:

IP : DINT *(default 0)*
The IP address of the device (normally set by negotiation).

SubnetMask : DINT *(default 0)*
The Subnet mask of the device (normally set by negotiation).

Gateway : DINT *(default 0)*
The TCP/IP gateway the device must use (normally set by negotiation).

DNS1 : DINT *(default 0)*
The first Domain Name Server the device should use to look up symbolic names (normally set by negotiation).

DNS2 : DINT *(default 0)*
The second Domain Name Server the device should use to look up symbolic names (normally set by negotiation).

Username : STRING (*Max 33 characters*)
The user name the device should use in order to connect to the mobile network.

Password : STRING (*Max 33 characters*)
The password the device should use in order to connect to the mobile network.

APN : STRING (*Max 33 characters*)
The APN the device should use in order to connect to the mobile network.

Authenticate : INT (default 3)
The PPP authentication type:

0 - None.
1 - PAP.
2 - CHAP.
3 - PAP/CHAP.

Returns:

None.

Declaration:

```
FUNCTION sockSetTCPIPParm;
VAR_INPUT
   // general TCP/IP parameters:
   IP           : DINT := 0;     // IP address of this device (typically set
when negotiating)
   SubnetMask   : DINT := 0;     // subnet mask (typically set when
negotiating)
   Gateway      : DINT := 0;     // gateway-address (typically set when
negotiating)
   DNS1         : DINT := 0;     // DNS-address 1 (typically set when
negotiating)
   DNS2         : DINT := 0;     // DNS-address 2 (typically set when
negotiating)
   Alive        : DINT := 0;     // keep alive in seconds.(0=disabled) (NOT
USED)
```

```
   // PPP parameters:
   Username     : STRING;        // username (max 33 char).
   Password     : STRING;        // password (max 33 char).

   // Dialup/mobile network parameters:
   APN          : STRING;        // APN (MAX 33 char)
   Authenticate : INT;           // Authentication type. 0=none, 1=PAP,
2=CHAP, 3=PAP/CHAP.
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   gprsParm  : sockGetTCPIPParm;
   gwParm    : sockGetGWParm;
   parmReset : BOOL := FALSE;
   CryptKey  : ARRAY[1..16] OF SINT;
END_VAR;

// Check mobile network and Gateway settings
gprsParm();
gwParm();
IF gprsParm.IP <> 0 OR gprsParm.SubnetMask <> 0 OR gprsParm.Gateway <> 0 OR
   gprsParm.DNS1 <> 0 OR gprsParm.DNS2 <> 0 OR gprsParm.Authenticate <> 0 OR
   strCompare(str1 := gprsParm.Username, str2 := "") <> 0 OR
   strCompare(str1 := gprsParm.Password, str2 := "") <> 0 OR
   strCompare(str1 := gprsParm.APN, str2 := "internet") <> 0 THEN
   // Set APN and keep the default values for the others
   sockSetTCPIPParm(APN := "internet");
   parmReset := TRUE;
END_IF;
IF NOT gwParm.GWEnabled OR gwParm.GWPort <> 5001 OR gwParm.CryptKey[1] <> 0 OR
   gwParm.CryptKey[2] <> 0 OR gwParm.CryptKey[3] <> 0 OR gwParm.CryptKey[4] <>
 0 OR
   gwParm.CryptKey[5] <> 0 OR gwParm.CryptKey[6] <> 0 OR gwParm.CryptKey[7] <>
 0 OR
   gwParm.CryptKey[8] <> 0 OR gwParm.CryptKey[9] <> 0 OR gwParm.CryptKey[10]
<> 0 OR
   gwParm.CryptKey[11] <> 0 OR gwParm.CryptKey[12] <> 0 OR gwParm.CryptKey[13]
 <> 0 OR
   gwParm.CryptKey[14] <> 0 OR gwParm.CryptKey[15] <> 0 OR gwParm.CryptKey[16]
 <> 0 OR
   strCompare(str1 := gwParm.GWIP, str2 := "gw.rtcu.dk") <> 0 OR
   strCompare(str1 := gwParm.GWKey, str2 := "AABBCCDD") <> 0 THEN
   // Clear encryption key
   CryptKey[1]  := 0;
   CryptKey[2]  := 0;
   CryptKey[3]  := 0;
   CryptKey[4]  := 0;
   CryptKey[5]  := 0;
   CryptKey[6]  := 0;
   CryptKey[7]  := 0;
   CryptKey[8]  := 0;
   CryptKey[9]  := 0;
   CryptKey[10] := 0;
   CryptKey[11] := 0;
   CryptKey[12] := 0;
   CryptKey[13] := 0;
   CryptKey[14] := 0;
```

```
   CryptKey[15] := 0;
   CryptKey[16] := 0;
   // Set Gateway parameters
   sockSetGWParm(GWEnabled := TRUE, GWIP := "gw.rtcu.dk", GWPort := 5001,
GWKey := "AABBCCDD", CryptKey := ADDR(CryptKey));
   parmReset := TRUE;
END_IF;
IF parmReset THEN
   // Reset device before the changes are used
   boardReset();
END_IF;

// Turn on power to the GSM module
gsmPower(power := TRUE);
gprsOpen();

BEGIN
   ...
END;
END_PROGRAM;
```

4.2.44.12 sockGetTCPIPParm (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.21 / 1.00.00

---

sockGetTCPIPParm will read out the TCP/IP parameters previously set from the RTCU IDE or using by the sockSetTCPIPParm or netSetMobileParam functions.

Note: The IP address byte order for this function is ddd.ccc.bbb.aaa which is reversed compared to the standard format.

This function is obsolete and it is recommended to use netGetMobileParam.

Input:

None.

Output:

IP : DINT
The IP address of the device.

SubnetMask : DINT
The Subnet mask of the device.

Gateway : DINT
The TCP/IP gateway the device uses.

DNS1 : DINT
The first Domain Name Server the device uses to look up symbolic names.

DNS2 : DINT
The second Domain Name Server the device uses to look up symbolic names.

Username : STRING
The user name the device uses in order to connect to the mobile network.

Password : STRING
The password the device uses in order to connect to the mobile network.

APN : STRING
The APN the device uses in order to connect to the mobile network.

Authenticate : INT
The PPP authentication type:
0 - None.
1 - PAP.
2 - CHAP.
3 - PAP/CHAP.

Declaration:

```
FUNCTION_BLOCK sockGetTCPIPParm;
VAR_OUTPUT
   // general TCP/IP parameters:
   IP           : DINT;
   SubnetMask   : DINT;
   Gateway      : DINT;
   DNS1         : DINT;
   DNS2         : DINT;

   // PPP parameters:
   Username     : STRING;        // username (max 33 char).
   Password     : STRING;        // password (max 33 char).

   // Dialup/mobile network parameters:
   APN          : STRING;        // APN (MAX 33 char)
   Authenticate : INT;           // Authentication type. 0=none, 1=PAP,
2=CHAP, 3=PAP/CHAP.
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   gprsParm  : sockGetTCPIPParm;
   gwParm    : sockGetGWParm;
   parmReset : BOOL := FALSE;
   CryptKey  : ARRAY[1..16] OF SINT;
END_VAR;

// Check mobile network and Gateway settings
gprsParm();
gwParm();
IF gprsParm.IP <> 0 OR gprsParm.SubnetMask <> 0 OR gprsParm.Gateway <> 0 OR
   gprsParm.DNS1 <> 0 OR gprsParm.DNS2 <> 0 OR gprsParm.Authenticate <> 0 OR
   strCompare(str1 := gprsParm.Username, str2 := "") <> 0 OR
   strCompare(str1 := gprsParm.Password, str2 := "") <> 0 OR
   strCompare(str1 := gprsParm.APN, str2 := "internet") <> 0 THEN
   // Set APN and keep the default values for the others
   sockSetTCPIPParm(APN := "internet");
   parmReset := TRUE;
END_IF;
IF NOT gwParm.GWEnabled OR gwParm.GWPort <> 5001 OR gwParm.CryptKey[1] <> 0 OR
   gwParm.CryptKey[2] <> 0 OR gwParm.CryptKey[3] <> 0 OR gwParm.CryptKey[4] <>
 0 OR
   gwParm.CryptKey[5] <> 0 OR gwParm.CryptKey[6] <> 0 OR gwParm.CryptKey[7] <>
```

```
 0 OR
   gwParm.CryptKey[8] <> 0 OR gwParm.CryptKey[9] <> 0 OR gwParm.CryptKey[10]
<> 0 OR
   gwParm.CryptKey[11] <> 0 OR gwParm.CryptKey[12] <> 0 OR gwParm.CryptKey[13]
 <> 0 OR
   gwParm.CryptKey[14] <> 0 OR gwParm.CryptKey[15] <> 0 OR gwParm.CryptKey[16]
 <> 0 OR
   strCompare(str1 := gwParm.GWIP, str2 := "gw.rtcu.dk") <> 0 OR
   strCompare(str1 := gwParm.GWKey, str2 := "AABBCCDD") <> 0 THEN
   // Clear encryption key
   CryptKey[1]  := 0;
   CryptKey[2]  := 0;
   CryptKey[3]  := 0;
   CryptKey[4]  := 0;
   CryptKey[5]  := 0;
   CryptKey[6]  := 0;
   CryptKey[7]  := 0;
   CryptKey[8]  := 0;
   CryptKey[9]  := 0;
   CryptKey[10] := 0;
   CryptKey[11] := 0;
   CryptKey[12] := 0;
   CryptKey[13] := 0;
   CryptKey[14] := 0;
   CryptKey[15] := 0;
   CryptKey[16] := 0;
   // Set Gateway parameters
   sockSetGWParm(GWEnabled := TRUE, GWIP := "gw.rtcu.dk", GWPort := 5001,
GWKey := "AABBCCDD", CryptKey := ADDR(CryptKey));
   parmReset := TRUE;
END_IF;
IF parmReset THEN
   // Reset device before the changes are used
   boardReset();
END_IF;

// Turn on power to the GSM module
gsmPower(power := TRUE);
gprsOpen();

BEGIN
   ...
END;
END_PROGRAM;
```

4.2.45 sos: System Object Storage (NX32L)

4.2.45.1 sos: System Object Storage

The System Object Storage(SOS) is used to store custom objects.
It supports five different object types: Booleans, integers, floating-point numbers, strings and binary objects.
The SOS should not be used for values that often change, but is suited for configuration settings and resources.
The custom objects can also be modified using the SOS Viewer.

• sosObjectDelete	Deletes a SOS object.
• sosBoolCreate	Creates a new boolean object.
• sosBoolSet	Updates the value of a boolean object.
• sosBoolGet	Retrieves the value for a boolean object.
• sosIntegerCreate	Creates a new integer object.
• sosIntegerSet	Updates the value of an integer object.
• sosIntegerGetDint	Retrieves the value for an integer object as a DINT.
• sosIntegerGetInt	Retrieves the value for an integer object as an INT.
• sosIntegerGetSint	Retrieves the value for an integer object as a SINT.
• sosFloatCreate	Creates a new floating-point object.
• sosFloatSet	Updates the value of a floating-point object.
• sosFloatGet	Retrieves the value for a floating-point object.
• sosStringCreate	Creates a new string object.
• sosStringSet	Updates the value of a string object.
• sosStringGet	Retrieves the value of a string object.
• sosDataCreate	Creates a new binary object.
• sosDataSet	Updates the value of a binary object.
• sosDataGet	Retrieves the value of a binary object.

Please note that when using the sosFloat functions, **math.inc** must be explicitly included by the application.

4.2.45.2 sosObjectDelete (Function)

```
Architecture:       NX32L
Device support:     All
Firmware version:   1.00.00
```

---

sosObjectDelete will delete the objects that matches the specified key.

Input:

key : STRING
The key for the object to delete. The wildcards ? and * are supported.

Returns: **INT**

0 - The object is deleted or did not exist.
-1 - Generic error.

Declaration:

```
FUNCTION sosObjectDelete : INT;
VAR_INPUT
```

```
   key   : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Delete the object with the key "config.password"
   sosObjectDelete(key := "config.password");
   ...
END;

END_PROGRAM;
```

4.2.45.3 sosBoolCreate (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.00.00

sosBoolCreate will create a new boolean object.
If the object already exists, it will be replaced.

Input:

key : STRING
The key for the object to create. May only contain a-z,A-Z,0-9, "." and "_" (maximum of 250 characters).

desc : STRING
The description for the object, to show in the IDE (maximum of 80 characters).

value : BOOL *default FALSE*
The value for the object.

Returns: INT

0	- The object is created.
-1	- Generic error.
-2	- Invalid parameter.

Declaration:

```
FUNCTION sosBoolCreate : INT;
VAR_INPUT
   key   : STRING;
   desc  : STRING;
   value : BOOL := FALSE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
```

```
   ena : BOOL;
END_VAR;
BEGIN
   ...
   // Create boolean object
   rc := sosBoolCreate(key := "config.enable", value:=ON, desc := "Enable the
configuration");
   DebugFmt(message:="sosBoolCreate=\1", v1:=rc);
   // Update boolean object
   rc:= sosBoolSet(key:="config.enable", value:=OFF);
   DebugFmt(message:="sosBoolSet=\1", v1:=rc);

   // Read boolean object
   rc:= sosBoolGet(key:="config.enable", value:=ena);
   DebugFmt(message:="sosBoolGet=\1 => \2", v1:=rc, v2:=ena);
   ...
END;

END_PROGRAM;
```

4.2.45.4 sosBoolSet (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.00.00

---

sosBoolSet will update the value of an existing boolean object.

Input:

key : STRING
The key for the object to update.

value : BOOL *default FALSE*
The new value for the object.

Returns: INT

0	- The object is updated.
-1	- Generic error.
-10	- Object not found.
-11	- Object has the wrong type.

Declaration:

```
FUNCTION sosBoolSet : INT;
VAR_INPUT
   key   : STRING;
   value : BOOL := FALSE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   ena : BOOL;
END_VAR;
BEGIN
   ...
```

```
   // Create boolean object
   rc := sosBoolCreate(key := "config.enable", value:=ON, desc := "Enable the
configuration");
   DebugFmt(message:="sosBoolCreate=\1", v1:=rc);
   // Update boolean object
   rc:= sosBoolSet(key:="config.enable", value:=OFF);
   DebugFmt(message:="sosBoolSet=\1", v1:=rc);

   // Read boolean object
   rc:= sosBoolGet(key:="config.enable", value:=ena);
   DebugFmt(message:="sosBoolGet=\1 => \2", v1:=rc, v2:=ena);
   ...
END;

END_PROGRAM;
```

4.2.45.5 sosBoolGet (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.00.00

---

sosBoolGet will retrieve the current value of a boolean object.

Input:

key : STRING
The key for the object to retrieve.

Output:

value : BOOL
The value of the object.

Returns: INT

0	- The value has been retrieved.
-1	- Generic error.
-10	- Object not found.
-11	- Object has the wrong type.
-12	- Content has invalid size.

Declaration:

```
FUNCTION sosBoolSet : INT;
VAR_INPUT
   key   : STRING;
   value : ACCESS BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc  : INT;
   ena : BOOL;
END_VAR;
BEGIN
   ...
   // Create boolean object
```

```
   rc := sosBoolCreate(key := "config.enable", value:=ON, desc := "Enable the
configuration");
   DebugFmt(message:="sosBoolCreate=\1", v1:=rc);
   // Update boolean object
   rc:= sosBoolSet(key:="config.enable", value:=OFF);
   DebugFmt(message:="sosBoolSet=\1", v1:=rc);

   // Read boolean object
   rc:= sosBoolGet(key:="config.enable", value:=ena);
   DebugFmt(message:="sosBoolGet=\1 => \2", v1:=rc, v2:=ena);
   ...
END;

END_PROGRAM;
```

4.2.45.6 sosIntegerCreate (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.00.00

---

sosIntegerCreate will create a new integer object.
If the object already exists, it will be replaced.

Input:

key : STRING
The key for the object to create. May only contain a-z,A-Z,0-9, "." and "_" (maximum of 250 characters).

desc : STRING
The description for the object, to show in the IDE (maximum of 80 characters).

value : DINT *default 0*
The initial value for the object. The value is not validated, making it possible to use it to e.g. determine unmodified values.

min_val : DINT *default -2147483648*
The minimum allowed value for the object. If both min_val and max_val are 0, there will be no limit.

max_val : DINT *default 2147483647*
The maximum allowed value for the object. If both min_val and max_val are 0, there will be no limit.

Returns: INT

0	- The object is created.
-1	- Generic error.
-2	- Invalid parameter.

Declaration:

```
FUNCTION sosIntegerCreate : INT;
VAR_INPUT
   key     : STRING;
   desc    : STRING;
   value   : DINT := 0;
   min_val : DINT := -2_147_483_648;
   max_val : DINT :=  2_147_483_647;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc      : INT;
   timeout : INT;
END_VAR;
BEGIN
   ...
   // Create integer object
   rc := sosIntegerCreate(key := "config.timeout", value:=600, desc :=
"Timeout for the connection");
   DebugFmt(message:="sosIntegerCreate=\1", v1:=rc);
   // Update integer object
   rc:= sosIntegerSet(key:="config.timeout", value:=6000);
   DebugFmt(message:="sosIntegerSet=\1", v1:=rc);

   // Read integer object
   rc:= sosIntegerGetInt(key:="config.timeout", value:=timeout);
   DebugFmt(message:="sosIntegerGetInt=\1 => \2", v1:=rc, v2:=timeout);
   ...
END;

END_PROGRAM;
```

4.2.45.7 sosIntegerSet (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.00.00

sosIntegerSet will update the value of an existing integer object.

Input:

key : STRING
The key for the object to update.

value : DINT *default 0*
The new value for the object.

Returns: INT

0	- The object is updated.
-1	- Generic error.
-2	- Invalid parameter.
-10	- Object not found.
-11	- Object has the wrong type.

Declaration:

```
FUNCTION sosIntegerSet : INT;
VAR_INPUT
   key   : STRING;
   value : DINT := 0;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc      : INT;
   timeout : INT;
END_VAR;
BEGIN
   ...
   // Create integer object
   rc := sosIntegerCreate(key := "config.timeout", value:=600, desc :=
"Timeout for the connection");
   DebugFmt(message:="sosIntegerCreate=\1", v1:=rc);
   // Update integer object
   rc:= sosIntegerSet(key:="config.timeout", value:=6000);
   DebugFmt(message:="sosIntegerSet=\1", v1:=rc);

   // Read integer object
   rc:= sosIntegerGetInt(key:="config.timeout", value:=timeout);
   DebugFmt(message:="sosIntegerGetInt=\1 => \2", v1:=rc, v2:=timeout);
   ...
END;

END_PROGRAM;
```

4.2.45.8 sosIntegerGetDint (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.00.00

sosIntegerGetDint will retrieve the current value of an integer object as a DINT.

Input:

key : STRING
The key for the object to retrieve.

Output:

value : DINT
The value of the object.

Returns: INT

0	- The value has been retrieved.
-1	- Generic error.
-10	- Object not found.
-11	- Object has the wrong type.
-12	- Content has invalid size.

Declaration:

```
FUNCTION sosIntegerGetDint : INT;
VAR_INPUT
   key   : STRING;
   value : ACCESS DINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc   : INT;
   baud : DINT;
END_VAR;
BEGIN
   ...
   // Create integer object
   rc := sosIntegerCreate(key := "config.baud", value:=115200, desc := "Baud
rate for the connection");
   DebugFmt(message:="sosIntegerCreate=\1", v1:=rc);
   // Update integer object
   rc:= sosIntegerSet(key:="config.baud", value:=6000);
   DebugFmt(message:="sosIntegerSet=\1", v1:=rc);

   // Read integer object
   rc:= sosIntegerGetDint(key:="config.baud", value:=baud);
   DebugFmt(message:="sosIntegerGetDint=\1 => \4", v1:=rc, v4:=baud);
   ...
END;

END_PROGRAM;
```

4.2.45.9 sosIntegerGetInt (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.00.00

sosIntegerGetInt will retrieve the current value of an integer object as an INT.

Input:

key : STRING
The key for the object to retrieve.

Output:

value : INT
The value of the object.

Returns: INT

0	- The value has been retrieved.
-1	- Generic error.
-10	- Object not found.
-11	- Object has the wrong type.
-12	- Content has invalid size.

Declaration:

```
FUNCTION sosIntegerGetInt : INT;
VAR_INPUT
   key   : STRING;
   value : ACCESS INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc      : INT;
   timeout : INT;
END_VAR;
BEGIN
   ...
   // Create integer object
   rc := sosIntegerCreate(key := "config.timeout", value:=600, desc :=
"Timeout for the connection");
   DebugFmt(message:="sosIntegerCreate=\1", v1:=rc);
   // Update integer object
   rc:= sosIntegerSet(key:="config.timeout", value:=6000);
   DebugFmt(message:="sosIntegerSet=\1", v1:=rc);

   // Read integer object
   rc:= sosIntegerGetInt(key:="config.timeout", value:=timeout);
   DebugFmt(message:="sosIntegerGetInt=\1 => \2", v1:=rc, v2:=timeout);
   ...
END;

END_PROGRAM;
```

4.2.45.10 sosIntegerGetSint (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.00.00

sosIntegerGetSint will retrieve the current value of an integer object as a SINT.

Input:

key : STRING
The key for the object to retrieve.

Output:

value : SINT
The value of the object.

Returns: INT

0	- The value has been retrieved.
-1	- Generic error.
-10	- Object not found.
-11	- Object has the wrong type.
-12	- Content has invalid size.

Declaration:

```
FUNCTION sosIntegerGetSint : SINT;
VAR_INPUT
   key   : STRING;
   value : ACCESS SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc    : INT;
   count : SINT;
END_VAR;
BEGIN
   ...
   // Create integer object
   rc := sosIntegerCreate(key := "config.count", value:=10, desc := "Number of
things");
   DebugFmt(message:="sosIntegerCreate=\1", v1:=rc);
   // Update integer object
   rc:= sosIntegerSet(key:="config.count", value:=9);
   DebugFmt(message:="sosIntegerSet=\1", v1:=rc);

   // Read integer object
   rc:= sosIntegerGetSint(key:="config.count", value:=count);
   DebugFmt(message:="sosIntegerGetSint=\1 => \2", v1:=rc, v2:=count);
   ...
END;

END_PROGRAM;
```

4.2.45.11 sosFloatCreate (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.00.00

sosFloatCreate will create a new FLOAT object.
If the object already exists, it will be replaced.
Please note that when using this function, **math.inc** must be explicitly included by the application.

Input:

key : STRING
The key for the object to create. May only contain a-z,A-Z,0-9, "." and "_" (maximum of 250 characters).

desc : STRING
The description for the object, to show in the IDE (maximum of 80 characters).

value : FLOAT *default 0*
The initial value for the object. The value is not validated, making it possible to use it to e.g. determine unmodified values.

min_val : FLOAT *default 0*
The minimum allowed value for the object. If both min_val and max_val are 0, there will be no limit. NaN is not a valid value for this parameter.

max_val : FLOAT *default 0*
The maximum allowed value for the object. If both min_val and max_val are 0, there will be no limit. NaN is not a valid value for this parameter.

Returns: INT

0 - The object is created.

-1 - Generic error.
-2 - Invalid parameter.

Declaration:

```
FUNCTION sosFloatCreate : INT;
VAR_INPUT
   key     : STRING;
   desc    : STRING;
   value   : FLOAT := 0.0;
   min_val : FLOAT := 0.0;
   max_val : FLOAT := 0.0;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;
VAR
   rc   : INT;
   temp : FLOAT;
END_VAR;
BEGIN
   ...
   // Create float object
   rc := sosFloatCreate(key := "config.max_temperature", value:=24.5, desc :=
"Largest temperature measured");
   DebugFmt(message:="sosFloatCreate=\1", v1:=rc);
   // Update float object
   rc:= sosFloatSet(key:="config.max_temperature", value:=26.7);
   DebugFmt(message:="sosFloatSet=\1", v1:=rc);

   // Read float object
   rc:= sosFloatGet(key:="config.max_temperature", value:=temp);
   DebugFmt(message:="sosFloatGet=\1 => " + floatToStr(v:=temp), v1:=rc);
   ...
END;

END_PROGRAM;
```

4.2.45.12 sosFloatSet (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.00.00

---

sosFloatSet will update the value of an existing FLOAT object.
Please note that when using this function, **math.inc** must be explicitly included by the application.

Input:

key : STRING
The key for the object to update.

value : FLOAT *default 0*
The new value for the object.

Returns: INT

0	- The object is updated.
-1	- Generic error.
-2	- Invalid parameter.
-10	- Object not found.
-11	- Object has the wrong type.

Declaration:

```
FUNCTION sosFloatSet : INT;
VAR_INPUT
   key   : STRING;
   value : FLOAT := 0.0;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;
VAR
   rc   : INT;
   temp : FLOAT;
END_VAR;
BEGIN
   ...
   // Create float object
   rc := sosFloatCreate(key := "config.max_temperature", value:=24.5, desc :=
"Largest temperature measured");
   DebugFmt(message:="sosFloatCreate=\1", v1:=rc);
   // Update float object
   rc:= sosFloatSet(key:="config.max_temperature", value:=26.7);
   DebugFmt(message:="sosFloatSet=\1", v1:=rc);

   // Read float object
   rc:= sosFloatGet(key:="config.max_temperature", value:=temp);
   DebugFmt(message:="sosFloatGet=\1 => " + floatToStr(v:=temp), v1:=rc);
   ...
END;

END_PROGRAM;
```

4.2.45.13 sosFloatGet (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.00.00

---

sosFloatGet will retrieve the current value of a FLOAT object.
Please note that when using this function, **math.inc** must be explicitly included by the application.

Input:

key : STRING
The key for the object to retrieve.

Output:

value : FLOAT
The value of the object.

Returns: INT

```
   0   - The value has been retrieved.
  -1   - Generic error.
 -10   - Object not found.
 -11   - Object has the wrong type.
 -12   - Content has invalid size.
```

Declaration:

```
FUNCTION sosFloatGet : INT;
VAR_INPUT
   key   : STRING;
   value : ACCESS FLOAT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;
VAR
   rc   : INT;
   temp : FLOAT;
END_VAR;
BEGIN
   ...
   // Create float object
   rc := sosFloatCreate(key := "config.max_temperature", value:=24.5, desc :=
"Largest temperature measured");
   DebugFmt(message:="sosFloatCreate=\1", v1:=rc);
   // Update float object
   rc:= sosFloatSet(key:="config.max_temperature", value:=26.7);
   DebugFmt(message:="sosFloatSet=\1", v1:=rc);

   // Read float object
   rc:= sosFloatGet(key:="config.max_temperature", value:=temp);
   DebugFmt(message:="sosFloatGet=\1 => " + floatToStr(v:=temp), v1:=rc);
   ...
END;

END_PROGRAM;
```

4.2.45.14 sosDoubleCreate (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.52.00

---

sosDoubleCreate will create a new DOUBLE object.
If the object already exists, it will be replaced.

Input:

key : STRING
The key for the object to create. May only contain a-z,A-Z,0-9, "." and "_" (maximum of 250 characters).

desc : STRING
The description for the object, to show in the IDE (maximum of 80 characters).

value : DOUBLE *default 0*
The initial value for the object. The value is not validated, making it possible to use it to e.g. determine unmodified values.

min_val : DOUBLE *default 0*
The minimum allowed value for the object. If both min_val and max_val are 0, there will be no limit. NaN is not a valid value for this parameter.

max_val : DOUBLE *default 0*
The maximum allowed value for the object. If both min_val and max_val are 0, there will be no limit. NaN is not a valid value for this parameter.

Returns: INT

0 - The object is created.
-1 - Generic error.
-2 - Invalid parameter.

Declaration:

```
FUNCTION sosDoubleCreate : INT;
VAR_INPUT
   key     : STRING;
   desc    : STRING;
   value   : DOUBLE := 0.0;
   min_val : DOUBLE := 0.0;
   max_val : DOUBLE := 0.0;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;
VAR
   rc   : INT;
   temp : DOUBLE;
END_VAR;
BEGIN
   ...
   // Create double object
   rc := sosDoubleCreate(key := "config.max_temperature", value:=24.5, desc :=
"Largest temperature measured");
   DebugFmt(message:="sosDoubleCreate=\1", v1:=rc);
   // Update double object
   rc:= sosDoubleSet(key:="config.max_temperature", value:=26.7);
   DebugFmt(message:="sosDoubleSet=\1", v1:=rc);

   // Read double object
   rc:= sosDoubleGet(key:="config.max_temperature", value:=temp);
   DebugFmt(message:="sosDoubleGet=\1 => " + doubleToStr(v:=temp), v1:=rc);
   ...
END;

END_PROGRAM;
```

4.2.45.15 sosDoubleSet (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.52.00

---

sosDoubleSet will update the value of an existing DOUBLE object.

Input:

key : STRING
The key for the object to update.

value : DOUBLE *default 0*
The new value for the object.

Returns: INT

0	- The object is updated.
-1	- Generic error.
-2	- Invalid parameter.
-10	- Object not found.
-11	- Object has the wrong type.

Declaration:

```
FUNCTION sosDoubleSet : INT;
VAR_INPUT
   key   : STRING;
   value : DOUBLE := 0.0;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;
VAR
   rc   : INT;
   temp : DOUBLE;
END_VAR;
BEGIN
   ...
   // Create double object
   rc := sosDoubleCreate(key := "config.max_temperature", value:=24.5, desc :=
"Largest temperature measured");
   DebugFmt(message:="sosDoubleCreate=\1", v1:=rc);
   // Update double object
   rc:= sosDoubleSet(key:="config.max_temperature", value:=26.7);
   DebugFmt(message:="sosDoubleSet=\1", v1:=rc);

   // Read double object
   rc:= sosDoubleGet(key:="config.max_temperature", value:=temp);
   DebugFmt(message:="sosDoubleGet=\1 => " + doubleToStr(v:=temp), v1:=rc);
   ...
END;

END_PROGRAM;
```

4.2.45.16 sosDoubleGet (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.52.00

---

sosDoubleGet will retrieve the current value of a DOUBLE object.

Input:

key : STRING
The key for the object to retrieve.

Output:

value : DOUBLE
The value of the object.

Returns: INT

0	- The value has been retrieved.
-1	- Generic error.
-10	- Object not found.
-11	- Object has the wrong type.
-12	- Content has invalid size.

Declaration:

```
FUNCTION sosDoubleGet : INT;
VAR_INPUT
   key   : STRING;
   value : ACCESS DOUBLE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
INCLUDE math.inc

PROGRAM test;
VAR
   rc   : INT;
   temp : DOUBLE;
END_VAR;
BEGIN
   ...
   // Create double object
   rc := sosDoubleCreate(key := "config.max_temperature", value:=24.5, desc :=
"Largest temperature measured");
   DebugFmt(message:="sosDoubleCreate=\1", v1:=rc);
   // Update double object
   rc:= sosDoubleSet(key:="config.max_temperature", value:=26.7);
   DebugFmt(message:="sosDoubleSet=\1", v1:=rc);

   // Read double object
   rc:= sosDoubleGet(key:="config.max_temperature", value:=temp);
   DebugFmt(message:="sosDoubleGet=\1 => " + doubleToStr(v:=temp), v1:=rc);
   ...
END;

END_PROGRAM;
```

4.2.45.17 sosStringCreate (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.00.00

---

sosStringCreate will create a new string object.
If the object already exists, it will be replaced.

Input:

key : STRING
The key for the object to create. May only contain a-z,A-Z,0-9, "." and "_" (maximum of 250 characters).

desc : STRING
The description for the object, to show in the IDE (maximum of 80 characters).

str : STRING
The value for the object.

max_len : INT(1..16384) ***default 254***
The maximum allowed length for the object.

Returns: INT

0 - The object is created.
-1 - Generic error.
-2 - Invalid parameter.

Declaration:

```
FUNCTION sosStringCreate : INT;
VAR_INPUT
   key     : STRING;
   desc    : STRING;
   str     : STRING;
   max_len : INT :=  254;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc   : INT;
   name : STRING;
END_VAR;
BEGIN
   ...
   // Create string object
   rc := sosStringCreate(key := "config.name", str:="Test configuration", desc
:= "Name of the configuration", max_len := 50);
   DebugFmt(message:="sosStringCreate=\1", v1:=rc);
   // Update string object
   rc:= sosStringSet(key:="config.name", str:="New name");
   DebugFmt(message:="sosStringSet=\1", v1:=rc);

   // Read string object
```

```
   rc:= sosStringGet(key:="config.name", str:=name);
   DebugFmt(message:="sosStringGet=\1 => " + name, v1:=rc);
   ...
END;

END_PROGRAM;
```

4.2.45.18 sosStringSet (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.00.00

---

sosStringSet will update the value of an existing string object.

Input:

key : STRING
The key for the object to update.

str : STRING
The new value for the object.

Returns: INT

0	- The object is updated.
-1	- Generic error.
-2	- Invalid parameter.
-10	- Object not found.
-11	- Object has the wrong type.

Declaration:

```
FUNCTION sosStringSet : INT;
VAR_INPUT
   key   : STRING;
   str   : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc   : INT;
   name : STRING;
END_VAR;
BEGIN
   ...
   // Create string object
   rc := sosStringCreate(key := "config.name", str:="Test configuration", desc
:= "Name of the configuration", max_len := 50);
   DebugFmt(message:="sosStringCreate=\1", v1:=rc);
   // Update string object
   rc:= sosStringSet(key:="config.name", str:="New name");
   DebugFmt(message:="sosStringSet=\1", v1:=rc);

   // Read string object
   rc:= sosStringGet(key:="config.name", str:=name);
   DebugFmt(message:="sosStringGet=\1 => " + name, v1:=rc);
```

```
   ...
END;

END_PROGRAM;
```

4.2.45.19 sosStringGet (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.00.00

---

sosStringGet will retrieve the current value of a string object.

Input:

key : STRING
The key for the object to retrieve.

Output:

str : STRING
The value of the object.

Returns: INT

0	- The value has been retrieved.
-1	- Generic error.
-10	- Object not found.
-11	- Object has the wrong type.

Declaration:

```
FUNCTION sosStringGet : INT;
VAR_INPUT
   key   : STRING;
   str   : ACCESS STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc   : INT;
   name : STRING;
END_VAR;
BEGIN
   ...
   // Create string object
   rc := sosStringCreate(key := "config.name", str:="Test configuration", desc
:= "Name of the configuration", max_len := 50);
   DebugFmt(message:="sosStringCreate=\1", v1:=rc);
   // Update string object
   rc:= sosStringSet(key:="config.name", str:="New name");
   DebugFmt(message:="sosStringSet=\1", v1:=rc);

   // Read string object
   rc:= sosStringGet(key:="config.name", str:=name);
   DebugFmt(message:="sosStringGet=\1 => " + name, v1:=rc);
   ...
END;
```

```
END_PROGRAM;
```

4.2.45.20 sosDataCreate (Function)

```
Architecture:       NX32L
Device support:     All
Firmware version:   1.00.00
```

---

sosDataCreate will create a new object for storing binary data.
If the object already exists, it will be replaced.

Input:

key : STRING
The key for the object to create. May only contain a-z,A-Z,0-9, "." and "_" (maximum of 250 characters).

desc : STRING
The description for the object, to show in the IDE (maximum of 80 characters).

value : PTR
Address of a buffer containing the data to store.

len : INT (0..32767)
The length of the data in the buffer.

max_len : INT (0..32767) ***default 100***
The maximum allowed length for the object.

Returns: INT

```
 0   - The object is created.
-1   - Generic error.
-2   - Invalid parameter.
```

Declaration:

```
FUNCTION sosDataCreate : INT;
VAR_INPUT
   key     : STRING;
   desc    : STRING;
   value   : PTR;
   len     : INT;
   max_len : INT :=  100;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc      : INT;
   len     : INT;
   arr     : ARRAY[1..100] OF SINT;
   rx      : ARRAY[1..50] OF SINT;
END_VAR;
BEGIN
   ...
```

```
   // Create binary object
   rc := sosDataCreate(key := "config.data", value:=ADDR(arr), len := 10, desc
:= "Data for the connection");
   DebugFmt(message:="sosDataCreate=\1", v1:=rc);
   // Update binary object
   rc:= sosDataSet(key:="config.data", value:=ADDR(arr), len := 20);
   DebugFmt(message:="sosDataSet=\1", v1:=rc);

   // Read binary object
   rc:= sosDataGet(key:="config.data", value:=ADDR(rx), len := len, max_len :=
50);
   DebugFmt(message:="sosDataGet=\1 => Len = \2", v1:=rc, v2:=len);
   ...
END;

END_PROGRAM;
```

4.2.45.21 sosDataSet (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.00.00

---

sosDataSet will update the value of an existing binary object.

Input:

key : STRING
The key for the object to update.

value : PTR
Address of a buffer containing the new value for the object.

len : INT (0..32767)
The length of the data in the buffer.

Returns: INT

0	- The object is updated.
-1	- Generic error.
-2	- Invalid parameter.
-10	- Object not found.
-11	- Object has the wrong type.

Declaration:

```
FUNCTION sosDataSet : INT;
VAR_INPUT
   key   : STRING;
   value : PTR;
   len   : INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc      : INT;
   len     : INT;
```

```
   arr      : ARRAY[1..100] OF SINT;
   rx       : ARRAY[1..50] OF SINT;
END_VAR;
BEGIN
   ...
   // Create binary object
   rc := sosDataCreate(key := "config.data", value:=ADDR(arr), len := 10, desc
:= "Data for the connection");
   DebugFmt(message:="sosDataCreate=\1", v1:=rc);
   // Update binary object
   rc:= sosDataSet(key:="config.data", value:=ADDR(arr), len := 20);
   DebugFmt(message:="sosDataSet=\1", v1:=rc);

   // Read binary object
   rc:= sosDataGet(key:="config.data", value:=ADDR(rx), len := len, max_len :=
50);
   DebugFmt(message:="sosDataGet=\1 => Len = \2", v1:=rc, v2:=len);
   ...
END;

END_PROGRAM;
```

4.2.45.22 sosDataGet (Function)

Architecture: NX32L
Device support: All
Firmware version: 1.00.00

---

sosDataGet will retrieve the current value of a binary object.

Input:

key : STRING
The key for the object to retrieve.

value : PTR
Address of a buffer to place the data into.

max_len : INT
The size of the buffer. If smaller than the size of the data, the data will be truncated to fit.

Output:

len : INT
The actual length of the available data.

Returns: INT

0	- The value has been retrieved.
-1	- Generic error.
-2	- Invalid parameter.
-10	- Object not found.
-11	- Object has the wrong type.

Declaration:

```
FUNCTION sosDataGet : INT;
VAR_INPUT
   key      : STRING;
   value    : PTR;
   max_len  : INT;
```

```
   len      : ACCESS INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   rc       : INT;
   len      : INT;
   arr      : ARRAY[1..100] OF SINT;
   rx       : ARRAY[1..50] OF SINT;
END_VAR;
BEGIN
   ...
   // Create binary object
   rc := sosDataCreate(key := "config.data", value:=ADDR(arr), len := 10, desc
:= "Data for the connection");
   DebugFmt(message:="sosDataCreate=\1", v1:=rc);
   // Update binary object
   rc:= sosDataSet(key:="config.data", value:=ADDR(arr), len := 20);
   DebugFmt(message:="sosDataSet=\1", v1:=rc);

   // Read binary object
   rc:= sosDataGet(key:="config.data", value:=ADDR(rx), len := len, max_len :=
50);
   DebugFmt(message:="sosDataGet=\1 => Len = \2", v1:=rc, v2:=len);
   ...
END;

END_PROGRAM;
```

4.2.46 udp: UDP/IP socket functions

4.2.46.1 udp: UDP/IP socket functions

The UDP socket functions are used for communicating over the internet via IP-networks. The udp() functions use UDP/IP.
For TCP/IP support, please see TCP/IP socket functions.

Please see the example program (shown for each function):

- udpStartListen — Starts to listen for data
- udpStopListen — Stop listening for data
- udpReceive — Receive data
- udpSend — Send data

For NX32L based devices it is recommended to use the Advanced so socket API.

IP address format:
In all places where an IP address aaa.bbb.ccc.ddd is stored in a DINT, the following format is used:

ddd is stored in bit 24..31.
ccc is stored in bit 16..23.
bbb is stored in bit 8..15.
aaa is stored in bit 0..7.

4.2.46.2 udpStartListen (Function)

```
Architecture:       X32 / NX32 / NX32L
Device support:     All
Firmware version:   1.00 / 1.00.00
```

This starts to listen for data received as a connectionless UDP packet on the specified local port. The data will then be received with the updReceive function block.

Note: many GSM networks are using a firewall so that inbound connections to the RTCU are blocked when using the mobile network interface.

Input:

port : DINT
Port number listening on.

rip : DINT
Remote IP address accepted. 0 = all IP addresses accepted (default).

iface : SINT *(default 0)*
The network interface to listen on. 0 = all networks (default), 1 = Mobile network, 2 = LAN network, etc. (See Network)

Returns: SINT

ID for the connection or 0 if an error occurred (out of resources).

Declaration:

```
FUNCTION udpStartListen : SINT;
VAR_INPUT
   port  : DINT;      // Port number to listen on
   rip   : DINT :=0;  // Accept from this remote IP only. 0=all.
   iface : SINT :=0;  // Interface to listen on
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM udp_server;
VAR
   fbReceive : udpReceive;            // Receive data on a UDP socket
   port      : INT := 2500;           // port to receive data on
   udp_id    : SINT;                  // listener ID
   host      : STRING;                // translated IP
   line      : STRING;                // ASCII string to send
   data      : ARRAY[1..255] OF SINT; // received data buffer
   rc        : INT;                   // return code of command
END_VAR;

gsmPower(power := ON);
gprsOpen();

// Wait for the cellular connection to be established
WHILE NOT gprsConnected() DO
   DebugMsg(message := "Waiting for cellular connection");
   Sleep(delay := 2500);
END_WHILE;

// Listen for incoming connection on a UDP port
udp_id := udpStartListen(port := port, rip := 0);
BEGIN
   // update udp received data
   fbReceive(id := udp_id, data := ADDR(data), maxsize := SIZEOF(data));

   IF fbReceive.ready THEN
      host := sockIPToName(ip := fbReceive.ip);
      DebugFmt(message := "Received '\1' bytes from '" + host + "'", v1 :=
fbReceive.size);
      line := strFromMemory(src := ADDR(data), len := fbReceive.size);
      DebugMsg(message := "Message : " + line);
   END_IF;
END;
END_PROGRAM;
```

4.2.46.3 udpStopListen (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This stops to listen for data on the specified connection ID.

Input:

id : SINT
ID for the connection.

Returns: INT

0 if successful, -1 indicates that the passed ID is illegal.

Declaration:

```
FUNCTION udpStopListen : INT;
VAR_INPUT
   id : SINT;  // ID of the connection.
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   server_active : BOOL; | listen for incoming when active
END_VAR;

PROGRAM udp_server;
VAR
   port   : INT := 2500;  // port to receive data on
   udp_id : SINT;         // listener ID
   rip    : DINT := 0;    // remote IP to receive data from 0=ALL
   status : RF_TRIG ;     // Declare an instance of the RF_TRIG
END_VAR;

gsmPower(power := ON);
gprsOpen();

BEGIN
   // start / stop based on input
   status(trig := server_active);
   // detect start/stop/running
   IF status.rq THEN
      // Listen for incoming connection on a UDP port
      udp_id := udpStartListen(port := port, rip := rip);
   ELSIF status.fq THEN
      // Terminate listening on a UDP port
      udpStopListen(id := udp_id);
   END_IF;
END;
END_PROGRAM;
```

4.2.46.4 udpSend (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This sends data as a connectionless UDP packet.

Input:

ip : DINT
IP address of the peer node (receiver of the data).

port : DINT
Port number to send the data to on the peer node.

data : PTR
Address of the buffer that contains the data to be sent.

size : INT
Number of bytes to send from the buffer.
Maximum size is 1540 bytes.

iface : SINT *(default 0)*
The network interface to use. 0 = Default network, 1 = Mobile network, 2 = LAN network, etc.
(See Network)
Note: For backwards compatibility the Mobile network is used as default network.

Returns: INT

Number of bytes that were sent on the socket, or:
0 - Communication problem. No data sent.
-1 - Illegal parameters.
-2 - Memory allocation error.

Declaration:

```
FUNCTION udpSend : INT;
VAR_INPUT
   ip    : DINT;     // IP address to send data to.
   port  : DINT;     // Port number to send data to
   data  : PTR;      // Address of the buffer to send
   size  : INT;      // Number of bytes to send
   iface : SINT;     // Interface to send on
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   send_message : BOOL R_EDGE; | Sends a message on rising edge
END_VAR;

PROGRAM udp_client;
VAR
   host  : STRING := "my.domain.com";   // host to retrieve data
   line  : STRING;                  // ASCII string to send
   data  : ARRAY[1..1540] OF SINT;  // send data buffer
   rip   : DINT;                    // remote ip to receive data
   rc    : INT;                     // return code of command
   port  : INT := 2500;             // remote host listener port
END_VAR;

   gsmPower(power := ON);
   gprsOpen();

   DebugMsg(message := "Ready to send message when input is activated.");
BEGIN
   IF send_message THEN
      // Return IP address (0 if none)
      rip := sockIPFromName(str := host);

      line := "Sending a message through UDP";
      // Copy contents of a string to memory
      strToMemory(dst := ADDR(data), str := line, len := strLen(str := line));
      // Send data on a UDP socket
      rc := udpSend(ip := rip, port := port, data := ADDR(data), size :=
strLen(str := line));
      IF rc > 0 THEN
         DebugFmt(message := "sent \1 Byte(s) of data.", v1 := rc);
```

```
      ELSE
         DebugFmt(message := "failed to send data error code '\1'", v1 := rc);
      END_IF;
   END_IF;
END;

END_PROGRAM;
```

4.2.46.5 udpReceive (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This receives data as a connectionless UDP packet from the specified connection. This function block will indicate when data is ready and placed into the receive buffer.

Input:

id : SINT
ID for the connection as returned by udpStartListen().

data : PTR
Address of the buffer that contains the received data.

maxsize : INT
Maximum number of bytes that can be received (size of "data").
The maximum size of any UDP package is 1540 bytes.
If a data package larger than "maxsize" is received, the data will be truncated but still delivered.

Output:

ready : BOOL
True if data has been received.

size : INT
Number of bytes received (if "ready" is true).

ip : INT
IP address of sender (if "ready" is true).

port : DINT
IP port of the sender (if "ready" is true).

iface : SINT
The network interface where the data was received (if "ready" is true). 1 = Mobile network, 2 = LAN network, etc. (See Network).

Declaration:

```
FUNCTION_BLOCK udpReceive;
VAR_INPUT
   id      : SINT; // ID of the connection.
   data    : PTR;  // Address of receive buffer
   maxsize : INT;  // Maximum number of bytes we can receive (size of 'data')
END_VAR;
VAR_OUTPUT
   ready   : BOOL; // data is ready
   iface   : SINT; // The network interface the data was received on.
   size    : INT;  // Number of bytes received
   ip      : DINT; // IP-address of sender.
```

```
   port    : DINT; // IP-port of the sender
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM udp_server;
VAR
   fbReceive : udpReceive;            // Receive data on a UDP socket
   port      : INT := 2500;           // port to receive data on
   udp_id    : SINT;                  // listener ID
   host      : STRING;                // translated IP
   line      : STRING;                // ASCII string to send
   data      : ARRAY[1..255] OF SINT; // received data buffer
   rc        : INT;                   // return code of command
END_VAR;

gsmPower(power := ON);
gprsOpen();

// Wait for the cellular connection to be established
WHILE NOT gprsConnected() DO
   DebugMsg(message := "Waiting for cellular connection");
   Sleep(delay := 2500);
END_WHILE;

// Listen for incoming connection on a UDP port
udp_id := udpStartListen(port := port, rip := 0);
BEGIN
   // update udp received data
   fbReceive(id := udp_id, data := ADDR(data), maxsize := SIZEOF(data));

   IF fbReceive.ready THEN
      host := sockIPToName(ip := fbReceive.ip);
      DebugFmt(message := "Received '\1' bytes from '" + host + "'", v1 :=
fbReceive.size);
      line := strFromMemory(src := ADDR(data), len := fbReceive.size);
      DebugMsg(message := "Message : " + line);
   END_IF;
END;
END_PROGRAM;
```

4.2.47 usb: USB host functions (NX32L)

4.2.47.1 usb: USB host functions

USB host functions are used to manage devices connected to the USB host port.

- usbHostEnable — Enable/Disable USB host port.
- usbHostEnumerate — Enumerate connected devices.
- usbHostGetInfo — Get information about a connected device.
- usbHostGetSerialPort — Get the port number for a Serial port.
- usbHostGetCameraPort — Get the port number for an USB camera.

Device classes supported:
USB hub, Serial port, USB Cameras, Mass storage and Ethernet.

4.2.47.2 usbHostEnable (Function)

```
Architecture:       NX32L
Device support:     NX-200, NX-400
Firmware version:   1.00.00
```

The usbHostEnable function enable/disable support for USB devices. When disabled the USB power is removed from the USB host port.
Please consult the technical manual for details on how many USB host ports a specific device offers.

The USB host is disabled as default, and must be enabled before devices can be discovered using usbHostEnumerate.

Input:

port : SINT (1 .. 127)
The USB host port to enable/disable. (1=usb1, 2=usb2, etc.)

enable : BOOL (false/true)
TRUE: Enables USB host support.
FALSE: Disables USB host support.

Returns: INT

1 - Success.
0 - The function is not supported.
-1 - The USB host port is not present.

Declaration:

```
FUNCTION usbHostEnable : INT;
VAR_INPUT
   port   : SINT;
   enable : BOOL;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   doEnable : BOOL; | enable/disable USB host
END_VAR;
```

```
VAR_OUTPUT
   isEnabled : BOOL := TRUE; | is USB host enabled
END_VAR;

PROGRAM test;
BEGIN
   IF doEnable XOR isEnabled THEN
      // Enable/Disable USB host
      usbHostEnable(port:=1,enable:=doEnable);
      isEnabled := doEnable;
   END_IF;
END;
END_PROGRAM;
```

4.2.47.3 usbHostEnumerate (Function)

Architecture: NX32L
Device support: NX-200, NX-400
Firmware version: 1.00.00

The function is used to retrieve information about devices connected to the USB host port. Only devices connected to an enabled port can be discovered. See usbHostEnable for details.

Input:

port : SINT (0..1, *default 0*)
The USB host port to enumerate. (0=All ports, 1=usb1, etc.)

index : INT (1..32767)
The index of the USB device to query.

Output:

device : STRING
This is the system device name, which can be used to identify the connected device.

type : INT
The type of device:

7	Camera.
4	Ethernet module
3	Mass storage device
2	Serial port.
1	USB hub.
-1	Unknown device.

Returns: INT

1	- Success.
0	- The function is not supported.
-1	- The device is not found or the USB host port is not present.(See usbHostEnable)

Declaration:

```
FUNCTION usbHostEnumerate : INT;
VAR_INPUT
   port   : SINT;
   index  : INT;
   device : ACCESS STRING;
   type   : ACCESS INT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   doScan   : BOOL R_EDGE; | Scan for connected USB devices
END_VAR;

PROGRAM example;
VAR
   device   : STRING;
   type     : INT;
   index    : INT;
   rc       : INT;
END_VAR;
// Enable USB host port
usbHostEnable(port:=1,enable:=TRUE);

BEGIN
   IF doScan THEN
      DebugMsg(message := "-- USB devices --");
      FOR index := 1 TO 32 DO
         // Get information about USB device
         rc := usbHostEnumerate(index := index, device := device, type :=
type);
         IF rc = 1 THEN
            DebugFmt(message := "\1 : Type = \2, Device = '" + device + "'",
v1 := index, v2 := type);
         ELSE
            EXIT;
         END_IF;
      END_FOR;
   END_IF;
END;
END_PROGRAM;
```

4.2.47.4 usbHostGetInfo (Functionblock)

Architecture: NX32L
Device support: NX-200, NX-400
Firmware version: 2.00.00

The function is used to retrieve information about a device connected to the USB host port. Only devices connected to an enabled port can be discovered. See usbHostEnable for details. Composite devices will report the same information for all the contained interfaces.

Input:

device : STRING
The system device name of the device, retrieved using the usbHostEnumerate function.

Output:

status : INT
The type of device:

1	Success
0	Not supported
-1	Device not found

VID: UINT
Vendor ID of the device.

PID: UINT
Product ID of the device.

manufacturer: STRING
The manufacturer name. An empty string if it is not set in the device

product: STRING
The product name. An empty string if it is not set in the device

version: STRING
The version number of the device. An empty string if it is not set in the device

serial: STRING
The serial number of the device. An empty string if it is not set in the device

Declaration:

```
FUNCTION_BLOCK usbHostGetInfo;
VAR_INPUT
   device  : STRING;
END_VAR;
VAR_OUTPUT
   status  : INT;
   VID     : UINT;
   PID     : UINT;
   manufacturer : STRING;
   product : STRING;
   version : STRING;
   serial  : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   doScan   : BOOL R_EDGE; | Scan for connected USB devices
END_VAR;

PROGRAM example;
VAR
   device   : STRING;
   type     : INT;
   index    : INT;
   rc       : INT;
   usbInfo  : usbHostGetInfo;
END_VAR;
// Enable USB host port
usbHostEnable(port:=1,enable:=TRUE);

BEGIN
   IF doScan THEN
      DebugMsg(message := "-- USB devices --");
      FOR index := 1 TO 32 DO
         // Get information about USB device
         rc := usbHostEnumerate(index := index, device := device, type :=
type);
         IF rc = 1 THEN
            DebugFmt(message := "\1 : Type = \2, Device = '" + device + "'",
v1 := index, v2 := type);
            usbInfo(device:=device);
            IF usbInfo.status > 0 THEN
```

```
                DebugFmt(message := "   ID : " + intToHex(v:=usbInfo.VID) + ":"
+ intToHex(v:=usbInfo.PID));
                DebugMsg(message := "   Man: " + usbInfo.manufacturer);
                DebugMsg(message := "   Prd: " + usbInfo.product);
                DebugMsg(message := "   Ver: " + usbInfo.version);
                DebugMsg(message := "   Ser: " + usbInfo.serial);
             END_IF;
          ELSE
             EXIT;
          END_IF;
       END_FOR;
    END_IF;
END;
END_PROGRAM;
```

4.2.47.5 usbHostGetSerialPort (Function)

Architecture: NX32L
Device support: NX-200, NX-400
Firmware version: 1.00.00

This returns the serial port number which can be use with serOpen to open a serial connection using an USB serial device.

Input:

device : STRING
The system device name of the USB serial port device retrieved using the usbHostEnumerate function.

Returns: SINT

```
 >0   - The serial port number used to open a connection with.
  0   - The function is not supported.
 -1   - The device is not a serial port or not found
```

Declaration:

```
FUNCTION usbHostGetSerialPort : SINT;
VAR_INPUT
   device : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
VAR_INPUT
   doOpen   : BOOL;           | open first found USB serial port
END_VAR;
VAR_OUTPUT
   isOpen   : BOOL := FALSE;  | is serial port open
   failed   : BOOL;           | failed to open port
END_VAR;

FUNCTION GetPortNum : SINT;
VAR
   device   : STRING;         // System name of USB device
   type     : INT := -1;      // Type of USB device
   index    : INT;
   port     : SINT;
   rc       : INT;
```

```
END_VAR;
GetPortNum := -1; // Not found
FOR index := 1 TO 32 DO
   rc := usbHostEnumerate(index := index, device := device, type := type);
   IF ((rc <= 0) OR (type = 2)) THEN
      EXIT;
   END_IF;
END_FOR;
IF type <> 2 THEN
   DebugMsg(message := "Failed to find any USB serial devices.");
   RETURN;
END_IF;
DebugFmt(message := "Found '" + device + "' at index \1", v1 := index);
port := usbHostGetSerialPort(device := device);
IF port <= 0 THEN
   DebugFmt(message := "Failed to get serial port number (code \1)", v1 :=
rc);
END_IF;
GetPortNum := port;
END_FUNCTION;

PROGRAM example;
VAR
   portNum  : SINT := -1;     // Select which port to use for the
communication
   timer    : TON;
   rc       : INT;
END_VAR;
// Enable USB host port
usbHostEnable(port:=1,enable:=TRUE);

BEGIN
   IF doOpen XOR isOpen THEN
      IF portNum >= 0 THEN
         serClose(port:=portNum);
         DebugFmt(message := "Serial port \1 was closed", v1 := portNum);
         portNum := -1;
      END_IF;
      IF doOpen THEN
         portNum := GetPortNum();
         IF portNum >= 0 THEN
            rc := serOpen(port := portNum);
            IF rc = 0 THEN
               DebugFmt(message := "Serial port \1 is open", v1 := portNum);
            ELSE
               DebugFmt(message := "Failed to open serial port \1 (\2)", v1 :=
portNum, v2 := rc);
               portNum := -1;
            END_IF;
         END_IF;
         failed := (portNum < 0);
         timer.pt := 0;
      END_IF;
      isOpen := doOpen;
   END_IF;
   timer(trig := (isOpen AND NOT failed));
   IF timer.q THEN
      serSendString(port := portNum, str := "Hello$N");
      timer(trig := FALSE, pt := 5000); // reset timer
   END_IF;
END;
END_PROGRAM;
```

4.2.47.6 usbHostGetCameraPort (Function)

Architecture: NX32L
Device support: NX-200, NX-400
Firmware version: 1.00.00

This returns the port number of the camera which can be used with camOpen to open a connection to an USB camera.

Input:

device : STRING
The system device name of the USB serial port device retrieved using the usbHostEnumerate function.

Returns: SINT

>=0 - The port number to use to open a connection to the camera.
-1 - The device is not a camera port or not found

Declaration:

```
FUNCTION usbHostGetCameraPort : SINT;
VAR_INPUT
   device : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc
VAR_INPUT
   doOpen   : BOOL;           | open first found USB camera
END_VAR;
VAR_OUTPUT
   isOpen   : BOOL := FALSE;  | is camera open
   failed   : BOOL;           | failed to open camera
END_VAR;

FUNCTION GetPortNum : SINT;
VAR
   device   : STRING;         // System name of USB device
   type     : INT := -1;      // Type of USB device
   index    : INT;
   port     : SINT;
   rc       : INT;
END_VAR;
// Enable USB host port
usbHostEnable(port:=1,enable:=TRUE);

GetPortNum := -1; // Not found
FOR index := 1 TO 32 DO
   rc := usbHostEnumerate(index := index, device := device, type := type);
   IF ((rc <= 0) OR (type = 7)) THEN
      EXIT;
   END_IF;
END_FOR;
IF type <> 7 THEN
   DebugMsg(message := "Failed to find any USB cameras.");
   RETURN;
END_IF;
DebugFmt(message := "Found '" + device + "' at index \1", v1 := index);
port := usbHostGetCameraPort(device := device);
```

```
IF port < 0 THEN
   DebugFmt(message := "Failed to get camera port number (code \1)", v1 :=
port);
END_IF;
GetPortNum := port;
END_FUNCTION;

PROGRAM example;
VAR
   portNum  : SINT := -1;     // Select which port to use for the
communication
   timer    : TON;
   rc       : INT;
END_VAR;

fsMediaOpen(media:=1);
BEGIN
   IF doOpen XOR isOpen THEN
      IF portNum >= 0 THEN
         camClose();
         DebugFmt(message := "Camera was closed");
         portNum := -1;
      END_IF;
      IF doOpen THEN
         portNum := GetPortNum();
         IF portNum >= 0 THEN
            rc := camOpen(port := portNum);
            IF rc = 0 THEN
               DebugFmt(message := "Camera \1 is open", v1 := portNum);
            ELSE
               DebugFmt(message := "Failed to open camera \1 (\2)", v1 :=
portNum, v2 := rc);
               portNum := -1;
            END_IF;
         END_IF;
         failed := (portNum < 0);
         timer.pt := 0;
      END_IF;
      isOpen := doOpen;
   END_IF;
   timer(trig := (isOpen AND NOT failed));
   IF timer.q THEN
      camSnapshotToFile(res := 7, filename := "B:\IMG\PIC.JPG");
      timer(trig := FALSE, pt := 5000); // reset timer
   END_IF;
END;
END_PROGRAM;
```

4.2.48 ver: Application version

4.2.48.1 ver: Application version functions

This group of functions allows versioning of the application. It also allows the application to take control when a new firmware or a new application is used.

- verSetAppProfile — This sets the application name and version.
- verGetAppVersion — This gets the application version.
- verGetAppName — This gets the application name.
- verCheckUpgrade — This determines if a background update is completed or in progress.

4.2.48.2 verSetAppProfile (Function)

Architecture: X32
Device support: All
Firmware version: 1.00

This function is used to set the name and version number of the application.
It is only necessary to set this information once - for example the first time the application is initialized.
The information will be saved to Flash memory by the firmware. The name and version number chosen are entirely up to the application and its contents are not used by the firmware.

Input:

name : STRING
The name of the application.
Maximum size is 15 characters.

version : INT
The version of the application scaled by 100 (2.10 = 210).

Returns:

None.

Declaration:

```
FUNCTION verSetAppProfile;
VAR_INPUT
   name    : STRING;  // name of application
   version : INT;     // Version of application
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   UpgChk : verCheckUpgrade;
END_VAR;

PROGRAM test;

   // Set Application name and version
   IF boardType() < 100 OR boardVersion() >= 252 THEN
      verSetAppProfile(name := "VerTest", version := 422);
```

```
   END_IF;
   ...

BEGIN
   UpgChk();
   ...
   // is Application ready to reset?
   IF ... THEN
      // is background upgrade ready?
      IF UpgChk.application OR UpgChk.firmware THEN
         // Reset RTCU device. The firmware or application is updated
automatically
         boardReset();
      END_IF;
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.48.3 verGetAppVersion (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

---

verGetAppVersion will return the version number that was previously set by the verSetAppProfile function or via Project: Settings.

Input:

None.

Returns: INT

The version of the application scaled by 100 (2.10 = 210).

Declaration:

```
FUNCTION verGetAppVersion : INT;
```

Example:

```
INCLUDE rtcu.inc

VAR
   UpgChk : verCheckUpgrade;
END_VAR;

PROGRAM test;

   // Get Application name and version
   DebugFmt(message := "Application: " + verGetAppName() + " \1.\2",
            v1 := verGetAppVersion() / 100,
            v2 := verGetAppVersion() MOD 100);
   ...

BEGIN
   UpgChk();
   ...
   // is Application ready to reset?
   IF ... THEN
      // is background upgrade ready?
      IF UpgChk.application OR UpgChk.firmware THEN
         // Reset RTCU device. The firmware or application is updated
```

```
automatically
         boardReset();
      END_IF;
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.48.4 verGetAppName (Function)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

---

verGetAppName will return the application name that was previously set with the verSetAppProfile function or via Project: Settings.

Input:
None.

Returns: STRING
The name of the application.

Declaration:
```
FUNCTION verGetAppName : STRING;
```

Example:

```
INCLUDE rtcu.inc

VAR
   UpgChk : verCheckUpgrade;
END_VAR;

PROGRAM test;

   // Get Application name and version
   DebugFmt(message := "Application: " + verGetAppName() + " \1.\2",
            v1 := verGetAppVersion() / 100,
            v2 := verGetAppVersion() MOD 100);
   ...

BEGIN
   UpgChk();
   ...
   // is Application ready to reset?
   IF ... THEN
      // is background upgrade ready?
      IF UpgChk.application OR UpgChk.firmware THEN
         // Reset RTCU device. The firmware or application is updated
automatically
         boardReset();
      END_IF;
   END_IF;
   ...
END;
END_PROGRAM;
```

4.2.48.5 verCheckUpgrade (Functionblock)

Architecture: X32 / NX32 / NX32L
Device support: All
Firmware version: 1.00 / 1.00.00

This function block is used to inform the VPL program that a background update is in progress and that a download of either a new firmware or a new application has completed.
With this functionality, the application can decide not to PowerDown() during an upgrade and it can also decide when the "switch" to the new application or firmware should happen. When a background update is available, the actual "upgrade" will happen when the device starts up next time - for example after a boardReset().

Input:

None.

Output:

firmware : BOOL
True if a new firmware is available as a background update.

application : BOOL
True if a new application is available as a background update.

transfer : BOOL
True if the application or firmware is being transferred in the background.

Declaration:

```
FUNCTION_BLOCK verCheckUpgrade;
VAR_OUTPUT
   firmware    : BOOL; // New firmware is available
   application : BOOL; // New application is available
   transfer    : BOOL; // New application or firmware being transferred in the
background
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR
   UpgChk : verCheckUpgrade;
END_VAR;

PROGRAM test;

   // Get Application name and version
   DebugFmt(message := "Application: " + verGetAppName() + " \1.\2",
            v1 := verGetAppVersion() / 100,
            v2 := verGetAppVersion() MOD 100);
   ...

BEGIN
   UpgChk();
   ...
   // is Application ready to reset?
   IF ... THEN
      // is background upgrade ready?
      IF UpgChk.application OR UpgChk.firmware THEN
```

```
            // Reset RTCU device. The firmware or application is updated
automatically
            boardReset();
         END_IF;
      END_IF;
      ...
END;
END_PROGRAM;
```

4.2.49 voice: Functions for delivering Voice Messages

4.2.49.1 Voice: Functions for delivering Voice

The voice functions enables speech to be delivered using the built in GSM module in the RTCU. Not all RTCU devices offers this capability, so please consult the data-sheet / technical manual for details.
The voice functions are typically used in conjunction with the GSM functions and the DTMF functions to implement interactive voice-response systems.

- voiceTalk — Say a voice message.
- voiceStop — Stop all voice messages in queue.
- voiceBusy — Query for if voice messages is currently playing.
- voiceSetChannel — Selects the channel to use for voice output.
- voicePresent — Query for voice messages present.
- voiceBackup — Make a backup of voice messages.
- voiceRestore — Restore voice messages from a backup.

See voiceTalk for a list of the supported audio formats.

4.2.49.2 voiceStop (Function)

Architecture: X32 / NX32 / NX32L
Device support: All with voice/DTMF capability
Firmware version: 1.00 / 1.00.00

Terminates any active voice messages. This function also clears the queue of voice messages.

Input:
None

Returns:
None

Declaration:

```
FUNCTION voiceStop;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Stop any voice that is playing
   voiceStop();
   ...
END;

END_PROGRAM;
```

4.2.49.3 voiceTalk (Function)

Architecture: X32 / NX32 / NX32L
Device support: All with voice/DTMF capability
Firmware version: 1.00 / 1.00.00

This function will play a pre-recorded voice message over the cellular module on the device. This functionality is useful for implementing voice/response systems in combination with the DTMF API.
The voice system has an queue of messages. When a new message is to be played, it is placed in this queue. The maximum number of voice messages in the queue is 32.
For an example program using the voiceTalk function, please look at the example below or look at the voicemessage example in the Examples section

If the specified message does not exist, a default "beep" message will be played.

On NX32L devices (from firmware R1.30.00) it is possible to play voice messages from locations other than the project drive.
The full path to the voice file is required. For example: "B:\voice\message.wav".

The following audio formats are supported:

	X32	NX32	NX32L
Sample rate:	4 kHz, 8kHz	4 kHz, 8kHz	4 kHz - 48kHz
Data bits:	8	8	8, 16, 24
Channels:	Mono	Mono	Mono, Stereo
Formats:	WAV	WAV	WAV, FLAC*, Ogg Vorbis*

* Requires runtime-firmware 1.80.00 or newer.

Input:

message : STRING
The name of the wav file to play.

pause : INT (0..25000) ***Default 150***
0..25000 : The number of milliseconds to wait after this message is finished, before the next message starts.

Returns:

None

Declaration:

```
FUNCTION voiceTalk;
VAR_INPUT
   message : STRING;
   pause   : INT := 150; // Wait default 150 mSec before next message
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

VAR_INPUT
   configured_message : STRING; | Voice message
END_VAR;

PROGRAM test;
```

```
BEGIN
   ...
   // Play the "Hello" message, and after the message finishes, wait 250
milliseconds
   voiceTalk(message := "Hello.wav", pause := 250);
   // Play the "Menu" message, 150 mSec between this and the next message in
queue
   voiceTalk(message := "Menu.wav");
   // Play the configured message, no delay between this message and the next
(no one in this example)
   voiceTalk(message := configured_message, pause := 0);
   ...
END;

END_PROGRAM;
```

4.2.49.4 voiceBusy (Function)

Architecture: X32 / NX32 / NX32L
Device support: All with voice/DTMF capability
Firmware version: 1.00 / 1.00.00

Will return information about if voice-messages currently is being played.

Input:
None

Returns: **BOOL**
TRUE: Voice messages is being played.
FALSE: No voice messags is being played.

Declaration:

```
FUNCTION voiceBusy : BOOL;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Test for voice messages being played.
   IF NOT voiceBusy() THEN
      ...
   END_IF;
   ...
END;

END_PROGRAM;
```

4.2.49.5 voiceSetChannel (Function)

Architecture: X32 / NX32 / NX32L
Device support: MX2 turbo/encore/warp, DX4i pro, NX-200, NX-400
Firmware version: 4.00 (4.10 for DX4i pro) / 1.00.00

This function is used to select where the voice output should be routed to.

By default the voice will be routed to the GSM connection for voice/DTMF use, but it can also be routed to the xVoice output available on some devices. Please consult the technical manual for more information.
Using xVoice it is possible to play digitized voice to an external speaker.

Input:

ch : SINT (0..1) ***Default 0***
The channel to route the output to.
0 - GSM
1 - xVoice output

Returns: INT

0 - Operation was successful.
1 - Invalid channel.
5 - Not supported on this device.

Declaration:

```
FUNCTION voiceSetChannel : INT;
VAR_INPUT
   ch : SINT;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

BEGIN
   ...
   // Switch to external voice output
   voiceSetChannel(ch := 1);
   // Play the "Welcome" message
   voiceTalk(message := "Welcome.wav");
   ...
END;

END_PROGRAM;
```

4.2.49.6 voiceSetVolume (Function)

Architecture: X32 / NX32
Device support: MX2 turbo/encore/warp, AX9 turbo/encore, DX4i pro-2G mk2
Firmware version: 4.56

Controls the volume level used by voiceTalk when playing messages during a voice session.
The volume level does not refer to an absolute level and may vary between different device types.

Input:

volume: SINT (0..100) ***Default 50***
Playback volume.

Returns: INT

0 - Operation was successful.
1 - Invalid volume
3 - General error
5 - Not supported on this device.

Declaration:

```
FUNCTION voiceSetVolume : INT;
VAR_INPUT
   volume : SINT := 50;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;

// Set the voice message playback volume
voiceSetVolume(volume := 90);

// Turn on power to the GSM module
gsmPower(power := TRUE);
...
BEGIN
   ...
   // Play the "Welcome" message
   voiceTalk(message := "Welcome.wav");
   ...
END;

END_PROGRAM;
```

4.2.49.7 voicePresent (Function)

Architecture: X32 / NX32 / NX32L
Device support: All with voice/DTMF capability
Firmware version: 4.34 / 1.00.00

Will return information about if voice-messages are present on device.

Input:

None

Returns:

TRUE: Voice messages are present.
FALSE: No voice messages are present.

Declaration:

```
FUNCTION voicePresent : BOOL;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   upg : verCheckUpgrade;
END_VAR;

   // Initialize file system
   fsMediaOpen(media := 1);
   // Change media
   fsDirChange(path := "B:\");
```

```
   // Restore voice
   upg();
   IF NOT upg.transfer AND NOT upg.firmware AND NOT upg.application THEN
      //
      // Restore voice
      //
      IF NOT voicePresent() AND fsFileExists(name := "b:\voice.bak") THEN
         DebugMsg(message := "Restoring voice...");
         DebugFmt(message := "voiceRestore = \1", v1 := voiceRestore(filename
:= "voice.bak"));
      END_IF;

      //
      // Backup voice
      //
      IF voicePresent() AND NOT fsFileExists(name := "b:\voice.bak") THEN
         DebugMsg(message := "Backing up voice...");
         DebugFmt(message := "voiceBackup = \1", v1 := voiceBackup(filename :=
 "voice.bak"));
      END_IF;
   END_IF;

BEGIN
   ...
END;
END_PROGRAM;
```

4.2.49.8 voiceBackup (Function)

Architecture: X32 / NX32
Device support: All with voice/DTMF capability
Firmware version: 4.34

This function will make a backup of the voice messages on the device and store it to a file on the file system.
The voice messages can be restored from the file with the voiceRestore function.
This function is only supported when the project in the device is compiled in X32 compilation mode.

Important:
The voice backup file is not compatible between devices with X32 and NX32 execution architectures.
This means that a backup taken from a MX2 can be restored to other MX2 or DX4 devices (both X32 execution architecture), but not a MX2 turbo (NX32 execution architecture).

Input:

filename : STRING
The name of the file to store the voice backup in.

Returns:

0 - Backup was successful.
1 - No voice messages present.
2 - Could not create file.
3 - Drive is full.
4 - Cannot make a backup from a Project drive.

Declaration:

```
FUNCTION voiceBackup : INT;
VAR_INPUT
```

```
   filename : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   upg : verCheckUpgrade;
END_VAR;

   // Initialize file system
   fsMediaOpen(media := 1);
   // Change media
   fsDirChange(path := "B:\");

   // Restore voice
   upg();
   IF NOT upg.transfer AND NOT upg.firmware AND NOT upg.application THEN
      //
      // Restore voice
      //
      IF NOT voicePresent() AND fsFileExists(name := "b:\voice.bak") THEN
         DebugMsg(message := "Restoring voice...");
         DebugFmt(message := "voiceRestore = \1", v1 := voiceRestore(filename
:= "voice.bak"));
      END_IF;

      //
      // Backup voice
      //
      IF voicePresent() AND NOT fsFileExists(name := "b:\voice.bak") THEN
         DebugMsg(message := "Backing up voice...");
         DebugFmt(message := "voiceBackup = \1", v1 := voiceBackup(filename :=
 "voice.bak"));
      END_IF;
   END_IF;

BEGIN
   ...
END;
END_PROGRAM;
```

4.2.49.9 voiceRestore (Function)

Architecture: X32 / NX32
Device support: All with voice/DTMF capability
Firmware version: 4.34

This function will restore voice messages from a file created with the voiceBackup function. This function is only supported when the project in the device is compiled in X32 compilation mode.

Important note for X32 execution architecture devices:
Calling this function will erase the flash area where the upgrade image is located overwriting any pending and uncommitted updates.
It is therefore advisable to check with verCheckUpgrade for any pending upgrades before calling this function.

Input:

filename : STRING
The name of the file to restore the voice backup from.

Returns:

0 - Restore was successful.
1 - Could not open file.
2 - Not a valid voice backup file.
4 - Cannot restore to a Project drive.

Declaration:

```
FUNCTION voiceRestore : INT;
VAR_INPUT
   filename : STRING;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM test;
VAR
   upg : verCheckUpgrade;
END_VAR;

   // Initialize file system
   fsMediaOpen(media := 1);
   // Change media
   fsDirChange(path := "B:\");

   // Restore voice
   upg();
   IF NOT upg.transfer AND NOT upg.firmware AND NOT upg.application THEN
      //
      // Restore voice
      //
      IF NOT voicePresent() AND fsFileExists(name := "b:\voice.bak") THEN
         DebugMsg(message := "Restoring voice...");
         DebugFmt(message := "voiceRestore = \1", v1 := voiceRestore(filename
:= "voice.bak"));
      END_IF;

      //
      // Backup voice
      //
      IF voicePresent() AND NOT fsFileExists(name := "b:\voice.bak") THEN
         DebugMsg(message := "Backing up voice...");
         DebugFmt(message := "voiceBackup = \1", v1 := voiceBackup(filename :=
 "voice.bak"));
      END_IF;
   END_IF;

BEGIN
   ...
END;
END_PROGRAM;
```

4.2.50 zp: Zero Power functions

4.2.50.1 zp: Zero Power Functions

Zero Power mode is an ultra low power saving mode that uses so little power that it can effectively be ignored and considered zero power consumption.
By using Zero Power mode, it is possible to extend operating time on batteries so that several years of operation can be achieved.

When operating in Zero Power mode, the device powers down and by a selected event wakes up by restarting the device.

The following functions are used to control the Zero Power mode:

zpPowerDown	The device enters Zero Power mode and waits for an event.
zpBootEvent	This queries which event that booted the device.

Zero Power mode is only supported on the RTCU SX1 series. Please consult the technical manual for further information about the configuration and use.

4.2.50.2 zpPowerDown (Function)

```
Architecture:       X32
Device support:     SX1
Firmware version:   2.70
```

The device enters Zero Power mode and waits for a specific event to occur before restart.
Zero Power mode is similar to pmPowerDown but uses much less power which allows operation for several years solely from the on-board battery of the device.

The device will wake up and reboot when one of several events occur:

- Vibration.
- Timeout.
- Digital input 1 change (controlled by dip-switch).
- Digital input 2 change (controlled by dip-switch).
- External power applied.
- Hardware reset (reset switch).

Please refer to the technical manual for the RTCU SX1 series for detailed information on the configuration of the digital inputs.

After exiting the Zero Power mode, the wake-up cause can be received by querying the boot event with zpBootEvent.

Input:

Time: DINT (-1 .. 2147483647) (default -1)
This is the time to power down in minutes. Leave this empty if not used.

Vibration: BOOL (default FALSE)
This enables vibration detection (see pmSetVibrationSensivity for adjusting sensitivity).

Returns:

None.

Declaration:

```
FUNCTION zpPowerDown;
VAR_INPUT
   Time       : DINT := -1;
   Vibration  : BOOL := FALSE;
END_VAR;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
DebugFmt(message := "Reason for boot : \1", v1 := zpBootEvent());
BEGIN
   // boot once a day or on vibration
   zpPowerDown(Time := (60*24), Vibration := TRUE);
END;

END_PROGRAM;
```

4.2.50.3 zpBootEvent (Function)

Architecture: X32
Device support: SX1
Firmware version: 2.70

This function will query the event which has caused the device to boot.

The boot events have the following precedence when multiple sources have been detected:

- Power Down has been called (see pmPowerDown).
- Hardware reset (reset switch used).
- Vibration (see zpPowerDown).
- Timeout (see zpPowerDown).
- Digital input 1 change.
- Digital input 2 change.
- External power applied.
- Software reset (e.g. boardReset, boardWatchdog, etc.)

If no change in system state can be detected, the function will return "-1". An example would be if the external power was removed and applied after entering Zero Power mode by using zpPowerDown.

Input:

None.

Returns: INT

7	Digital input 2 change.
6	Digital input 1 change.
5	Vibration (see zpPowerDown).
4	Timeout (see zpPowerDown).
3	External power applied.
2	Power Down has been called (see pmPowerDown).
1	Hardware reset (reset switch used).
0	Software reset (e.g. boardReset, boardWatchdog, etc.)
-1	Unknown boot source.

Declaration:

```
FUNCTION zpBootEvent : INT;
```

Example:

```
INCLUDE rtcu.inc

PROGRAM example;
DebugFmt(message := "Reason for boot : \1", v1 := zpBootEvent());
BEGIN
   // boot once a day or on vibration
   zpPowerDown(Time := (60*24), Vibration := TRUE);
END;

END_PROGRAM;
```

* * * * THIS PAGE IS INTENTIONALLY LEFT BLANK * * *

I/O Extension

5 I/O Extension

5.1 Introduction

By using the I/O Extension functionality, it becomes extremely simple to extend the number and the diversity of I/Os beyond those that are available natively on the RTCU device by adding cost-effective external I/O modules.

The I/O Extension functionality offers the following key features:

- High performance fail-safe MODBUS implementation - RTU protocol version.
- Uses industry standard MODBUS-based I/O modules
- Physical layer used is RS-485 available as a standard or option feature.
- MODBUS master mode supports seamless I/O extension with up to 64 I/O modules and 1024 * I/O points.
- MODBUS slave mode support operating as an I/O node under the control of for example a SCADA system.
- MODBUS master and slave mode can operate simultaneously in advanced application scenarios.
- Fully transparent I/O extension. This means that resources on external I/O modules become an integral part of the on-board I/O subsystem.
- Automatic run-time I/O maintenance. This means that absolutely no application handling is required to operate the I/O extension.
- Comprehensive exception handling paradigm for support of mission critical applications.
- Almost any standard MODBUS I/O module can be used but modules offered by Logic IO are especially easy to set up and have hassle-free operation.

Using the I/O Extension functionality is extremely simple, and it is recommended to go through the Using the I/O Extension (master mode) section of this manual to get the best and fastest working experience. Within a few minutes, it is possible to configure and use MODBUS-based remote I/O as if they were an integral part of the on-board I/Os.

The I/O Extension functionality requires no VPL application support to work as everything is operated automatically by the powerful firmware-based I/O manager. Please note that for the same reason IOs which are controlled by another MODBUS master in slave mode must not be mapped in the Job as updates done by the firmware-based I/O manager will be overwritten by the application I/O manager. Please see Using the I/O Extension (slave mode) for implementation reference.

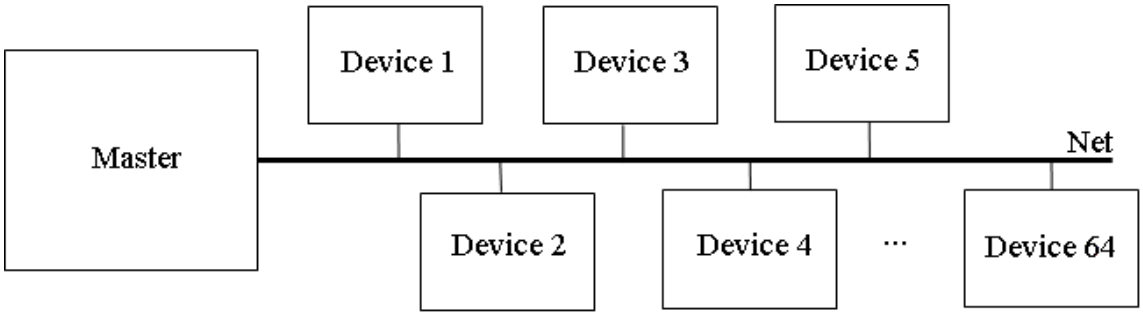
For specialised applications where full control of the MODBUS communication is required, it is possible to bypass the I/O Extension and use basic MODBUS communication primitives that allow transmission and reception of data according to the MODBUS protocol. Please reference the MODBUS API section for further information.

The in- and outputs on external I/O devices are updated asynchronously with the on-board I/O to ensure that the communication required will not slow down the execution of the application. There is also an option for synchronous updates if such is required (see ioSetMode for the possible update modes).

5.2 Architecture

I/O extension uses two concepts: nets and devices. An I/O extension network consists of a master and up to 64 slave devices connected to the net by using the RS-485 physical media. Connection between the master and the devices is called a "net". The master device updates the in- and outputs by reading/writing to the registers of the slave devices. Each slave device has a unique ID and only accepts commands that are sent to it.

This architecture is illustrated below:



Nets
A net is a connection to the RTCU device where one or more devices can be connected.
There are two different modes in which a net can be used: master and slave.

Devices
A device is a remote device that offers inputs and/or outputs. Only a master device can trigger a communication with a device.

Modes
The RTCU device can act as master or slave on the MODBUS network.

- Master: In master mode, the RTCU device uses the connected devices to extend the number of in- and outputs.
  When it is configured as master, it controls the communication with the devices.

- Slave : In slave mode, the RTCU device offers its in- and outputs to a remote master device.

5.3 Using the I/O Extension, Master mode

This tutorial will guide you through the use of the I/O Extension and will in detail demonstrate and explain the steps required to use the powerful features available.

The following steps are required to set up and use the I/O Extension:

1. Creating the application and the job.
2. Adding the net to the I/O Extension.
3. Adding MODBUS I/O modules to the net.
4. Configuring the job and effectively binding variables to physical I/O.
5. Testing the application in the RTCU Emulator.
6. Transferring the application to a physical RTCU device.
7. Monitoring the I/O by using the I/O Monitor dialog.

1. Creating the Application and the Job

To illustrate the I/O Extension, the following small VPL program will be used:

```
INCLUDE rtcu.inc

VAR_INPUT
   DI1 : BOOL;
   DI2 : BOOL;
END_VAR;

VAR_OUTPUT
   DO1 : BOOL;
   DO2 : BOOL;
END_VAR;

PROGRAM main;
BEGIN
   DO1 := DI1;
   DO2 := DI2;
END;
END_PROGRAM;
```

The above application is very simple as it has two BOOL variables in the VAR_INPUT section and another two BOOL variables in the VAR_OUTPUT section.
The logic is equally simple as the only operation consists of copying the state of the two input/output pairs to each other.

Despite the simplicity of the above application, it will allow us to demonstrate how to set up and configure the I/O variables so that:

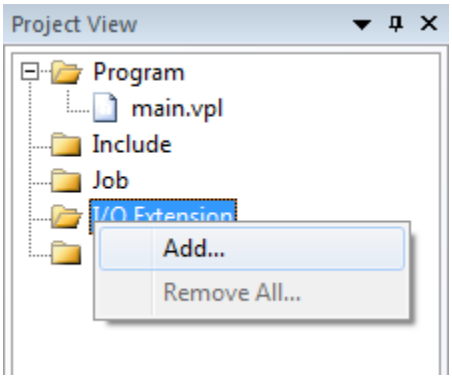
- DI1 will be allocated to external I/O digital input #1.
- DO1 will be allocated to the on-board digital output #1.
- DI2 will be allocated to external I/O digital input #2.
- DO2 will be allocated to the on-board digital output #2.

For detailed information on how to create and compile the application and create the job, please refer to the Tutorial section.

2. Adding the Net to the I/O Extension

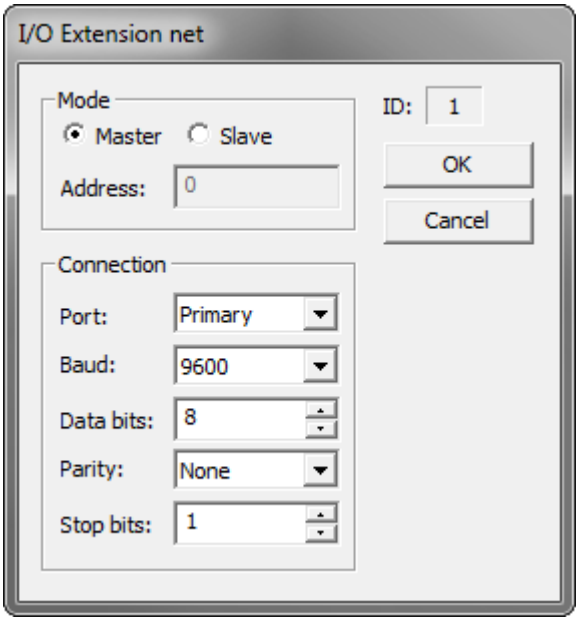
After the application and the job have been created, a net must be added to the I/O Extension.

Define a net by right clicking on the item "I/O Extension" and then click "Add..."



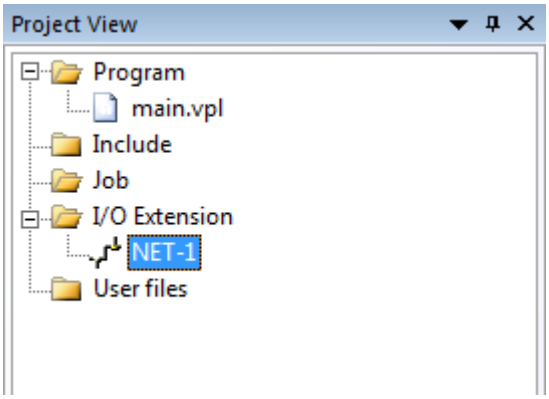
In the "I/O Extension net" it is possible to choose whether the connected RTCU device will act as a master to control the I/O devices or as a slave to allow another device to use its input/outputs. In this tutorial we will configure the device to be a master.

Please note that only serial channels with RS-485 capability are supported by the I/O extension. It is not possible to change the net label and the net ID as it is currently limited to defining only one net.



Please make sure that the "Connection" parameters (Port, Baud, Data bits, etc.) match the parameters of the physical MODBUS network deployed.

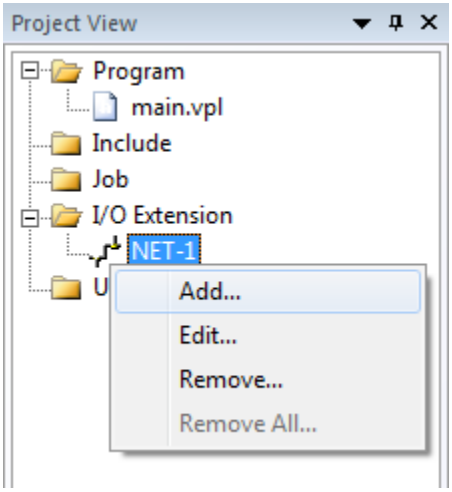
Click on "OK", then a net is created on the project tree.



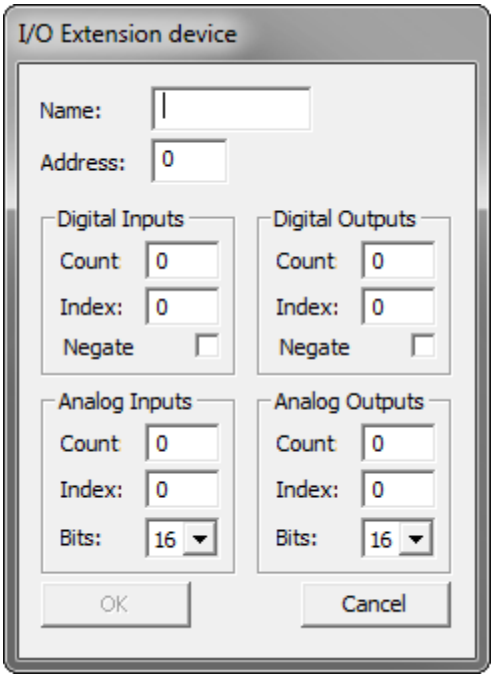
3. Adding MODBUS I/O modules to the Net

After the Net has successfully been added, it is time to add one or many I/O modules to the net. To demonstrate the concept of adding MODBUS I/O modules to the net, we will limit this step to only one module. Adding several modules is, however, equally simple if the outlined steps are followed.

The next step is to add the I/O Module to the net already present in the project. Right-click on the net labeled "NET-1" and click on "Add..."

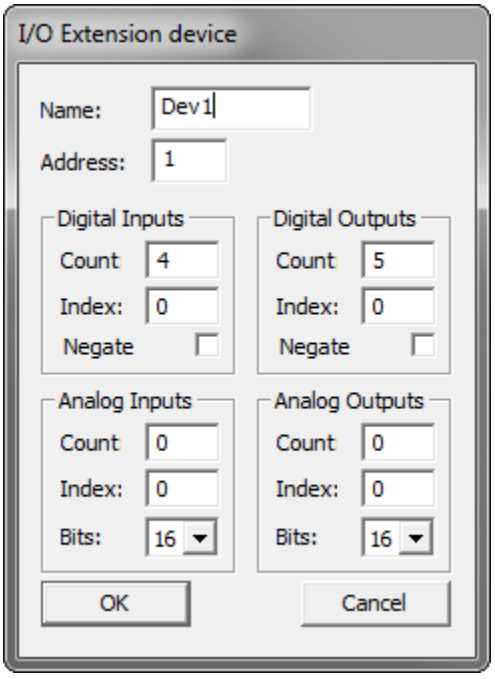


The following dialog shows:



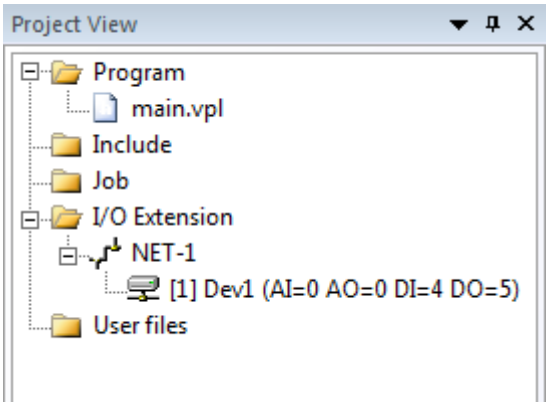
In the above dialog, a descriptive name for the I/O module/device can be given. In addition, the MODBUS address and various other parameters, as described in the Project Control - I/O extension device, can be entered. It is important to get all the parameters for the specific module correct. Therefore it may be necessary to consult the technical documentation for the specific module.

In our case we will be adding an I/O module with 4 digital inputs and 5 digital outputs:

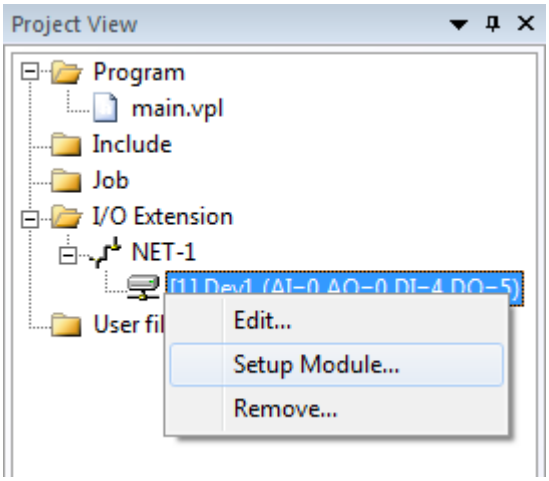


As it can be seen above, the name of the module is "Dev1" and the MODBUS address is "1".

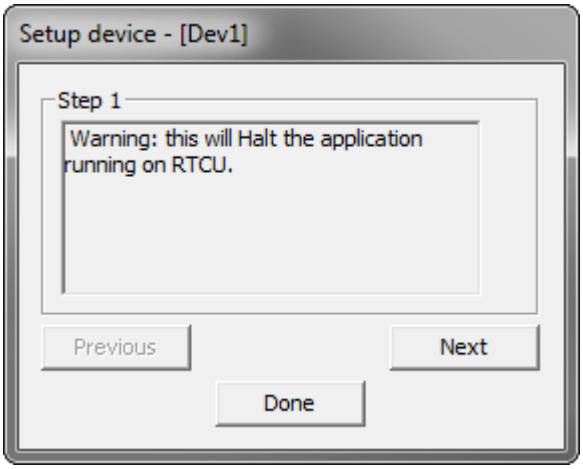
When this has been done, a new item should appear in the project as in the following figure:



Please note that the modules delivered by Logic IO are configured to 9600 bps and have the assignment "address 1". If another baud rate or address is set in step 2 above, or more than one module is attached to the net with the same address, please right-click on the device name and then click on "setup module".



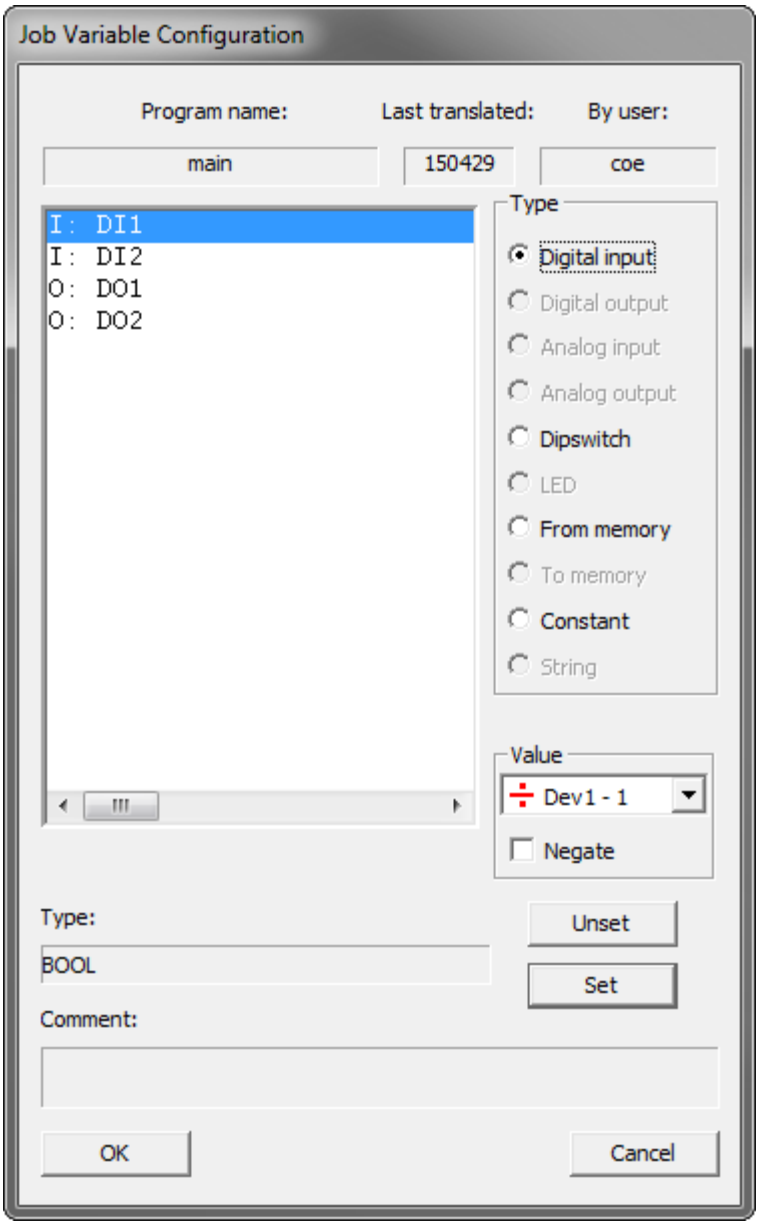
A setup wizard for the modules delivered by Logic IO as shown below will guide you through this in order to change the configuration of the device:



4. Configure the Job, effectively binding variables to physical I/O

As there is no difference for I/Os located natively on-board on RTCU devices and I/Os located remotely on a MODBUS module, the job configuration step works virtually the same as described in the Configure Job section.

The job configuration:

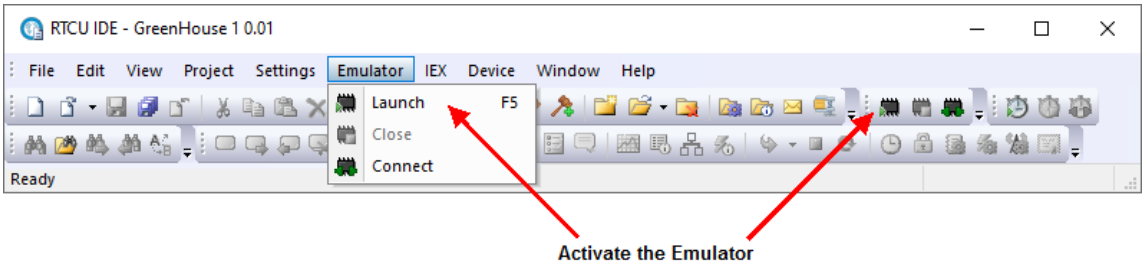


As it can be seen from the above configuration, the configuration as described in step 1 has now been defined:

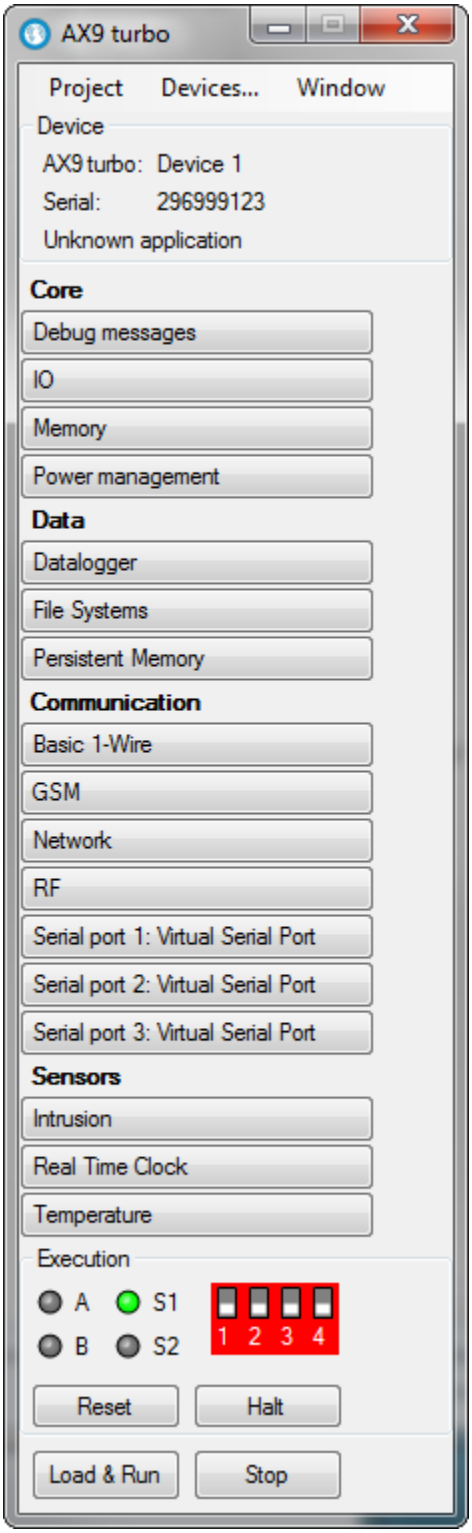
- DI1 will be allocated to external I/O digital input #1.
- DO1 will be allocated to the on-board digital output #1.
- DI2 will be allocated to external I/O digital input #2.
- DO2 will be allocated to the on-board digital output #2.

5. Test the application in the RTCU Emulator

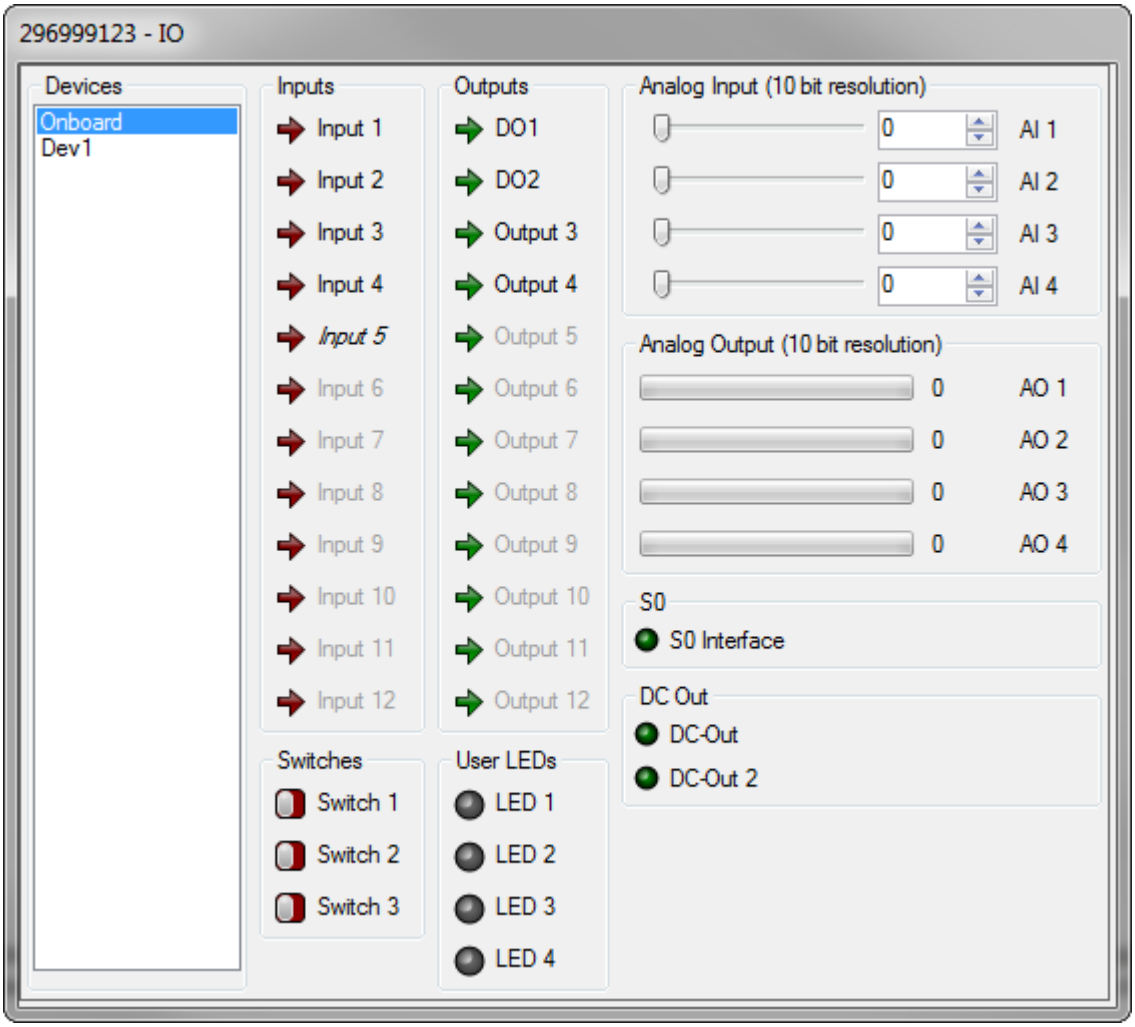
Let us test the project that we have created above. Click on the "Emulator" menu item, then "Emulator..." in the drop-down menu:



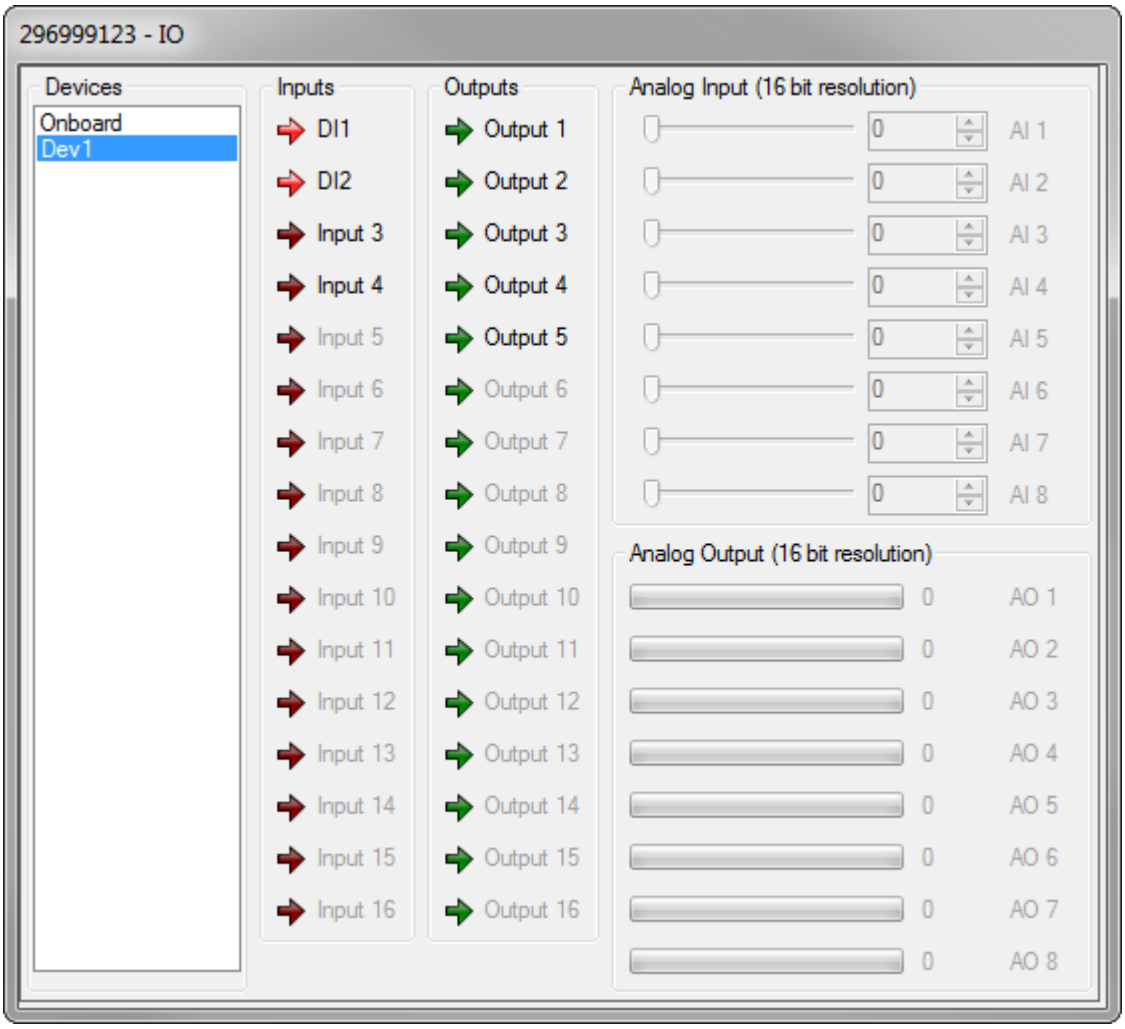
The Emulator window will appear as in the following:



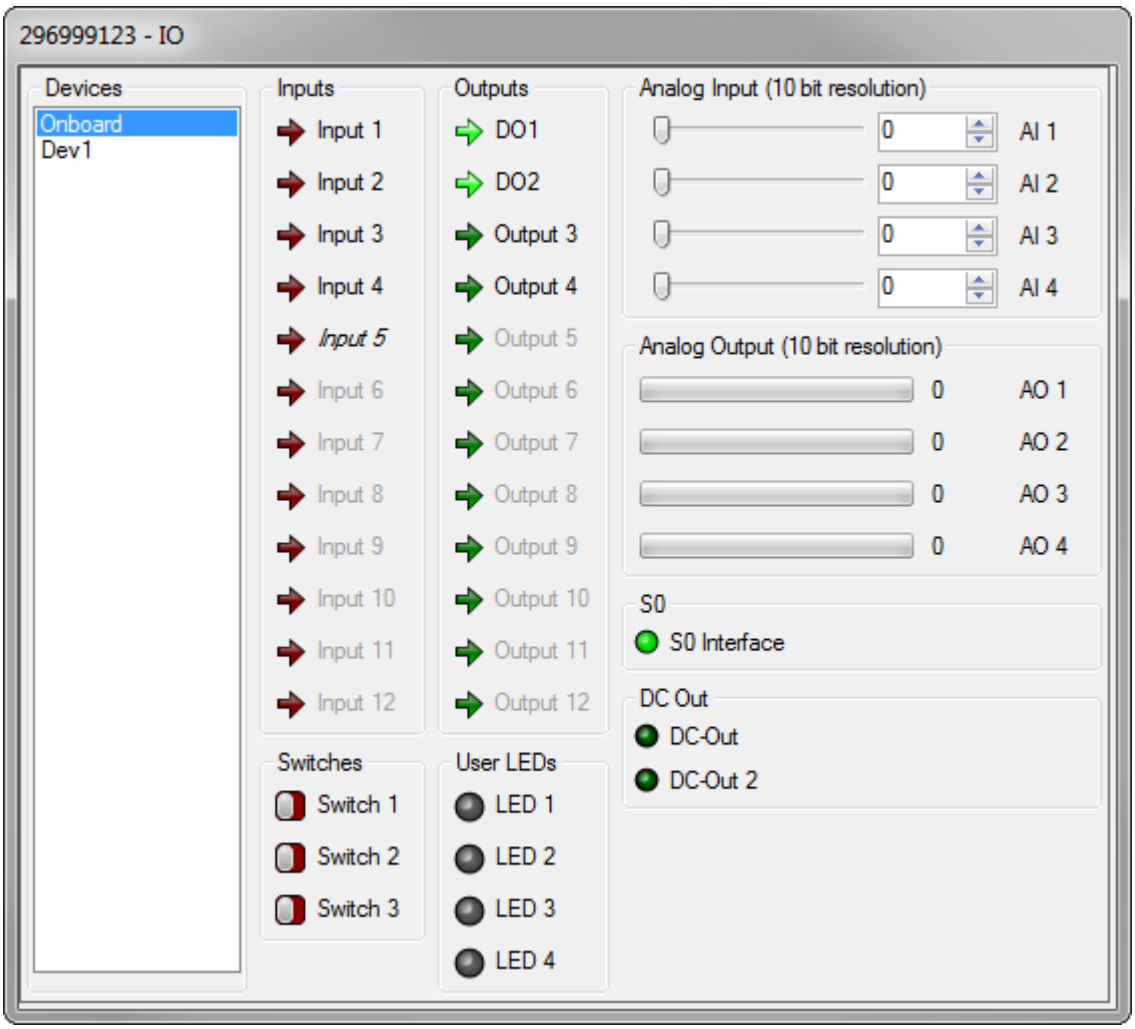
Enable the I/O window by clicking on "IO". This will bring up the following dialog:



Please note that there are two devices in the "Devices" section. By default the window shows native inputs/outputs, but extended I/Os can be monitored by clicking on the "Dev1". The output variables "DO1" and "DO2" appears on the "Onboard/Output" section. Let us start the emulator by clicking on "Load & Run" on the emulator control window. The applications program is now running. As there are not applied any values to the inputs, DO1 and DO2 show "0". Click on "Dev1" and then activate "DI1" and "DI2" by clicking on the arrow next to them. The I/O control windows should now look like the following:



Observe the "DO1" and "DO2" by clicking on "Onboard" in the "Devices" list. These two outputs should look like:

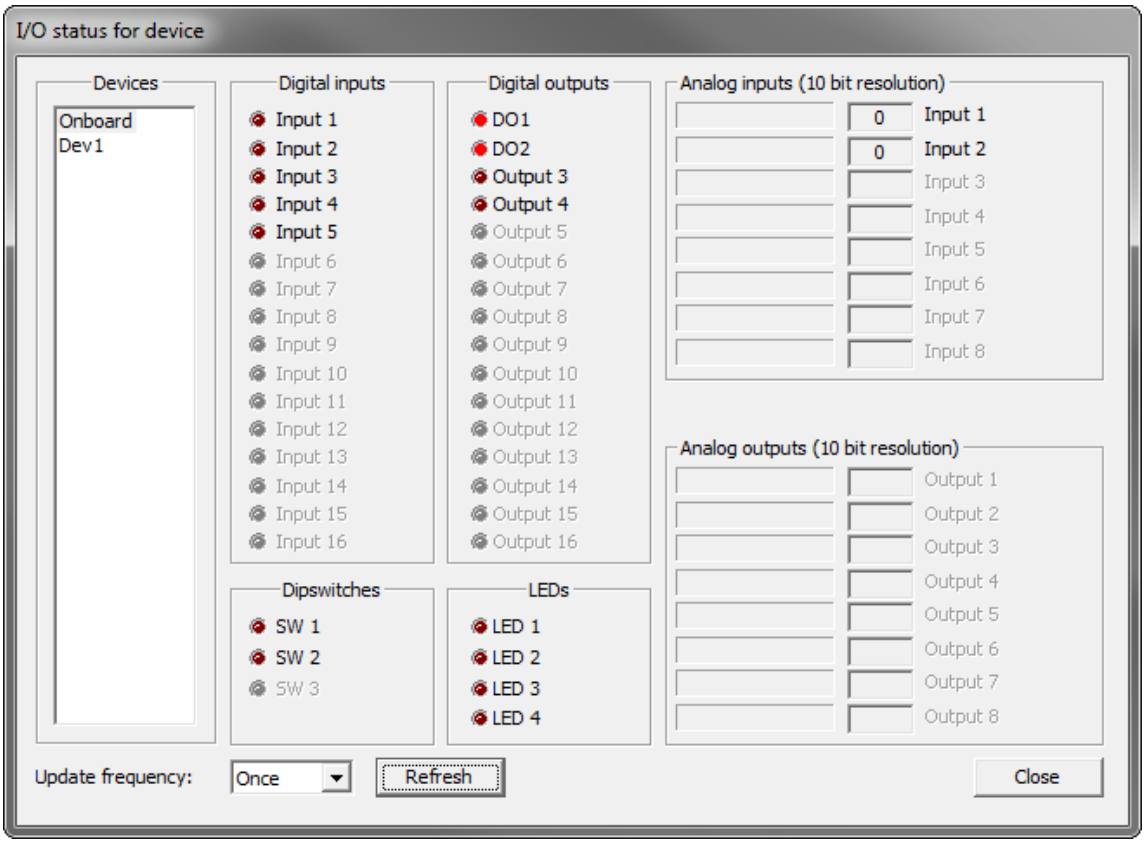


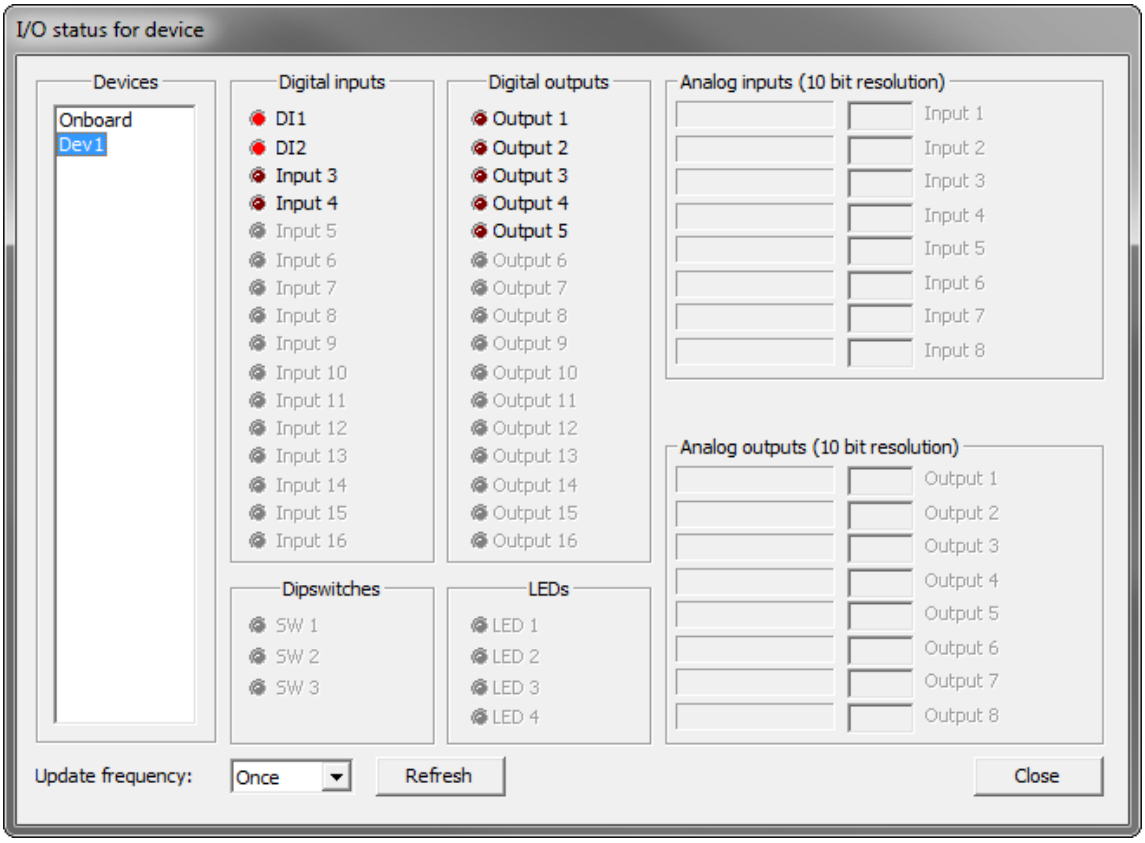
6. Transfer the application to a physical RTCU device

Transferring the application to a physical device is described in the Transfer to RTCU section. As the configured I/O Extension information is an integral part of the project, it will automatically be transferred to the device. If the device attached does not support the I/O Extension information, it will simply be ignored, and the application may not operate as intended.

7. Monitor I/O using the I/O Monitor dialog

The Device I/O Monitor fully supports the I/O Extension and on-board I/O as well, and remote I/O can easily be monitored. The following two figures show the I/O status of the project in this tutorial running in a physical device while "Onboard" and "Dev1" are selected in the "Devices" section:





This is the end of the tutorial. For further information on I/O Extension dialogs, please consult the sections Project Control - I/O extension, Project Control - I/O extension net, Project Control - I/O extension device, and Project Control - Setup Module.

5.4 Using the I/O Extension, Slave mode

This tutorial will guide you through the use of the I/O Extension in slave mode and will in detail demonstrate and explain the steps required to use the powerful features available.

The following steps are required to set up and use the I/O Extension:

1. Creating the application and the job.
2. Adding the net to the I/O Extension.
3. Configuring the job and effectively binding variables to physical I/O .
4. Testing the application in the RTCU Emulator.
5. Transferring the application to a physical RTCU device.
6. Monitoring the I/O by using the I/O Monitor dialog.

1. Creating the Application and the Job

To illustrate the I/O Extension for a slave, the following small VPL program will be used:

```
INCLUDE rtcu.inc

VAR_OUTPUT
   LED : BOOL;
END_VAR;

PROGRAM main;
VAR
   cur   : ARRAY [1..4096] OF DINT;
   old   : ARRAY [1..4096] OF DINT;
   i     : INT := 10000;
END_VAR;

BEGIN
   IF (i > 4096) THEN
      memioReadX(index:=1, mem:=ADDR(cur), len:=4096);
      i := 1;
   ELSE
      IF NOT ( cur[i] = old[i] ) THEN
         LED := NOT LED;
         old[i] := cur[i];
      END_IF;
      i := i + 1;
   END_IF;
END;
END_PROGRAM;
```

The above application is very simple with a single BOOL variable in the VAR_OUTPUT section and access to the memory I/O.
The logic is equally simple as the only operation is to check whether the memory has changed and toggle an output.

Despite the simplicity of the above application, it will allow us to demonstrate how to set up and configure the I/O variables so that:

- LED will be allocated to the on-board user LED.
- Memory I/O can be accessed by a MODBUS master.

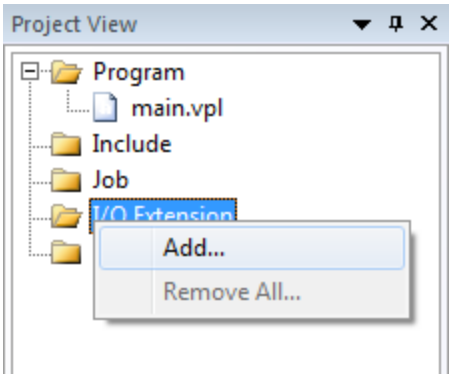
For detailed information on how to create and compile the application and create the job, please refer to the Tutorial section.

2. Adding the Net to the I/O Extension

After the application and the job have been created, a net must be added to the I/O Extension.

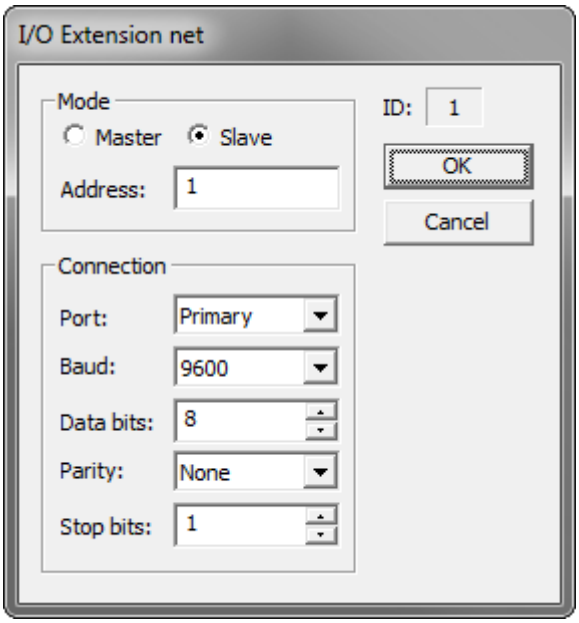
To allow adding a net, first enable X32 or NX32 in Project Settings.

Then define a net by right-clicking on the item "I/O Extension" and then click on "Add..."



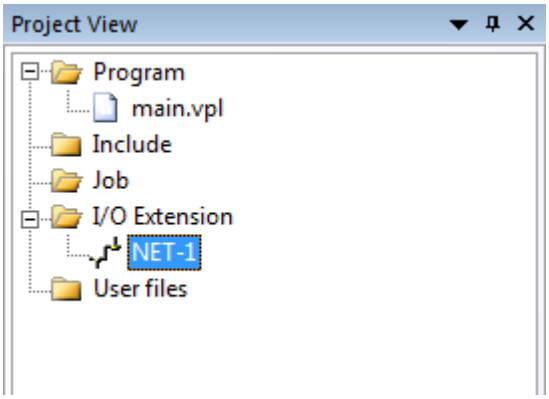
In the "I/O Extension net" it is possible to choose whether the connected RTCU device will act as a slave to let another device use its input/outputs or as a master to control the I/O devices. In this tutorial we will configure the device to be a slave in order to show the purpose of I/O expansion.

Please note that only serial channels with RS-485 capability are supported by the I/O extension. It is not possible to change the net label and the net ID as it is currently limited to defining only one net.



Please make sure that the "Connection" parameters (Port, Baud, Databits, etc. - see Project Control - I/O Extension net) match the parameters of the physical MODBUS network deployed and that an unused MODBUS address is used.

Click on "OK and a net will be created on the project tree.

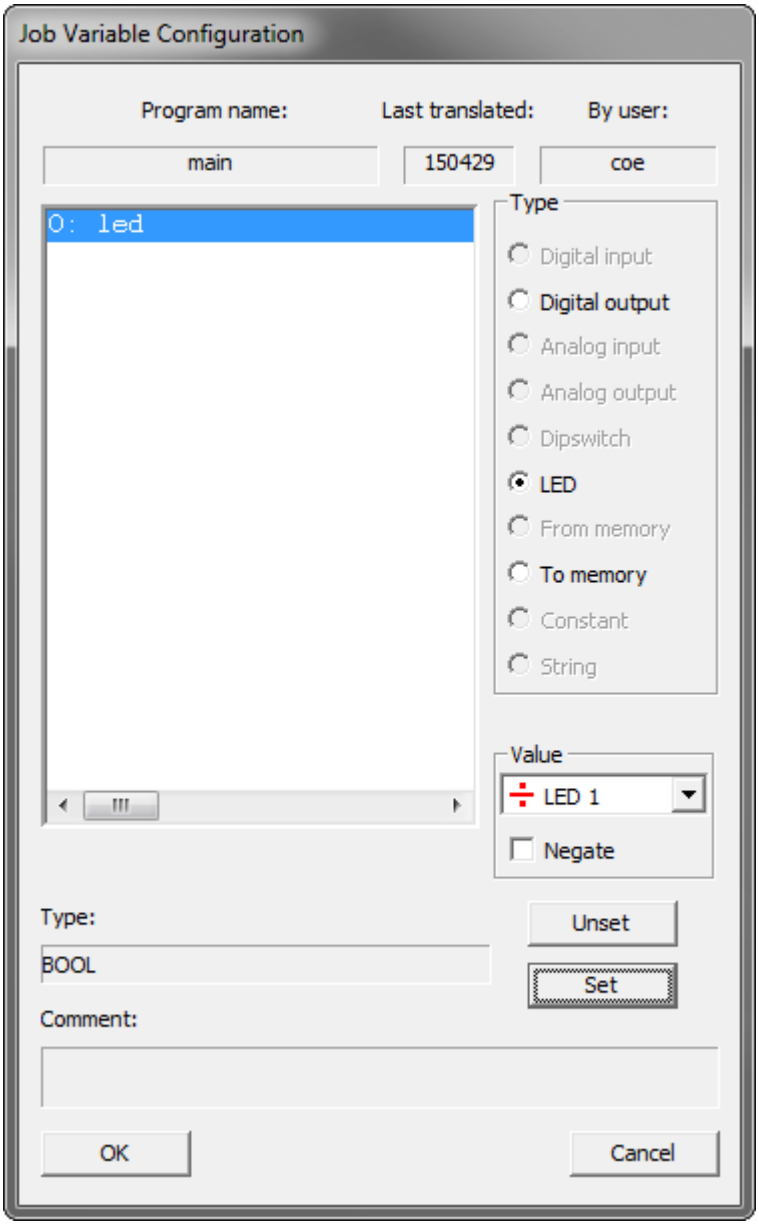


4. Configure the Job, effectively binding variables to physical I/O

When working as a MODBUS slave, it is vital to remember that any I/O which will be controlled by a master must not be mapped in the job variable configuration. The reason for this is that any value written by the master will be overwritten with the application values at BEGIN-END/UPDATEIO.

Any I/O not controlled by the MODBUS master can be mapped and used. For further details, please see the Configure Job section.

The job configuration:



As it can be seen from the above configuration, the configuration as described in step 1 has now been defined:

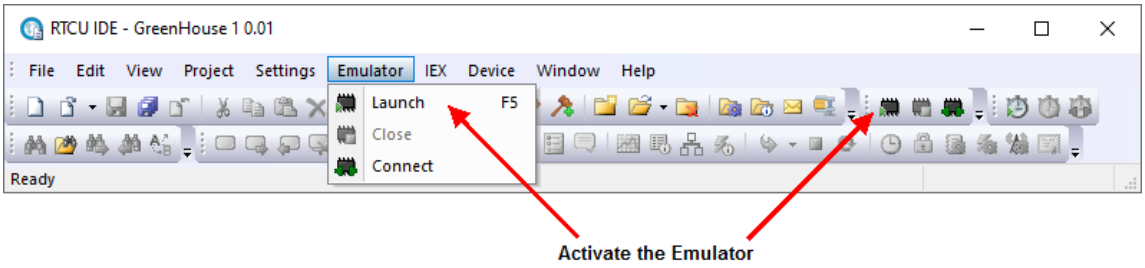
- LED will be allocated to the on-board user-controlled LED 1.

5. Test the application in the RTCU Emulator

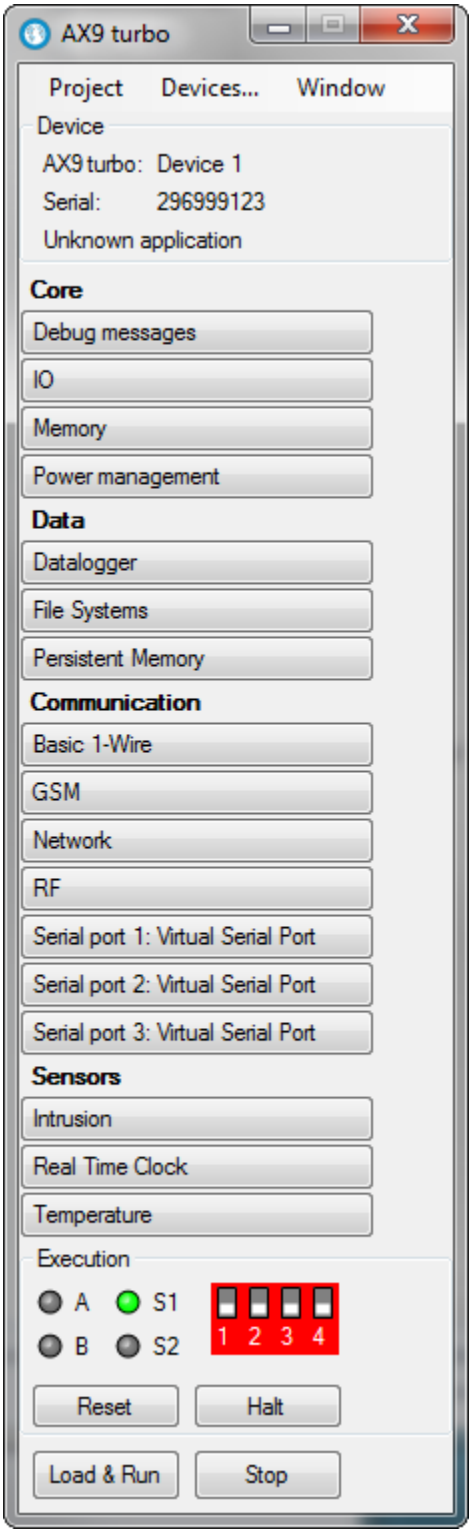
Let us test the project that we have created above.

First refresh the job by selecting "Build" from the "Project" menu.

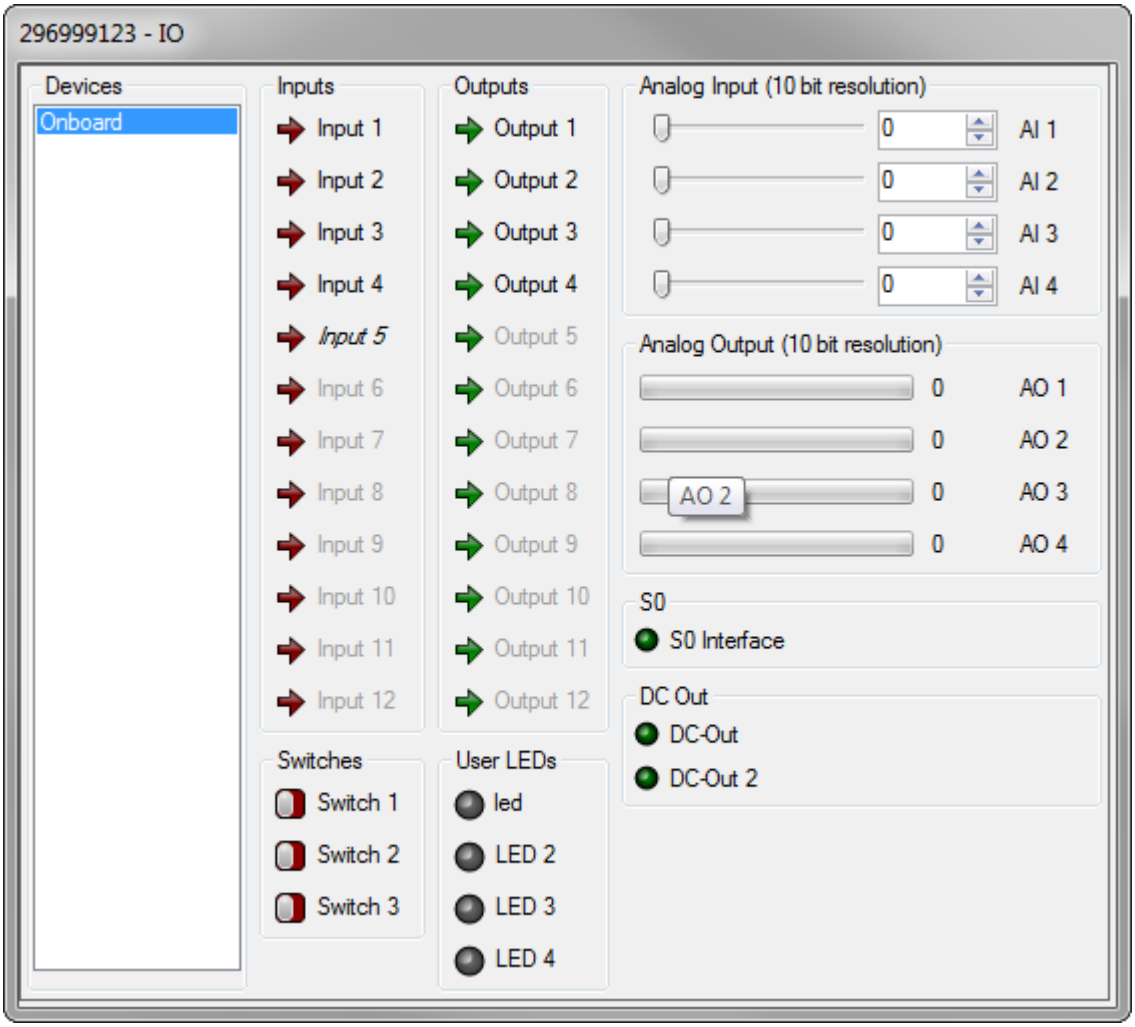
Then click on the "Emulator" menu item and finally click on "Emulator..." in the drop-down menu:



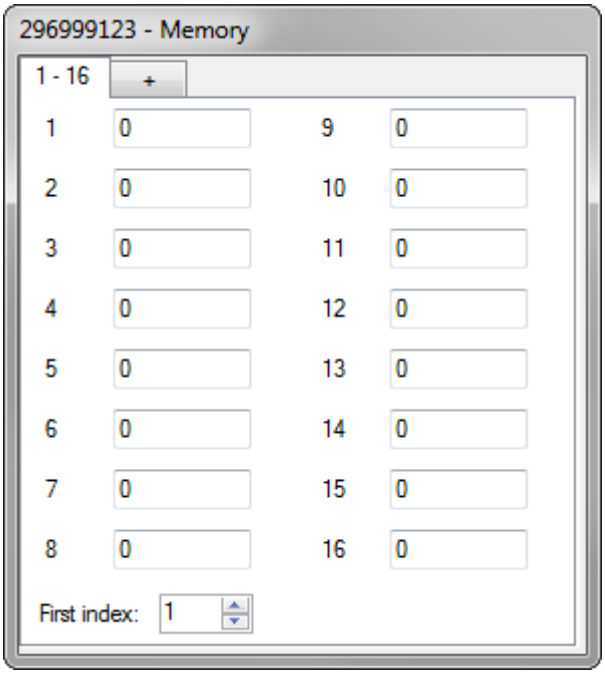
The Emulator window will appear as in the following:



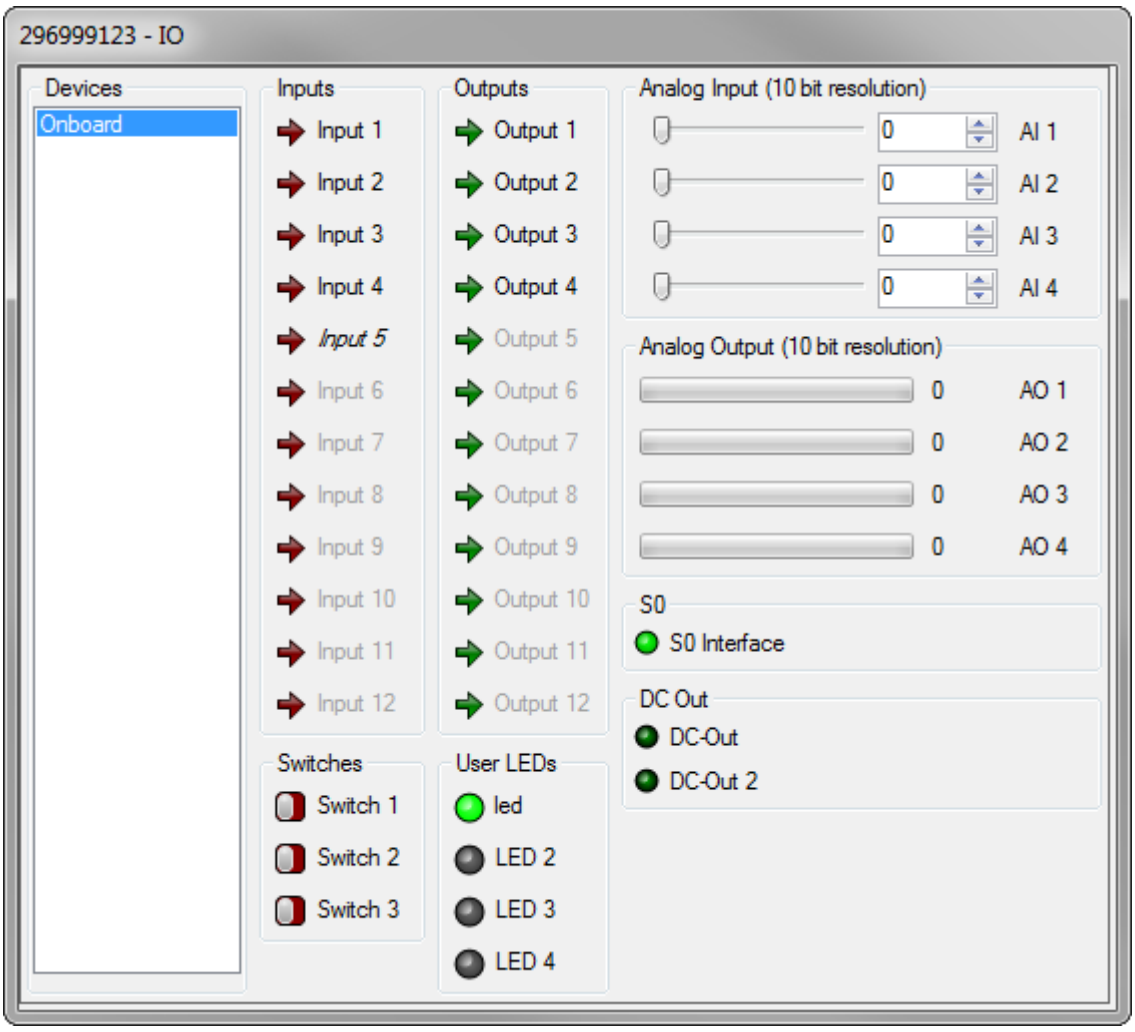
Enable the I/O window by clicking on "IO". This will bring up the following dialog:



After this enable "Memory". This will bring up the following dialog:



Let us start the emulator by clicking on "Load & Run" on the emulator control window. The program of the application is now running. As there has not yet been applied any values to the memory, LED will show as off. Enter a value in entry "1" of the "Memory" to simulate a master writing to holding register 4096 (see MODBUS commands for details) and wait for the LED to turn on. The "IO" window should now look like this:

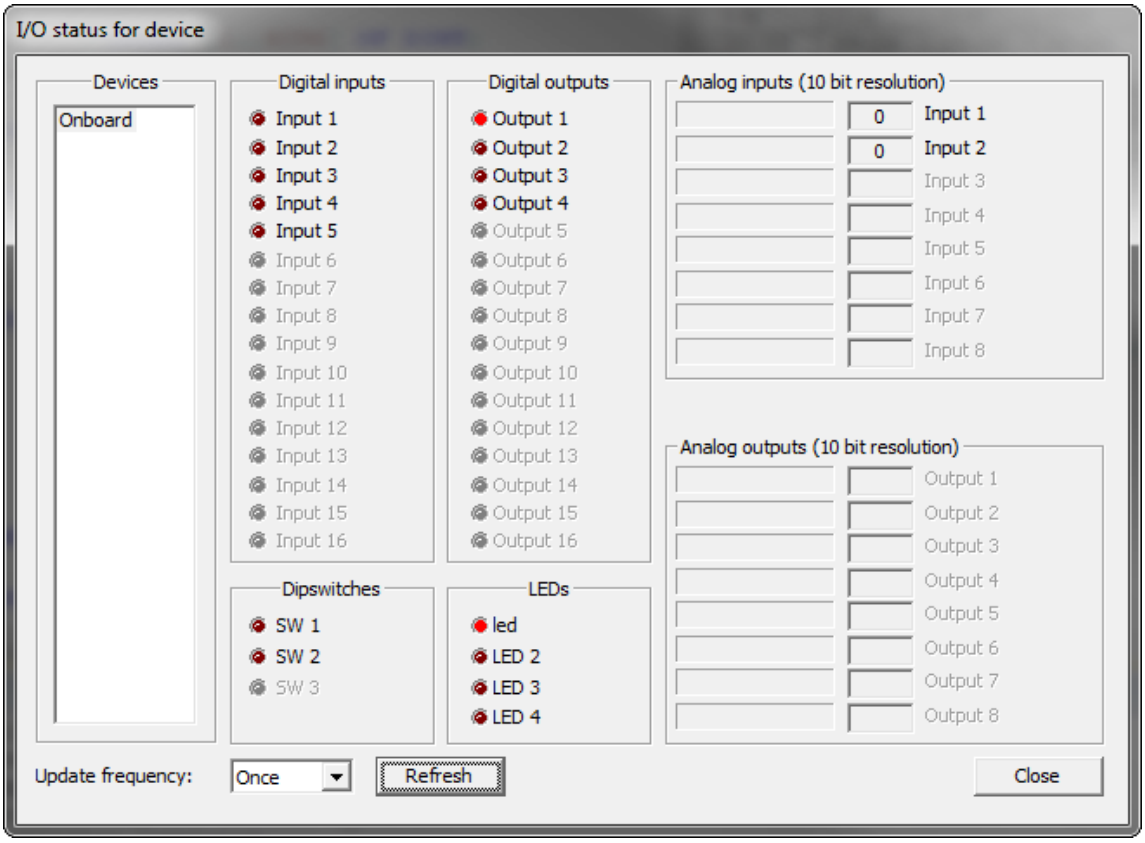


6. Transfer the application to a physical RTCU device

Transferring the application to a physical device is described in the Transfer to RTCU section. As the configured I/O Extension information is an integral part of the project, it will automatically be transferred to the device. If the device attached does not support the I/O Extension information, it will simply be ignored, and the application may not operate as intended.

7. Monitor I/O using the I/O Monitor dialog

The Device I/O Monitor fully supports the onboard I/O and changes from the MODBUS master can easily be monitored. The following figure shows the I/O status of the project in this tutorial running on a physical device after an onboard digital input and a register value have been changed.



This is the end of the tutorial. For further information on I/O Extension dialogs, please consult the sections Project Control - I/O extension, Project Control - I/O extension net, Project Control - I/O extension device, and Project Control - Setup Module.

5.5 Exception handling

The communication between the RTCU device and I/O devices is fully automated, but in some cases the communication can be interrupted. For example if the device is unplugged from the net, or if the device and RTCU device configurations do not match. I/O exception handling is introduced to monitor the devices and report the changes of their status.

Exception handling is done by two VPL commands; **ioWaitException()**, which monitors the net and reports exceptions when the status of one or more devices change, and **ioGetStatus()** which will get the status of each device. Four possible exceptions are covered by the I/O exception handling:

- Device is operational and working correctly .
- Device is unplugged or not responding.
- I/O configuration of the RTCU and the device do not match.
- The device is responding but the response cannot be understood.

Please refer to ioWaitException (Function) and ioGetStatus (Function) for detailed information on the functions.

Even if an exception occurs, the RTCU does not fault but continue to try and update the net. This allows a service technician to exchange a defect device without interrupting the net and the application running on the device. Since the application does not halt, it is possible to react when an exception occurs and for example send an SMS that includes a fault report.

5.6 MODBUS commands

This is an overview of the MODBUS commands that are used by the RTCU device in master mode and accepted by the device in slave mode.

Master

The MODBUS commands that the RTCU device uses to read and write I/O in remote devices.

Command	*Name*	*Description*
02 (0x02)	Reads discrete inputs.	This is used to read the digital inputs from the devices.
04 (0x04)	Reads input registers.	This is used to read the analog inputs from the devices.
15 (0x0F)	Writes multiple coils.	This is used to set the digital outputs for the devices.
16 (0x10)	Writes multiple holding registers.	This is used to set the analog outputs for the devices.

Slave

The MODBUS commands that the RTCU device accepts in slave mode.
The RTCU device will return an error message with exception code 01 (0x01) if it receives a MODBUS command not listed here.

Please note that memory registers in the RTCU device are 32 bits long and are read and written lower half first and then upper half.
Register channel 0x1000 will therefore contain RTCU memory position 1 Byte 1 + 2 and channel 0x1001 will contain position 1 Byte 3 + 4.

The command tables follow this convention:

Field #	Name	Byte count	Value / Comments

01 (0x01) Read coils

Request			
00	Address	1 byte	1 - 247.
01	Function code	1 byte	0x01.
02 - 03	Starting channel	2 bytes	0x0000 - 0x003F for DO onboard RTCU. 0x0040 - 0x023F for DO in extension modules.
04 - 05	Number of channels	2 bytes	0x0001 - 0x0240.

Error response			
00	Address	1 byte	1 - 247.
01	Function code	1 byte	0x81.
02	Exception code	1 byte	0x03 for number of channels too high. 0x02 for when one or more of the channels are not present.

02 (0x02) Read discrete inputs

Request			
00	Address	1 byte	1 - 247.
01	Function code	1 byte	0x02.
02 - 03	Starting channel	2 bytes	0x0000 - 0x003F for DI onboard RTCU. 0x0040 - 0x023F for DI in extension modules.
04 - 05	Number of channels	2 bytes	0x0001 - 0x0240.

Error response

00	Address	1 byte	1 - 247.
01	Function code	1 byte	0x82.
02	Exception code	1 byte	0x03 for number of channels too high. 0x02 for when one or more of the channels are not present.

03 (0x03) Read holding registers

Request

00	Address	1 byte	1 - 247.
01	Function code	1 byte	0x03.
02 - 03	Starting channel	2 bytes	0x0000 - 0x003F for AO onboard RTCU. 0x0040 - 0x013F for AO in extension modules. 0x1000 - 0x17FF for RTCU memory registers.
04 - 05	Number of channels	2 bytes	0x0001 - 0x007D.

Error response

00	Address	1 byte	1 - 247.
01	Function code	1 byte	0x83.
02	Exception code	1 byte	0x03 for number of channels too high. 0x02 for when one or more of the channels are not present.

04 (0x04) Read input registers

Request

00	Address	1 byte	1 - 247.
01	Function code	1 byte	0x04.
02 - 03	Starting channel	2 bytes	0x0000 - 0x003F for AI onboard RTCU. 0x0040 - 0x013F for AI in extension modules. 0x1000 - 0x17FF for RTCU memory registers.
04 - 05	Number of channels	2 bytes	0x0001 - 0x007D.

error response

00	Address	1 byte	1 - 247.
01	Function code	1 byte	0x84.
02	Exception code	1 byte	0x03 for number of channels too high. 0x02 for when one or more of the channels are not present.

05 (0x05) Write single coil

Request

00	Address	1 byte	1 - 247.
01	Function code	1 byte	0x05.
02 - 03	Channel	2 bytes	0x0000 - 0x003F for DO onboard RTCU. 0x0040 - 0x013F for DO in extension modules.
04 - 05	Value	2 bytes	0x0000 or 0xFF00.

error response

00	Address	1 byte	1 - 247.
01	Function code	1 byte	0x85.
02	Exception code	1 byte	0x03 for invalid value. 0x02 for invalid channel.

06 (0x06) Write single holding register

Request

00	Address	1 byte	1 - 247.
01	Function code	1 byte	0x06.
02 - 03	Channel	2 bytes	0x0000 - 0x003F for AO onboard RTCU. 0x0040 - 0x013F for AO in extension modules. 0x1000 - 0x17FF for RTCU memory registers.
04 - 05	Value	2 bytes	0x0000 - 0x03FF for AO onboard RTCU. 0x0000 - 0xFFFF for all other channels.

Error response

00	Address	1 byte	1 - 247.
01	Function code	1 byte	0x83.
02	Exception code	1 byte	0x02 for invalid channel.

15 (0x0F) Write multiple coils

Request

00	Address	1 byte	1 - 247.
01	Function code	1 byte	0x0F.
02 - 03	Starting channel	2 bytes	0x0000 - 0x003F for DO onboard RTCU. 0x0040 - 0x023F for DO in extension modules.
04 - 05	Number of channels	2 bytes	0x0001 - 0x0240.
06	Byte count	1 byte	N.
07 -	Output values	N * 1 byte	Value.

error response

00	Address	1 byte	1 - 247.
01	Function code	1 byte	0x8F.
02	Exception code	1 byte	0x03 for number of channels too high or with invalid byte count 0x02 for when one or more of the channels are not present.

16 (0x10) Write multiple holding registers

Request

00	Address	1 byte	1 - 247.
01	Function code	1 byte	0x10.
02 - 03	Starting channel	2 bytes	0x0000 - 0x003F for AO onboard RTCU. 0x0040 - 0x013F for AO in extension modules. 0x1000 - 0x17FF for RTCU memory registers.
04 - 05	Number of channels	2 bytes	0x0001 - 0x007B.
06	Byte count	1 byte	2 * N.
07 -	Values	N * 2 bytes	Value.

Error response

00	Address	1 byte	1 - 247.
01	Function code	1 byte	0x90.
02	Exception code	1 byte	0x03 for number of channels too high of invalid byte count 0x02 for when one or more of the channels are not present

5.7 Limitations

Because of the limitations on MODBUS protocol, I/O extension addressing is limited to 247. Maximum number of devices that can be connected to the bus simultaneously is limited to 64.

The master device initiates the communication and sends the commands to the slave devices to read or write to their registers. A slave device cannot initiate communication with the master device. The master device needs to poll the inputs on each device in order to update the input status. The update cycle time is therefore increased with the number of devices connected to the net.

The total number of digital input/outputs per device is limited to 16, and analog input/output is limited to 8. This means that digital inputs/outputs can be extended up to 1024 and analog input/outputs can be extended up to 512 when the maximum number of slave devices are connected to the net.

Despite the fact that the use of I/O extension is designed to be similar to use of the native input/output of the RTCU and with the only difference being in the I/O configuration in the project tree, some devices need to be set a defined value instead of the value that would be used to set a native analog output. In such cases, please refer to the technical manual of the device to see which value to use to get an expected value at the output.

Because of limitations on some modules, the maximum baud rate that can be selected is set to 57,600 bps, even if the serial port natively supports baud rates up to 115200 bps.

When I/O extension is used, the configured port cannot be used as an ordinary RS-485/RS-232 port.

* * * * THIS PAGE IS INTENTIONALLY LEFT BLANK * * *

Examples

6 Examples

6.1 Examples

The next pages present a number of example programs demonstrating various aspects of typical RTCU applications. All examples are installed with the RTCU IDE Development Environment and can be found at "My documents\RTCU Projects\Examples".

There is also an online tutorial available that you could follow to get a crash course in the usage of the RTCU IDE environment.

- Simple application - 1
- Simple application - 2
- Greenhouse - 1 (simple)
- Greenhouse - 2 (advanced)
- Send SMS message
- Receive SMS message
- Voice message
- Voice message HQ
- BLE Beacon(LX)
- BLE Scanner(LX)
- Bluetooth Server (NX32L)
- Bluetooth Low Energy (NX32L)
- Datalogger (simple)
- GPS Mobile (simple)
- Remote IO
- REST Example
- REST Basic Authentication Example
- Telnet server
- Telnet server2
- Web client
- VSMS using RCH (simple)
- Mobile tracking (simple)
- Socket communication (advanced)
- RS485 Network - Master
- RS485 Network - Slave
- Thread Example
- MX2 GPSLog
- Navigation Example
- NMP Example
- SMTP Example (tank monitoring)
- MODBUS Network - Slave (simple)
- SNMP Manager
- SNMP Client
- Wired M-Bus
- Wireless M-Bus receiver
- Wireless M-Bus sender

Should you need further examples, please contact support

6.2 Examples - Simple application - 1

```
//-------------------------------------------------------------------------
-
// test.vpl, created 2000-08-16 10:11
//
//-------------------------------------------------------------------------
-
INCLUDE rtcu.inc

//
// This program activates an output when two inputs are active

VAR_INPUT
   i1 : BOOL; | Input 1
   i2 : BOOL; | Input 2
END_VAR;

VAR_OUTPUT
   o1 : BOOL; | Output
END_VAR;

PROGRAM simple1;

BEGIN
   o1 := i1 AND i2;
END;

END_PROGRAM;
```

6.3 Examples - Simple application - 2

```
//-------------------------------------------------------------------------
-
// test2.vpl, created 2000-08-01 20:10
//
//-------------------------------------------------------------------------
-
INCLUDE rtcu.inc

// This program:
//   negates o1 on each leading edge detected on input i1
//   negates o2 on each falling edge detected on input i1.
//

// Next follows all the variables that can be configured via the configuration
dialog
VAR_INPUT
   i1 : BOOL; | Input
END_VAR;

VAR_OUTPUT
   o1 : BOOL; | Output1 (Negated on leading edge)
   o2 : BOOL; | Output2 (Negated on falling edge)
END_VAR;

// Here follows the programs global variables
VAR
   edge : RF_TRIG;
END_VAR;

PROGRAM test;

// The next code will only be executed once after the program starts
BEGIN
   // Code from this point until END will be executed repeatedly
   edge(trig := i1);
   IF edge.rq THEN
      o1 := NOT o1;
   ELSIF edge.fq THEN
      o2 := NOT o2;
   END_IF;
END;

END_PROGRAM;
```

6.4 Examples - Greenhouse - 1 (simple)

The simple Greenhouse example is also the target for the online tutorial.

```
//-------------------------------------------------------------------------
-
// Greenhouse.vpl, created 2000-05-16 09:01
// This program solves the following job:
//
// Gardener Green has a greenhouse in which he wants to control the
temperature.
// For that purpose he has bought a couple of temperature sensors and some
// actuators. One of the temperature sensors will be adjusted to give a
// signal when the temperature falls below a given temperature, and the
// other sensor will give a signal when the temperature rises above a given
// temperature. Using inputs from these sensores he wants to turn on a
// heating unit and turn on the ventilation duct.
// If the system does not succeed in bringing the temperature within the
// given limits, the gardener wants to have a SMS message sent to his
// mobile telephone. This message will notify him about the status of the
// greenhouse.
//-------------------------------------------------------------------------
-
INCLUDE rtcu.inc

VAR_INPUT
   Sensor_L    : BOOL; | Sensorinput for low temperature
   Sensor_H    : BOOL; | Sensorinput for high temperature
   Alarm_time : INT;  | Time, 0..32767 minutes, time before SMS message is
sent
   phone_number : SRING; | Telephone number to send SMS message to
END_VAR;

VAR_OUTPUT
   Out_Heater      : BOOL; | Output to activate heater
   Out_Ventilation : BOOL; | Output to activate ventilation
   online : BOOL; | Is device connected to a GSM basestation
END_VAR;


VAR
   timer : TON; // ON-delay timer is declared.
                // A On-delay timer goes active when 'trig' has
                // been active for 'pt' seconds.
   send_alarm : R_TRIG; // detects leading edge on alarm
END_VAR;

PROGRAM greenhouse;

// Initialization code:
// Set the timer
// (Convert to milliseconds)
timer(pt := Alarm_time*60*1000);

// turn on GSM-module
gsmPower(power := ON);

BEGIN

   // Update status for connected to GSM base station
   online := gsmConnected();
```

```
   // If the temperature is to high then activate ventilation:
   IF Sensor_H THEN
      Out_Ventilation:= ON;
   ELSE
      Out_Ventilation:= OFF;
   END_IF;

   // If the temperature is to low then activate the heater:
   IF Sensor_L THEN
      Out_Heater:= ON;
   ELSE
      Out_Heater:= OFF;
   END_IF;

   // Start timer on the leading edge of the sensor-inputs:
   timer(trig:= Sensor_L OR Sensor_H);

   // Detect leading edge on alarm:
   send_alarm(trig:=timer.q);

   IF send_alarm.q THEN
      IF Sensor_L THEN
         // send SMS for temperature too low
         gsmSendSMS(phonenumber:=phone_number, message:="Temperature too
low");
      ELSIF Sensor_H THEN
         // send SMS for temperature too high
         gsmSendSMS(phonenumber:=phone_number, message:="Temperature too
high");
      END_IF;
   END_IF;
END;

END_PROGRAM;
```

6.5 Examples - Greenhouse - 2 (advanced)

```
//-------------------------------------------------------------------------
// This program extends the basic Greenhouse control program in that
// a connected lamp in the greenhouse can be turned on/off in several
// different ways:
// -Sending an SMS message to the device. By sending the message
// "ON" the lamp will turn on. Sending the message "OFF" will turn
// off the lamp.
//
// -Calling the device and control it by voice/keypress interaction
// (Voice Response System).
// When the device answers the call it will greet the caller with
// the message:
// "Press 1 to activate lamp, press 0 to deactivate the lamp"
// The caller can then press '0' or '1'. Any other key wil result
// in the message: "Illegal keypress"
// When a legal command is received the device says:
// "Have a nice day".
//
// The conneced lamp must be initially turned on when the device powers up.
//
// In addition to the lamp there is also a door-contact used for detecting
// when the door is opened. If this happens and the feature is enabled the
// device will call the gardener and say "Burgler in the greenhouse"
// This alarmfeature can be enabled/disabled with a Keyswitch that is mounted
// at the entrance to the greenhouse.
//-------------------------------------------------------------------------
INCLUDE rtcu.inc

VAR_INPUT
   Sensor_L    : BOOL; | Sensorinput for low temperature
   Sensor_H    : BOOL; | Sensorinput for high temperature
   Alarm_time : INT;  | time, 1..32767 minutter, before SMS message is sent

   Doorcontact     : BOOL R_EDGE;  | Doorcontact that will ignite the burglar-
alarm
   Alarm_keyswitch: BOOL;          | Is the burglar-alarm activated?
   PhoneNo         : STRING;       | Phonenumber to call when burglar-alarm
occurs
END_VAR;

VAR_OUTPUT
   Out_Heater      : BOOL; | Output to activate heater
   Out_Ventilation: BOOL; | Output to activate ventilation
   Lamp       : BOOL:=ON;  | Lamp (initially ON)
   Connected : BOOL;       | The device is connected to the GSM-network.
   OffHook    : BOOL;      | "off-hook" condition.
END_VAR;

VAR
   timer : TON; // ON-delay timer is declared.
                // An On-delay timer goes active when 'trig' has
                // been active for 'pt' seconds.
   send_alarm : R_TRIG; // detects leading edge on alarm
   state      : INT := 1; // State for Voice call
   sms        : gsmIncomingSMS;
   voicecall : gsmIncomingCall;
END_VAR;


//-------------------------------------------------------------------------
```

```
// Control the temperature in the greenhouse, and send SMS message
// if temperature is outside limits for a specified period
//--------------------------------------------------------------------------
FUNCTION TemperatureControl;

   // if sensor for high temp. is activated we turn on the ventilation:
   Out_Ventilation := Sensor_H;

   // if sensor for low temp. is activated we turn on the heater:
   Out_Heater := Sensor_L;

   // Start timer on the leading edge of the sensor-inputs:
   timer(trig:= Sensor_L OR Sensor_H);

   // Detect leading edge on alarm:
   send_alarm(trig:=timer.q);

   IF send_alarm.q THEN
      IF Sensor_L THEN
         // send sms for temperature too low
         gsmSendSMS(phonenumber:=PhoneNo, message:="Temperature too low");
      ELSIF Sensor_H THEN
         // send sms for temperature too high
         gsmSendSMS(phonenumber:=PhoneNo, message:="Temperature too high");
      END_IF;
   END_IF;

END_FUNCTION;


//--------------------------------------------------------------------------
// Control the lamp in the greenhouse using a SMS message
//--------------------------------------------------------------------------
FUNCTION SMSLamp;
VAR
   tmpstr : string;
END_VAR;

   // Check if we have received a SMS message:
   IF sms.status > 0 THEN
      // SMS message received
      // If it's an "ON" message
      // (Note: strCompare doesn't check the case by default)
      tmpstr:=strRemoveSpaces(str:=sms.message);
      IF strCompare(str1:=tmpstr, str2:="ON")=0 THEN
         lamp:=ON;
      // Else if it's an "OFF" message
      ELSIF strCompare(str1:=tmpstr, str2:="OFF")=0 THEN
         lamp:=OFF;
      END_IF;
   END_IF;

END_FUNCTION;


//--------------------------------------------------------------------------
// Control the lamp in the greenhouse using a simple voice-response system
//--------------------------------------------------------------------------
FUNCTION VoiceLamp;
VAR
   key    : INT; // key selected in voice-menu
END_VAR;

   // Voice/response:
```

```
   CASE state OF
      1: // waiting for call:
         IF voicecall.status > 0 THEN
            gsmAnswer();
            state:=2;
         END_IF;

      2: // Select on/off
         voiceTalk(message:="SelOnOff.wav");
         key :=dtmfGetKey(timeout:=7000);
         IF key=0 OR key=1 THEN
            lamp:=BOOL(key);
            state:=3;
         ELSIF key <> -1 THEN
            voiceTalk(message:="Illegal.wav");
         END_IF;

      3: // Command done
         voiceTalk(message:="AllDone.wav");
         gsmHangup();
         state:=1;
   END_CASE;

   // if the user has terminated the connection we force the state back to
   // the inital state (waiting for call)
   IF NOT gsmOffHook() THEN
      state:=1;
   END_IF;
END_FUNCTION;


//-------------------------------------------------------------------------------
// Detect intrusion in the greenhouse, and send an SMS message
//-------------------------------------------------------------------------------
FUNCTION BurglerAlarm;
VAR
   j : INT;
END_VAR;

   // If burglaralarm is activated and we have an alarm:
   IF Alarm_keyswitch AND Doorcontact THEN
      // We will try 3 times to deliver the alarm:
      FOR j:=1 TO 3 DO
         // If call is being answered
         IF gsmMakeCall(phonenumber:=PhoneNo) THEN
            // Deliver the message
            voiceTalk(message:="Burglar.wav");
            gsmHangup();
            EXIT;
         END_IF;
      END_FOR;
   END_IF;
END_FUNCTION;


//-------------------------------------------------------------------------------
// The main program. This just calls the various functions sequentially.
//-------------------------------------------------------------------------------
PROGRAM greenhouse2;

// Initialization code:

// Set the timer
// (Convert to seconds)
```

```
timer(pt := Alarm_time*60*1000);

// turn on GSM-module
gsmPower(power := ON);

BEGIN
   // Update Function blocks
   timer();
   sms();
   voicecall();

   OffHook := gsmOffHook();
   Connected := gsmConnected();

   // Control the temperature
   TemperatureControl();

   // Control lamp using SMS messages
   SMSLamp();

   // Control lamp using Voice-response system
   VoiceLamp();

   // Detect burgler alarm
   BurglerAlarm();
END;

END_PROGRAM;
```

6.6 Examples - Send SMS message

```
//-------------------------------------------------------------------------
-
// Send SMS.vpl, created 2000-08-06 02:15
//
// This program will count the number of leading edges on a digital input.
// After each leading edge, a SMS message will be sent with information on
// the number of leading edges detected so far.
//-------------------------------------------------------------------------
-
INCLUDE rtcu.inc

// Next follows all the variables that can be configured via the configuration
dialog
VAR_INPUT
   input : BOOL R_EDGE;
END_VAR;

VAR_OUTPUT
   Connected : BOOL; | Connection to GSM-network.
END_VAR;

// Here follows the programs global variables
VAR
   edge_no         : INT := 0;
   message_to_send : STRING;
END_VAR;

PROGRAM Send_SMS;
// The next code will only be executed once after the program starts
// Controls power to the GSM module
gsmPower(power := ON);

BEGIN
   // Code from this point until END will be executed repeatedly
   Connected := gsmConnected();

   IF input THEN
      edge_no := edge_no + 1;
      // Send an SMS message
      message_to_send:=strFormat(format:="\1 leading edges detected",
                                 v1:=edge_no);
      gsmSendSMS(phonenumber:="+44 1234 5678",
                 message:=message_to_send);
   END_IF;
END;

END_PROGRAM;
```

6.7 Examples - Receive SMS message

```
//------------------------------------------------------------------------------
// Receive SMS.vpl, created 2000-08-06 11:14
//
// This program allows control of 8 outputs by sending a SMS-message
// to the RTCU device. "ON" followed by a number in the interval 1..8
activates
// the specified output.
// "OFF" followed by a number in the interval 1..8 deactivates the specified
// output.
//------------------------------------------------------------------------------
INCLUDE rtcu.inc

VAR_OUTPUT
   outputs      : ARRAY[1..8] OF BOOL; | All 8 outputs
   connected    : BOOL;                | Connected to a GSM basestation.
END_VAR;

VAR
   sms     : gsmIncomingSMS; // Receives incoming SMS messages
   extract : strGetValues; // Matching of strings
END_VAR;

PROGRAM ReceiveSMS;

// The next code will only be executed once after the program starts
gsmPower(power:=ON);

BEGIN
   sms();
   connected := gsmConnected();

   IF sms.status > 0 THEN
      // Message received
      // Check if it is a "ON" message
      extract(format:="ON \1", str:=sms.message);
      IF extract.match AND extract.v1 >= 1 AND extract.v1 <= 8 THEN
         outputs[extract.v1] := ON;
      END_IF;

      // Check if it is a "OFF" message
      extract(format:="OFF \1", str:=sms.message);
      IF extract.match AND extract.v1 >= 1 AND extract.v1 <= 8 THEN
         outputs[extract.v1] := OFF;
      END_IF;
      DebugMsg(message:=strConcat(str1:="Message received from ",
str2:=sms.phonenumber));
      DebugMsg(message:=strConcat(str1:="Message is ", str2:=sms.message));
   END_IF;
END;

END_PROGRAM;
```

6.8 Examples - Voice message

```
//--------------------------------------------------------------------------
// This program allows control of the 8 digital outputs by using
// voice response principle.
//--------------------------------------------------------------------------
INCLUDE rtcu.inc

// Next follows all the variables that can be configured via the configuration
dialog
VAR_OUTPUT
   connected : BOOL; | Connected to GSM Basestation.
   offhook   : BOOL; | off-hook signal
   outputs   : ARRAY[1..8] OF BOOL; | Outputs we control
END_VAR;

VAR
   no      : INT;
   command : INT;
   state   : SINT := 1;
   incoming: gsmIncomingCall;
END_VAR;

PROGRAM SetOutput;

gsmPower(power:=ON);

BEGIN
   // Update Function block
   incoming();

   // Indicate if we are connected to a GSM Basestation
   connected := gsmConnected();

   CASE state OF
      1: // Waiting for incoming call:
         IF incoming.status > 0 THEN
            DebugMsg(message:="Incoming call, answering");
            gsmAnswer();
            state:=2;
         END_IF;

      2: // Select output number:
         voiceTalk(message:="SelOutNo.wav");
         no:=dtmfGetKey(timeout:=5000);
         IF no>=1 AND no<=8 THEN
            state:=3;
            DebugFmt(message:="Output number \1 selected", v1:=no);
         END_IF;

      3: // Select on/off:
         voiceTalk(message:="SelOnOff.wav");
         command:=dtmfGetKey(timeout:=5000);
         IF command=0 OR command=1 THEN
            outputs[no]:=BOOL(command);
            DebugFmt(message:="Output number \1 set to \2", v1:=no,
v2:=command);
            state:=2;
         ELSIF command=10 THEN
            state:=2;
         END_IF;
   END_CASE;
```

```
   // Are we still "online":
   offhook := gsmOffHook();

   // if the user has terminated the connection we go back
   // to the initial state (waiting for call)
   IF NOT offhook THEN
      state:=1;
   END_IF;
END;

END_PROGRAM;
```

6.9 Examples - Voice Message HQ

```
//-------------------------------------------------------------------------------
// Simple program that plays HQ audio for the caller.
//-------------------------------------------------------------------------------
INCLUDE rtcu.inc

VAR_OUTPUT
  connected : BOOL; | Connected to GSM Basestation.
  offhook   : BOOL; | off-hook signal
END_VAR;

VAR
  state   : SINT := 1;
  incoming: gsmIncomingCall;
END_VAR;

PROGRAM voicemsg;

fsMediaOpen(media:=1);
gsmPower(power:=ON);

BEGIN
  incoming();
  connected := gsmConnected();

  CASE state OF
    1: // Waiting for incoming call:
    IF incoming.status > 0 THEN
      DebugMsg(message:="Incoming call, answering");
      gsmAnswer();
      state:=2;
    END_IF;

    2: // Play part of an audio book
    voiceTalk(message:="B:\voice\babbage.ogg",pause:=1000);
    IF dtmfGetKey(timeout:=5000)<>-1 THEN
       gsmHangup();
    END_IF;
  END_CASE;

  offhook := gsmOffHook();
  IF NOT offhook THEN
    state:=1;
  END_IF;
END;

END_PROGRAM;
```

6.10 Examples - BLE Beacon (LX)

```
//-------------------------------------------------------------------------
-
// beacon.vpl
// This example scans for BLE devices until it finds one matching the wanted
UUID in the iBeacon data.
// It then adds it to the accept list and switches to only receiving from
devices on the acccept list.
// The example is configured to look for an BlueUp SafeX beacon, see
https://www.blueupbeacons.com/index.php?page=safex
// The beacon is configured to use Safety Packets which include the status of
the button and other sensors.
//
// By sending an SMS with the text "connect" to the device, it will connect to
the beacon,
// show the battery level and disconnect again.
//
//-------------------------------------------------------------------------
-
INCLUDE rtcu.inc
// Uncomment math.inc to add math library support.
//INCLUDE math.inc

//  Input variables that can be configured via the configuration dialog (These
are global)
VAR_INPUT

END_VAR;

//  Output variables that can be configured via the configuration dialog
(These are global)
VAR_OUTPUT

END_VAR;

//  These are the global variables of the program
VAR
   target_uuid : STRING := "A565CC84-1E0E-4B5A-B2D3-17C98FF5410A";
   target_MAC  : STRING := "";

   scanning    : BOOL := TRUE;
END_VAR;


// Read Big endian 16-bit integer from pointer
FUNCTION readIntBE:INT;
VAR_INPUT
   p   : PTR;
END_VAR;
VAR
   tmp : INT;
END_VAR;
   memcpy(dst:=ADDR(tmp)+1, src:=p, len:=1);
   memcpy(dst:=ADDR(tmp), src:=p+1, len:=1);
   readIntBE := tmp;
END_FUNCTION;

// Extract UUID from pointer, using the format used in the beacon.
FUNCTION parseProxUuid:STRING;
VAR_INPUT
   p     : PTR;
```

```
END_VAR;
VAR
   str    : STRING;
   data   : ARRAY[0..16] OF USINT;
   i      : INT;
END_VAR;
   memcpy(dst:=ADDR(data), src:=p, len:=16);
   str := sintToHex(v:=data[0])+sintToHex(v:=data[1])+sintToHex(v:=data[2])
+sintToHex(v:=data[3])
      +"-"+sintToHex(v:=data[4])+sintToHex(v:=data[5])
+"-"+sintToHex(v:=data[6])+sintToHex(v:=data[7])
      +"-"+sintToHex(v:=data[8])+sintToHex(v:=data[9])
+"-"+sintToHex(v:=data[10])+sintToHex(v:=data[11])
      +sintToHex(v:=data[12])+sintToHex(v:=data[13])+sintToHex(v:=data[14])
+sintToHex(v:=data[15]);
   parseProxUuid := str;
END_FUNCTION;


// Parse proximity beacon.
// Returns 1 if target_uuid is found, otherwise 0.
FUNCTION ParseProx : INT;
VAR_INPUT
   mac          : STRING;
   p            : PTR;
   len          : INT;
END_VAR;
VAR
   uuid         : STRING;
   major        : UINT;
   minor        : UINT;
   meas_power   : SINT;
END_VAR;
   ParseProx := 0;
   // Extract values:
   uuid := parseProxUuid(p:=p);
   major := readIntBE(p:=p+16);
   minor := readIntBE(p:=p+18);
   memcpy(dst:=ADDR(meas_power), src:=p+20, len:=1);
   // show values
   DebugFmt(message:=" "+mac);
   DebugFmt(message:="   "+uuid);
   DebugFmt(message:="   Major: "+intToHex(v:=major));
   DebugFmt(message:="   Minor: "+intToHex(v:=minor));
   DebugFmt(message:="   Measured power: \1 dB", v1:=meas_power );
   IF uuid = target_uuid THEN
      // SafeX beacon found: Show the status for it.
      DebugFmt(message:="    Button short pushed: \1", v1:=(minor AND 1));
      DebugFmt(message:="    Button long pushed: \1", v1:=(minor AND 2));
      DebugFmt(message:="    Movement: \1", v1:=(minor AND 4));
      DebugFmt(message:="    Free Fall: \1", v1:=(minor AND 8));
      ParseProx := 1;
   END_IF;
END_FUNCTION;

// Parses beacon
// Returns 1 if target_uuid is found, otherwise 0.
FUNCTION ParseBeacon : INT;
VAR_INPUT
   mac          : STRING;
   p            : PTR;
   len          : INT;
END_VAR;
VAR
```

```
   b_type      : UINT;
END_VAR;
   ParseBeacon := 0;
   // Read beacon type
   memcpy(dst:=ADDR(b_type), src:=p, len:=2);

   IF b_type = 16#1502 THEN
      // proximity beacon
      ParseBeacon := ParseProx(mac:= mac, p:=p+2, len:=len-2);
   END_IF;

END_FUNCTION;



// Callback that parses manufacturer specific data.
FUNCTION CALLBACK cbAdvMan;
VAR_INPUT
   mac      : STRING;
   ev_type  : UINT;
   adv_type : USINT;
   data     : PTR;
   len      : INT;
   rssi     : SINT;
   arg      : DINT;
END_VAR;
VAR
   cid      : UINT;
   rc       : INT;
END_VAR;
   IF adv_type = 16#FF THEN
      // Extract Company Identifier
      memcpy(dst:=ADDR(cid), src:=data, len:=2);

      IF cid = 16#004c THEN //Apple => iBeacon

         // Parse beacon, searching for the wanted UUID
         rc := ParseBeacon(mac:=mac, p:=data+2, len := len-2);
         IF rc = 1 AND scanning THEN
            // Device found, restart scan to use accept filter that only
accepts this device.
            target_MAC := MAC;
            // Stop scan
            rc := bleObserverStop();
            DebugFmt(message:="bleObserverStop: \1", v1:=rc);
            scanning := FALSE;
            // Add MAC to accept list
            rc := bleAcceptFilterAdd(mac:=target_MAC);
            DebugFmt(message:="bleAcceptFilterAdd: \1", v1:=rc);

            // Restart scan with new filter policy
            rc := bleObserverStart(
               scan_type:=1,// Active
               filter_policy := 1//Accept only from accept list
               );
            DebugFmt(message:="bleObserverStart: \1", v1:=rc);
         END_IF;
      END_IF;
   END_IF;
END_FUNCTION;



// Thread for connecting to a device, reading the battery status and
```

```
disconnecting.
THREAD_BLOCK beaconThread;
VAR
   rc, rc2      : INT;
   i, j         : INT;
   s, c         : UINT;
   uuid         : STRING;
   primary      : BOOL;
   flags        : DINT;
   dev          : SYSHANDLE;
   bat_level_id : UINT;
   bat_level    : USINT;
   size         : INT;

   state        : INT := 0;
   old_state    : INT:= -1;
   running      : BOOL;
END_VAR;
   DebugMsg(message:="Starting beacon thread");
   running := TRUE;

   // Stop scanning so the connection can be created
   rc := bleObserverStop();
   DebugFmt(message:="bleObserverStop: \1", v1:=rc);

   WHILE running DO
      IF old_state <> state THEN
         // Show state changes
         DebugFmt(message:="S: \1 => \2", v1:=old_state, v2:=state);
         old_state:=state;
      END_IF;

      CASE state OF
      0:
         // not connected, start connection
         DebugFmt(message:="Connect...");
         rc := bleConnect(dev:=dev, mac:=target_mac);
         DebugFmt(message:="bleConnect: \1", v1:=rc);
         IF rc = _BLE_OK THEN
            state := 1;
         ELSE
            state := state+100;
         END_IF;
      1:
         // bleConnect succeeded, check status
         rc := bleDeviceStatus(dev:=dev);
         DebugFmt(message:="bleDeviceStatus: \1, "+bleDeviceMacGet(dev:=dev),
v1:=rc);
         IF rc = 2 THEN
            // Connection lost, switch to disconnect state
            DebugFmt(message:="connection lost");
            state := 10;
         ELSIF rc = 5 THEN
            // Connecting
            // Try again
            state := state+100;
         ELSIF rc = 10 THEN
            // Connected, waiting to be ready, so try again
            state := state+100;
         ELSIF rc = 20 THEN
            // Ready, no cache
            state := 2;
         ELSIF rc > 20 THEN
            // ready, cache present, so skip reading cache
```

```
            state := 3;
         END_IF;
      2:
         // Ready, try reading cache
         rc := bleServiceCacheUpdate(dev:=dev);
         DebugFmt(message:="bleServiceCacheUpdate: \1", v1:=rc);
         IF rc = _BLE_OK THEN
            state := 3;
         ELSE
            state := 10;
         END_IF;
      3:
         // Find wanted characteristic
         state := state+100;

         // Battery level characteristic
         rc := bleCharFind(dev:=dev, uuid:="00002a19-0000-1000-8000-
00805f9b34fb", char:=c, flags := flags);
         DebugFmt(message:="bleCharFind: \1, Char: \2, flags: \4", v1:=rc,
v2:=c, v4:=flags);
         IF rc = _BLE_OK THEN
            bat_level_id := c;
            state := 4;
         END_IF;
      4:
         // Read from characteristic
         size:=1;
         rc := bleReadVal(dev:=dev, char:=bat_level_id, size := size, data :=
ADDR(bat_level));
         DebugFmt(message:="bleReadVal(\2): \1", v1:=rc, v2:=bat_level_id);
         IF rc = 1 THEN
            DebugFmt(message:="Battery level: \1 %", v1:=bat_level);
            state := 5;
         ELSIF rc = -19 THEN
            // Disconnected
            DebugFmt(message:="connection lost");
            state := 10;
         ELSE
            state := state+100;
         END_IF;
      5:
         // We are done, switch to disconnect.
         state := 10;

      10:
         // Disconnect
         DebugMsg(message:="Disconnecting...");
         rc := bleDisConnect(dev:=dev);
         DebugFmt(message:="bleDisconnect: \1", v1:=rc);

         IF rc = _BLE_OK THEN
            // Stop thread
            running := FALSE;
            state := 100;
         ELSE
            state := state+100;
         END_IF;
      END_CASE;

      IF state >= 100 THEN
         // step failed; wait 5s and try again
         state := state -100;
         Sleep(delay:=5000);
      END_IF;
```

```
   END_WHILE;

   // Start scanner again
   rc := bleObserverStart(
      scan_type:=1,// Active
      filter_policy := 1//Accept only from accept list
      );
   DebugFmt(message:="bleObserverStart: \1", v1:=rc);

END_THREAD_BLOCK;

VAR
   beaconThr:beaconThread;
END_VAR;


PROGRAM beacon;
// These are the local variables of the program block
VAR
   rc  : INT;
   sms : gsmIncomingSMS;
END_VAR;
// The next code will only be executed once after the program starts

   // Enable BT
   rc := blePower(power:=ON);
   DebugFmt(message:="blePower: \1", v1:=rc);

   rc := bleAcceptFilterClear();
   DebugFmt(message:="bleAcceptFilterClear: \1", v1:=rc);

   // Register callback for advertising data type 16#FF, Manufacturer specific
data
   rc := bleRegisterAdvData(cb_Adv:=@cbAdvMan, adv_type := 16#FF);
   DebugFmt(message:="BleRegisterAdvData: \1", v1:=rc);



   // Start listening
   rc := bleObserverStart(
      scan_type:=1,// Active
      filter_policy := 0//Accept all
      );
   DebugFmt(message:="bleObserverStart: \1", v1:=rc);


BEGIN
// Code from this point until END will be executed repeatedly
   // Check for command
   sms();
   IF sms.status > 0 THEN
      IF sms.message = "connect" THEN
         IF target_MAC <> "" THEN
            // Start thread to connect to the target
            beaconThr();
         ELSE
            DebugMsg(message:="Target not found");
         END_IF;
      END_IF;
   END_IF;

END;
```

```
END_PROGRAM;
```

6.11 Examples - BLE Scanner (LX)

```
//------------------------------------------------------------------------
-
// ex_scan.vpl
// This example starts scanning for BLE advertising and prints out all
// advertising data that contains the Complete Local Name.
//
//------------------------------------------------------------------------
-
INCLUDE rtcu.inc
// Uncomment math.inc to add math library support.
//INCLUDE math.inc

//  Input variables that can be configured via the configuration dialog (These
are global)
VAR_INPUT

END_VAR;

//  Output variables that can be configured via the configuration dialog
(These are global)
VAR_OUTPUT

END_VAR;

//  These are the global variables of the program
VAR

END_VAR;

// Callback for showing the name of the device
FUNCTION CALLBACK cbAdvName;
VAR_INPUT
   mac      : STRING;
   ev_type  : UINT;
   adv_type : USINT;
   data     : PTR;
   len      : INT;
   rssi     : SINT;
   arg      : DINT;
END_VAR;
VAR
   str      : STRING;
END_VAR;
   str:=strFromMemory(src:=data, len:=len);
   // Show MAC, name and RSSI
   DebugFmt(message:=mac+" (\1): "+str, v1:=rssi);
END_FUNCTION;



PROGRAM ex_scan;
// These are the local variables of the program block
VAR
   rc : INT;
END_VAR;
// The next code will only be executed once after the program starts
   // Enable BT
   rc := blePower(power:=ON);
   DebugFmt(message:="blePower: \1", v1:=rc);
```

```
   // Register callback for advertising data type 16#09, Complete Local Name
   rc := bleRegisterAdvData(cb_Adv:=@cbAdvName, adv_type := 16#09);
   DebugFmt(message:="BleRegisterAdvData: \1", v1:=rc);

   // Start listening
   rc := bleObserverStart(
      scan_type:=1,// Active
      filter_policy := 0//Accept all
      );
   DebugFmt(message:="bleObserverStart: \1", v1:=rc);
BEGIN
   // Code from this point until END will be executed repeatedly
   Sleep(delay:=10000);

END;

END_PROGRAM;
```

6.12 Examples - Bluetooth Server (NX32L)

```
//-------------------------------------------------------------------------
-
// bt_server.vpl, created 2018-03-15 11:01
//
//-------------------------------------------------------------------------
-
INCLUDE rtcu.inc
// Uncomment math.inc to add math library support.
//INCLUDE math.inc


#DEFINE STATE_NOT_CONNECTED   0
#DEFINE STATE_CONNECTING      1
#DEFINE STATE_CONNECTED       2

//  Input variables that can be configured via the configuration dialog (These
are global)
VAR_INPUT
   in      : ARRAY [1..4] OF BOOL; | Input signals to show to the user
   visible : BOOL;                 | Device is visible while this signal is
active
END_VAR;

//  Output variables that can be configured via the configuration dialog
(These are global)
VAR_OUTPUT
   out     : ARRAY [1..4] OF BOOL; | Output signals that are controllable by
the user
END_VAR;

//  These are the global variables of the program
VAR
   connection_state : INT := STATE_NOT_CONNECTED;
   port: SINT;
END_VAR;



THREAD_BLOCK btThread
VAR
   rc      : INT;
   address : STRING;
   ch      : SINT;
   pass    : DINT;
END_VAR;
   WHILE TRUE DO
      //
      rc := btWaitEvent(timeout:=10000, dev := address);
      IF rc <> _BT_ERR_TIMEOUT THEN
         DebugFmt(message:="event \1: "+address, v1:=rc);
         CASE rc OF
         _BT_EVENT_INCOMING:
            rc := btHandleSerialPortIncomingConnection(ch := ch, port :=
port);
            DebugFmt(message:="btHandleSerialPortIncomingConnection(\2) : \1,
\3", v1:=rc, v2:=ch, v3:=port);
            IF(ch = 6) THEN
               IF connection_state <> STATE_NOT_CONNECTED THEN
                  // Close old port if it is open
                  serClose(port:=port);
```

```
                END_IF;

                // Open new port
                rc := serOpen(port:=port);
                DebugFmt(message:="serOpen(\2) : \1", v1:=rc, v2:=port);
                IF rc = 0 THEN
                   connection_state := STATE_CONNECTING;
                END_IF;

             END_IF;
          _BT_EVENT_DEV_FOUND:
             DebugFmt(message:="Found "+address);
          _BT_EVENT_DEV_LOST:
             DebugFmt(message:="Lost "+address);
          _BT_EVENT_PAIR_REQ:
             DebugFmt(message:="Requested confirm for "+address);
             rc := btHandlePairRequest(passkey:=pass);
             DebugFmt(message:="Pass: \4, \1", v1:=rc, v4:=pass);
             rc := guiShowMessage(message:="Is the pairing code
"+dintToStr(v:=pass)+" correct?", type := 2, timeout := 20);
             DebugFmt(message:="guiShowMessage: \1", v1:=rc);
             // Accept if "Yes" was pressed
             rc := btSendPairResponse(accept:=(rc = 3));
             DebugFmt(message:="btSendPairResponse: \1", v1:=rc);
          ELSE
             DebugFmt(message:="unknown event: \1", v1:=rc);
          END_CASE;
          btEventDone();
       END_IF;

   END_WHILE;
END_THREAD_BLOCK;
```

```
PROGRAM bt_server;
// These are the local variables of the program block
VAR
   old_visible : BOOL;
   rc          : INT;
   thEvent     : btThread;
   RX          : serFrameReceiver;
   rxbuffer    : SINT; // Declare a buffer for receiving frames from the
serial port

END_VAR;

// The next code will only be executed once after the program starts

   // Turn on the display for use when pairing
   displayPower(power := ON);

   // Turn on power
   rc := btPower();
   DebugFmt(message:="btPower: \1", v1:=rc);


   thEvent();

   // Enable serial port connections
   rc := btSerialPortProfileEnable(ch:=6, max_con := 1);
   DebugFmt(message:="btSerialPortProfileEnable: \1", v1:=rc);
```

```
   // Make sure discoverable will be set immediately.
   old_visible := NOT visible;

BEGIN
// Code from this point until END will be executed repeatedly

   IF old_visible <> visible THEN
      rc := btMakeDiscoverable(discoverable := visible, timeout := 0);
      DebugFmt(message:="btMakeDiscoverable(\2): \1", v1:=rc, v2:=
INT(visible));
      old_visible := visible;
   END_IF;

   IF connection_state = STATE_CONNECTING THEN
      RX(port:=port, enable:=TRUE, frame:=ADDR(rxbuffer), maxSize:=1);
      connection_state := STATE_CONNECTED;
      // Send instructions
      serSendString(port:=port, str:="Send r to read values, 1-4 to toggle
output$N");
      serSendString(port:=port, str:="Send A-D to set output high, a-b to set
output low$N");

   END_IF;

   IF connection_state = STATE_CONNECTED THEN
      RX();
      IF RX.ready THEN
         // Indicate a frame has been received, data is available in
'rxbuffer', length in 'RX.size'
         DebugFmt(message:="CMD: \1", v1:=rxbuffer);
         IF rxbuffer = 16#72 THEN
            // "r"
            serSendString(port:=port, str:=strFormat(format:="IN: \1,\2,\3,
\4$N",v1:=INT(in[1]),v2:=INT(in[2]),v3:=INT(in[3]),v4:=INT(in[4])));

         END_IF;
         IF rxbuffer >= 16#31 AND rxbuffer<= 16#34 THEN
            // "1"-"4"
            // Toggle output
            out[rxbuffer-16#30]:=NOT out[rxbuffer-16#30];
         END_IF;
         IF rxbuffer >= 16#41 AND rxbuffer<= 16#44 THEN
            // "A"-"D"
            // Set output high
            out[rxbuffer-16#40]:= ON;
         END_IF;
         IF rxbuffer >= 16#61 AND rxbuffer<= 16#64 THEN
            // "a"-"d"
            // Set output low
            out[rxbuffer-16#60]:= OFF;
         END_IF;
         // Release the buffer for the receiver as we are now finished with
the received data
         serFrameReceiveDone(port:=port);
      END_IF;
   END_IF;
END;

END_PROGRAM;
```

6.13 Examples - Bluetooth Low Energy (NX32L)

```
//--------------------------------------------------------------------------
-
// iTag.vpl, created 2018-05-09 11:20
// The iTag button is a BLE keychain device that is typically used as
tracker/key finder.
// It has a single button and a buzzer which is used for alerting.
//
// This application scans for and connects to all found iTag buttons.
// When the button is pressed on any connected iTag, all the iTags will be
alerted,
// one at a time for a short moment.
//
//--------------------------------------------------------------------------
-
INCLUDE rtcu.inc
// Uncomment math.inc to add math library support.
//INCLUDE math.inc

#DEFINE TAG_COUNT 8

// Tag states
#DEFINE ST_DISCON 0
#DEFINE ST_FOUND  1
#DEFINE ST_CON    2

// Service and characteristic UUID for the button service.
#DEFINE ITAG_SERVICE_UUID "0000ffe0-0000-1000-8000-00805f9b34fb"
#DEFINE ITAG_NOTIFY_UUID "0000ffe1-0000-1000-8000-00805f9b34fb"

// Service and characteristic UUID for the immediate alert service.
#DEFINE ALERT_SERVICE_UUID  "00001802-0000-1000-8000-00805f9b34fb"
#DEFINE ALERT_LEVEL_UUID    "00002a06-0000-1000-8000-00805f9b34fb"

// Struct block with data for a tag
STRUCT_BLOCK iTag_data;
   // Address of the tag
   address     : STRING;
   // Service and characteristic ID for the alert service
   alert_s     : INT;
   alert_c     : INT;
   // Tag state
   status      : INT;
END_STRUCT_BLOCK;

//  Input variables that can be configured via the configuration dialog (These
are global)
VAR_INPUT

END_VAR;

//  Output variables that can be configured via the configuration dialog
(These are global)
VAR_OUTPUT

END_VAR;

//  These are the global variables of the program
VAR
   devInfo  : btDeviceInfo;
   tags     : ARRAY [1..TAG_COUNT] OF iTag_data;
```

```
   tagLock  : MUTEX;
END_VAR;


//
// Helper function that returns a string version of the error codes
//
FUNCTION getError : STRING;
VAR_INPUT
   v : INT;
END_VAR;
   IF v > _BT_OK THEN
      getError := "";
   ELSE
      CASE v OF
          1: getError := "OK";
          0: getError := "Not supported";
         -1: getError := "Not open";
         -2: getError := "Alreay open";
         -3: getError := "Not present";
         -4: getError := "Not found";
         -5: getError := "Adapter not found";
         -6: getError := "Invalid adapter";
         -7: getError := "No more resources";
         -8: getError := "Invalid argument";
         -9: getError := "No data available";
        -10: getError := "Already paired";
        -11: getError := "Pair";
        -12: getError := "Busy";
        -13: getError := "Wrong State";
        -14: getError := "Connection failed";
        -15: getError := "Operation not available";
        -16: getError := "Op not permitted";
        -17: getError := "Timeout";
        -18: getError := "Auth";
        -19: getError := "Not connected";
        -99: getError := "Generic error";
      ELSE
           getError := strFormat(format:="Unknown error: \1", v1:=v);
      END_CASE;
   END_IF;
END_FUNCTION;



//
// This function prints information about a device and if it is a tag,
// it adds it to the "tags" array
//
FUNCTION DeviceFound;
VAR_INPUT
   dev   : STRING;
END_VAR;
VAR
   i     : INT;
   j     : INT;
   found : BOOL := FALSE;
END_VAR;

   devInfo(dev := dev);
   IF devInfo.status <> _BT_OK THEN
         DebugFmt(message:="devInfo() failed: \1 "+getError(v:=devInfo.status)
+", " + dev, v1:=devInfo.status);
   END_IF;
```

```
   IF devInfo.name = "" THEN
      RETURN;
   END_IF;
   DebugFmt(message:="  " + devInfo.name+" , Class: \4, type: \3, connected:
\1, tx power: \2",
      v2:=devInfo.tx_power, v4:=devInfo.clss, v3:=devInfo.addr_type,
v1:=INT(devInfo.connected));
   DebugFmt(message:="  status: \1, paired: \2, trusted: \3, RSSI: \4",
v1:=devInfo.status,
      v2:=INT(devInfo.paired), v3:=INT(devInfo.trusted), v4:=devInfo.RSSI);
   DebugFmt(message:="  Profiles: \1", v1:=devInfo.profiles);

   FOR i:= 1 TO 16 DO
      IF devInfo.uuid[i] <> "" THEN
         DebugFmt(message:="   [\1]: " + devInfo.uuid[i], v1:=i);
      END_IF;

      IF devInfo.UUID[i] = ITAG_SERVICE_UUID THEN
         DebugMsg(message:="   Found iTAG device");
         // Device is an iTag, add it to the "tags" array
         mxLock(mx:=tagLock);
         FOR j := 1 TO TAG_COUNT DO
            IF tags[j].status = ST_DISCON THEN
               tags[j].status := ST_FOUND;
               tags[j].address := dev;
               tags[j].alert_s := -1;
               tags[j].alert_c := -1;

               DebugFmt(message:="   Stored iTag in entry \1", v1:=j);
               EXIT;
            END_IF;
         END_FOR;
         mxUnlock(mx:=tagLock);

      END_IF;

   END_FOR;


END_FUNCTION;


//
// This function removes a tag from the "tags" array
//
FUNCTION DeviceLost;
VAR_INPUT
   dev   : STRING;
END_VAR;
VAR
   j     : INT;
END_VAR;

   DebugMsg(message:="Lost device "+dev);
   mxLock(mx:=tagLock);
   FOR j := 1 TO TAG_COUNT DO
      IF tags[j].status > ST_DISCON AND (tags[j].address = dev) THEN
         tags[j].status := ST_DISCON;
         DebugFmt(message:="Removed iTag from entry \1", v1:=j);
      END_IF;
   END_FOR;
   mxUnlock(mx:=tagLock);
```

```
END_FUNCTION;


//
// Thread for handling btWaitEvent.
//
THREAD_BLOCK thBtEvent
VAR
   rc       : INT;
   i        : INT;
   address  : STRING;
   ch       : SINT;
   port     : SINT;
   service  : INT;
   chara    : INT;
   size     : INT;
   level    : SINT := 0;
   data     : ARRAY [1..10] OF SINT;
END_VAR;
   WHILE TRUE DO
      rc := btWaitEvent(timeout:=10000, dev := address);
      IF rc <> _BT_ERR_TIMEOUT THEN
        DebugFmt(message:="event \1: "+address, v1:=rc);
        CASE rc OF
         _BT_EVENT_INCOMING:
            rc := btHandleSerialPortIncomingConnection(ch := ch, port :=
port);
            DebugFmt(message:="btHandleSerialPortIncomingConnection(\2) : \1,
\3", v1:=rc, v2:=ch, v3:=port);
         _BT_EVENT_DEV_FOUND:
            DebugFmt(message:="Found "+address);
            DeviceFound(dev := address);

         _BT_EVENT_DEV_LOST:
            DebugFmt(message:="Lost "+address);
            DeviceLost(dev := address);

         _BT_EVENT_PAIR_REQ:
            DebugFmt(message:="Requested confirm for "+address);
            // We do not want pairing in this example
            rc := btSendPairResponse(accept:=FALSE);
            DebugFmt(message:="btSendPairResponse: \1", v1:=rc);
         _BT_EVENT_NOTIFY:
            size := SIZEOF(data);
            rc := btleHandleNotification(service := service, char := chara,
size := size, data:=ADDR(data));
            DebugFmt(message:="btleHandleNotification: \1, service \2, char
\3, sz: \4", v1:=rc, v2:=service,
               v3:=chara, v4:=size);
            IF size > 0 THEN
               // Button has been pushed
               FOR i := 1 TO TAG_COUNT DO
                  // Trigger alert on all connected tags
                  IF tags[i].status = 2 THEN
                     // Start alerting
                     level := 2; // "High Alert"
                     rc := btleWriteVal(dev:=tags[i].address,
service:=tags[i].alert_s, char :=tags[i].alert_c,
                        data :=ADDR(level) , size := 1);
                     IF rc <> _BT_OK THEN
                        DebugFmt(message:="btleWriteVal: \1", v1:=rc);
                     END_IF;
                     Sleep (delay:=250);
                     // Stop alerting
```

```
                    level := 0;// "No Alert"
                    rc := btleWriteVal(dev:=tags[i].address,
service:=tags[i].alert_s, char :=tags[i].alert_c,
                       data :=ADDR(level) , size := 1);
                    DebugFmt(message:="btleWriteVal: \1", v1:=rc);
                    IF rc = _BT_ERR_NOT_CONNECTED THEN
                       // Device has disappeared, disconnect from it and
remove it.
                       // The _BT_EVENT_DEV_LOST event is triggered to update
the state
                       rc := btleDisconnect(dev := tags[i].address);
                       DebugFmt(message:="btleDisconnect: \1 ", v1:=rc);
                       rc := btDeviceRemove(dev := tags[i].address);
                       DebugFmt(message:="btDeviceRemove: \1 ", v1:=rc);
                    END_IF;
                    // Pause before triggering the next tag
                    Sleep(delay:=500);
                 END_IF;
              END_FOR;
           END_IF;
        END_CASE;
        rc := btEventDone();
        DebugFmt(message:="btEventDone: \1 "+getError(v:=rc), v1:=rc);
      END_IF;

   END_WHILE;
END_THREAD_BLOCK;
```

```
PROGRAM iTag;
// These are the local variables of the program block
VAR
   rc      : INT;
   thEvent : thBtEvent;
   i       : INT;
   service : INT;
   s       : INT;
   s_rc    : INT;
   sUUID   : STRING;
   primary : BOOL;
   flags   : DINT;
   ch      : INT;
   c       : INT;
   c_rc    : INT;
   cUUID   : STRING;

END_VAR;
// The next code will only be executed once after the program starts
tagLock := mxInit();

rc := btPower();
DebugFmt(message:="btPower: \1 "+getError(v:=rc)+"", v1:=rc);

thEvent();
rc := btDeviceRemove();
DebugFmt(message:="btDeviceRemove: \1 ", v1:=rc);


rc := btScanStart(transport := 2);
DebugFmt(message:="btScanStart: \1 "+getError(v:=rc), v1:=rc);
```

```
BEGIN
// Code from this point until END will be executed repeatedly

   DebugFmt(message:="Devices:");
   FOR i := 1 TO TAG_COUNT DO
      // Check tag status
      mxLock(mx:=tagLock);
      DebugFmt(message:="Dev \1: status \2", v1:=i, v2:=tags[i].status);
      IF tags[i].status = ST_FOUND THEN
         // Connect to the device to scan the services and characteristics
         rc := btleConnect (dev:=tags[i].address);
         DebugFmt(message:="btleConnect: \1", v1:=rc);
         IF rc = _BT_OK THEN
            // Scan services
            s := 1;
            REPEAT
              s_rc := btleServiceGet(dev:=tags[i].address, idx:=s,
service:=service, primary:=primary, UUID:=sUUID);
              IF s_rc = _BT_OK THEN
                DebugFmt(message:="  Service: \1, primary: \2, UUID: "+sUUID,
v1:=service, v2:=INT(primary));
                c := 1;
                REPEAT
                    // Scan characteristics
                    c_rc := btleCharGet(dev:=tags[i].address, idx:=c,
service:=service, char:=ch, UUID:=cUUID, flags := flags);
                    IF c_rc = _BT_OK THEN
                      DebugFmt(message:="    Service: \1, Char: \2, flags: \4,
UUID: "+cUUID, v1:=service, v2:=ch, v4:=flags);

                      // check for alert service UUID
                      IF sUUID = ALERT_SERVICE_UUID AND cUUID =
ALERT_LEVEL_UUID THEN
                         tags[i].alert_s := service;
                         tags[i].alert_c := ch;

                      END_IF;
                      // Check for Button service UUID
                      IF  sUUID = ITAG_SERVICE_UUID AND cUUID =
ITAG_NOTIFY_UUID THEN
                         // Register notifications on the button service
                         rc := btleNotifyStart(dev := tags[i].address, service
:= service, char := ch);
                         DebugFmt(message:="btleNotifyStart
"+tags[i].address+": \1", v1:=rc);
                      END_IF;
                    ELSE
                       // print return code on error
                       DebugFmt(message:="    btleCharGet "+tags[i].address+":
\1", v1:=c_rc);
                    END_IF;
                  c := c + 1;
                UNTIL c_rc <> _BT_OK END_REPEAT;
              ELSE
               // print return code on error
               DebugFmt(message:="  btleServiceGet "+tags[i].address+": \1",
v1:=s_rc);
              END_IF;
            s := s + 1;
            UNTIL s_rc <> _BT_OK END_REPEAT;
            // update status
            tags[i].status := ST_CON;
```

```
            END_IF;

        END_IF;
        mxUnlock(mx:=tagLock);

    END_FOR;
    DebugFmt(message:="");
    Sleep(delay:=5000);

END;

END_PROGRAM;
```

6.14 Examples - Datalogger (simple)

```
//------------------------------------------------------------------------------
-
// Datalogger testprogram
//------------------------------------------------------------------------------
-
INCLUDE rtcu.inc

VAR
   LogWriter : LogWrite; // FB used for writing log entries
   LogReader : LogRead;  // FB used for reading entry at current read position
END_VAR;

//------------------------------------------------------------------------------
// Mainprogram
//------------------------------------------------------------------------------
PROGRAM test;
   // Test if the Datalogger is initialized
   IF NOT LogIsInitialized(key:=4712) THEN
      // Initialize datalogger system, 2 values per record, 10 records in
total
      LogInitialize(key:=4712, numlogvalues:=2, numlogrecords:=10);
      DebugMsg(message:="Log initialized");
   ELSE
      DebugMsg(message:="Log was already initialized");
   END_IF;

   // Check how many records we have room for in the datalogger
   // Should be 10
   DebugFmt(Message:="(a) LogMaxNumOfRecords() = \4",
v4:=logMaxNumOfRecords());
   // Check how many values in each record
   // Should be 2
   DebugFmt(Message:="(b) LogValuesPerRecord() = \1",
v1:=logValuesPerRecord());

   // Make an entry in the datalogger
   LogWriter(tag:=1, value[1]:=10, value[2]:=20);
   // And another
   LogWriter(tag:=2, value[1]:=11, value[2]:=21);

   // Check how many records present in the datalogger
   // Should be 2
   DebugFmt(Message:="(c) LogNumOfRecords() = \4", v4:=logNumOfRecords());

   // Set read position to the oldest (first) record
   LogFirst();
   // Fetch data from read position
   LogReader(); // LogReader.tag is 1, value[1] is 10, value[2] is 20
   // Advance read position (forward in time)
   LogNext();
   LogReader(); // LogReader.tag is 2, value[1] is 11, value[2] is 21
   // LogReader.tag should be 2
   DebugFmt(Message:="(d) LogReader.tag = \1", v1:=LogReader.tag);

   // Advance read position (backward in time)
   // We are now back to the first record
   LogPrev();
   LogReader(); // LogReader.tag is 1, value[1] is 10, value[2] is 20
   // LogReader.tag should be 1
   DebugFmt(Message:="(e) LogReader.tag = \1", v1:=LogReader.tag);
```

```
   // Modify the current record
   LogReWriteTag(tag:=123); // This will change the tagvalue of the current
record to 123
   // Now read record
   LogReader(); // LogReader.tag is now 123, value[1] is 10, value[2] is 20
   // LogReader.tag should be 123
   DebugFmt(Message:="(f) LogReader.tag = \1",    v1:=LogReader.tag);
   DebugFmt(Message:="(f) LogReader.year = \1",   v1:=LogReader.year);
   DebugFmt(Message:="(f) LogReader.month = \1", v1:=LogReader.month);
   DebugFmt(Message:="(f) LogReader.day = \1",    v1:=LogReader.day);
   DebugFmt(Message:="(f) LogReader.hour = \1",   v1:=LogReader.hour);
   DebugFmt(Message:="(f) LogReader.minute = \1",v1:=LogReader.minute);
   DebugFmt(Message:="(f) LogReader.second = \1",v1:=LogReader.second);
BEGIN

END;

END_PROGRAM;
```

6.15 Examples - GPS Mobile (simple)

```
//-------------------------------------------------------------------------
-
// GPS Mobile.vpl, created 2002-05-30 09:30
//
// Small application that shows how to obtain GPS positions from the external
// GPS receiver on the RTCU product. The program will show in the debug window
// the current distance and bearing to Logic IO in
// Denmark. If an SMS text message is sent to the RTCU device, it will
respond
// back to the senders phone with the current distance/bearing to Logic IO
//
// Please note that the LED1/2 indicator will be:
//   green if connected to GSM net but NO valid GPS position
//   red if GPS position valid but NOT connected to GSM net
//   orange (green and red)  if connected to GSM net and valid GPS position
//-------------------------------------------------------------------------
-
INCLUDE rtcu.inc

VAR_OUTPUT
   gsmConnect : BOOL; | Indicates that the GSM is connected to a basestation
(Green)
   gpsValid   : BOOL; | Indicates that a position fix has been obtained (Red)
END_VAR;

PROGRAM GPS_Example;
VAR
   sms      : gsmIncomingSMS;
   gps      : gpsFix;
   gc       : gpsDistanceX;
   str      : STRING;
   awaitFix : BOOL;
END_VAR;
   // Turn on power to the GPS receiver
   gpsPower(power:=TRUE);
   // Turn on power to the GSM module
   gsmPower(power:=TRUE);
BEGIN
   // Update function block
   sms();
   // Update GPS data
   gps();
   // If we got a valid fix from the GPS
   IF gps.mode > 1 THEN
      // Calculate distance and bearing to Logic IO in Denmark
      gc(latitude1:=55513078, longitude1:=9510530, latitude2:=gps.latitude,
longitude2:=gps.longitude);
      // Build string with information
      str:=strFormat(format:="Distance to Logic IO=\4.\3 KM, bearing=\2 deg",
v4:=gc.distance/1000, v3:=INT(gc.distance MOD 1000), v2:=gc.bearing);
      DebugMsg(message:=str);
   END_IF;

   // If we receive a SMS message, we want the next GPS position that is valid
   IF sms.status>0 THEN
      awaitFix:=TRUE;
   END_IF;

   // If we are waiting for next valid GPS position, and GPS position is
valid,
```

```
    IF awaitFix AND gps.mode > 1 THEN
       awaitFix:=false;
       // Send an SMS with the position
       gsmSendSMS(phonenumber:=sms.phonenumber, message:=str);
    END_IF;

    // Indicate on LED (green part) if we are connected to a GSM basestation
    gsmConnect:=gsmConnected();
    // Indicate on LED (red part) if valid GPS position
    gpsValid := gps.mode > 1;
END;

END_PROGRAM;
```

6.16 Examples - Remote IO

```
//-------------------------------------------------------------------------
-
// Remote IO.vpl, created 2003-01-04 14:35
//
// This application will monitor upto 16 digital inputs. If any of these
changes
// state, the application will send an SMS message to a predefined number
containing
// information about which inputs has changed state. When the application
receives
// such an SMS message, it will update its own outputs accordingly. The number
// which will receive the status changes, can be set with the "phone=xxx" SMS
// command (see below). All SMS messages can be sent in lower and/or uppercase
//
// Format of SMS messages:
//
// Snn=x;Smm=y;Skk=z
// Where nn,mm and kk is the input number (1..16) and x,y and z is the NEW
state
// for the input (0/1).
//
// phone=xxxxxxxx
// Where xxxxxxxx is the number all status changes should be sent to, if blank
// no change of states are reported. The phonenumber will be saved at
// persistent memory location 1
//-------------------------------------------------------------------------
-
INCLUDE rtcu.inc

// Input variables that can be configured via the configuration dialog (These
are global)
VAR_INPUT
   DIn : ARRAY[1..16] OF BOOL; | Digital inputs
END_VAR;

// Output variables that can be configured via the configuration dialog (These
are global)
VAR_OUTPUT
   DOut : ARRAY[1..16] OF BOOL; | Digital outputs
   gsmOK: BOOL; | TRUE if connected to GSM basestation (Blinks during startup)
END_VAR;


PROGRAM Remote_IO;

VAR
   command    : STRING;
   position   : INT;
   extract    : strGetValues;
   str        : STRING;       // Temporary string
   statestr   : STRING;       // Temporary string
   index      : SINT;         // Temporary SINT
   DIn_old    : ARRAY[1..16] OF BOOL; // Keeps a copy of the old state of the
inputs
   sms        : gsmIncomingSMS;       // Receives incoming SMS messages
   pdu        : gsmIncomingPDU;
   pdubuf     : ARRAY[1..140] OF SINT;
   gsm_linsec : DINT;
END_VAR;
```

```
// Switch power on to GSM module
gsmPower(power:=TRUE);

// Wait for successful connect to the GSM network
WHILE NOT gsmConnected() DO
   Sleep(delay:=500);
   DebugMsg(message:="Waiting for connect to GSM network...");
   gsmOK:=NOT gsmOK; // The LED will blink during this
   UPDATEOUT;
END_WHILE;

gsm_linsec:=clockNow() + (24*60*60);

pdu(message:=ADDR(PDUbuf));

BEGIN
   // Update function blocks
   sms();
   pdu();

   IF pdu.status>0 THEN
      DebugFmt(message:="PDU received. \1",v1:=pdu.length);
      DebugFmt(message:=pdu.phonenumber);
   END_IF;

//-----------------------------------------------------------------------------
// Check all inputs for state change
//-----------------------------------------------------------------------------
   str:="";
   FOR index:=1 TO 16 DO
      // If the input has changed state since last time
      IF Din[index] <> Din_old[index] THEN
         // Build the "Snn=x;" string
         statestr:=strFormat(format:="S\1=\2;", v1:=index,
v2:=SINT(Din[index]));
         // Add it to the command string we will send
         str:=strConcat(str1:=str, str2:=statestr);
         // Remember the new state
         Din_Old[index]:=Din[index];
      END_IF;
   END_FOR;

   // If any state changes, go send them
   IF strLen(str:=str)>0 THEN
      statestr:=LoadString(index:=1);
      // ...but only if phonenumber is defined
      IF strLen(str:=statestr)=0 THEN
         DebugMsg(message:="Phonenumber is empty, sending NO state change");
      ELSE
         DebugMsg(message:="Sending new state change:");
         DebugMsg(message:=str);
         gsmSendSMS(phonenumber:=statestr, message:=str);
      END_IF;
   END_IF;


   //-----------------------------------------------------------------------------
--
   // See if any incoming SMS messages
   //-----------------------------------------------------------------------------
--
   IF sms.status > 0 THEN
      position:=1;
      // Loop through the received message, and decode it
```

```
      WHILE TRUE DO
         // Find next ";" delimited command
         command:=strToken(str:=sms.message, delimiter:=";",index:=position);
         position:=position+1;
         DebugMsg(message:=command);
         // If empty, no more commands...
         IF strLen(str:=command)=0 THEN
            DebugMsg(message:="No more commands...");
            EXIT;
         END_IF;

         //-----------------------------------------------------------------
--------
         // See if it is a "Snn=x" command...
         // if we find a "S" at first position, and a "=" at position 3 or 4
         //-----------------------------------------------------------------
--------
         IF strFind(str1:=command, str2:="S")=1 AND
            (strFind(str1:=command, str2:="=")=3 OR strFind(str1:=command,
str2:="=")=4) THEN
            extract(str:=command, format:="S\1=\2");
            IF extract.match THEN
               // extract.v1 contains input number, extract.v2 contains the
new state
               IF extract.v1 >= 1 AND extract.v1 <= 16 THEN
                  // Reflect the new state on the outputs
                  DOut[extract.v1]:=BOOL(extract.v2);
               END_IF;
            END_IF;

         //-----------------------------------------------------------------
--------
         // See if it's a "phone=xxxxx" command
         // If so, we save the new number, and sends back a confirmation to
the sender
         //-----------------------------------------------------------------
--------
         ELSIF strFind(str1:=command, str2:="phone=")>0 THEN
            // Save the parameters
            SaveString(index:=1, str:=strMid(str:=command, start:=7));
            str:=strConcat(str1:="New receiver set, number=",
str2:=LoadString(index:=1));
            gsmSendSMS(phonenumber:=sms.phonenumber, message:=str);
         END_IF;
      END_WHILE;
   END_IF;

   IF clockNow() > gsm_linsec THEN
      DebugMsg(message:="GSM Module turned OFF");
      gsmPower(power:=OFF);
      DebugMsg(message:="GSM Module turned ON");
      gsmPower(power:=ON);
      gsm_linsec:=clockNow() + (24*60*60);

      // Wait for successful connect to the GSM network
      WHILE NOT gsmConnected() DO
         Sleep(delay:=500);
         DebugMsg(message:="Waiting for connect to GSM network...");
      END_WHILE;
      DebugMsg(message:="Now connected to GSM network");
   END_IF;

   gsmOK:=gsmConnected();
END;
```

```
END_PROGRAM;
```

6.17 Examples - REST Example

```
//-------------------------------------------------------------------------
-
// REST example
//
// Starts a REST server on port 80 or 443 and then performs a request on it
every 30s.
//
// When using HTTPS, at least three certificates are needed:
// The client will use the certificate+key "client".
// The "client" certificate must be signed by the root certificate "ca".
// The server will use the certificate+key "server", which must be signed
// by a trusted root certificate.
//
//
//-------------------------------------------------------------------------
-
INCLUDE rtcu.inc

// Set to 1 to use HTTPS, set to 0 to use HTTP.
#DEFINE USE_TLS   1
// Set to 1 to use the string template functions to create JSON,
// set to 0 to use the JSON API.
#DEFINE USE_ST    1

// Functions that parse JSON structures and dumps them to the device output.
FUNCTION INTERFACE dumpValue;
VAR_INPUT
   json   : SYSHANDLE;
   prefix : STRING;
   key    : STRING;
   idx    : INT := -1;
   type   : INT;
   txt    : STRING;
END_VAR;
END_FUNCTION;

FUNCTION dumpStructure;
VAR_INPUT
   json   : SYSHANDLE;
   prefix : STRING;
END_VAR;
VAR
   type, i, rc : INT;
   key         : STRING;
END_VAR;
   type := jsonGetType(o:=json, idx:=-1);
   IF type = 1 THEN
      // object
      DebugMsg(message:=prefix+"{");
      i := 0;
      REPEAT
         rc := jsonGetKey(o:=json, idx:= i, key:=key, type:=type);
         IF rc = 1 THEN
            dumpValue(json:=json, prefix:=prefix+" ", txt:=key+" ", key:=key,
idx:=-1, type:=type);
         END_IF;
         i := i+1;
      UNTIL  rc < 0
      END_REPEAT;
      DebugMsg(message:=prefix+"}");
```

```
   ELSIF type = 2 THEN
      // array
      DebugMsg(message:=prefix+"[");
      FOR i := 0 TO jsonArraySize(o:=json)-1 DO
            dumpValue(json:=json, prefix:=prefix+" ", txt:="["+intToStr(v:=i)
+"] ", key:="", idx:=i, type:=jsonGetType(o:=json, idx:= i));
      END_FOR;
      DebugMsg(message:=prefix+"]");
   END_IF;
END_FUNCTION;

FUNCTION IMPLEMENTATION dumpValue;
VAR
   rc  : INT;
   tmp : SYSHANDLE;
   str : STRING;
   d   : DINT;
   b   : BOOL;
   f   : DOUBLE;
END_VAR;
   CASE type OF
   1,2:// Object or array
      DebugFmt(message:=prefix+txt+":");
      rc := jsonGetValue(o:=json, key:=key, idx:=idx, value := tmp);
      dumpStructure(json:=tmp, prefix := prefix);
      rc := jsonFree(o:=tmp);
   3: // string
      jsonGetString(o:=json, key:=key, idx:=idx, value :=str);
      DebugFmt(message:=prefix+txt+": "+str);
   4: // int
      jsonGetInt(o:=json, key:=key, idx:=idx, value :=d);
      DebugFmt(message:=prefix+txt+": \4", v4:=d);
   5: // Float
      jsonGetFloat(o:=json, key:=key, idx:=idx, value :=f);
      DebugFmt(message:=prefix+txt+": "+doubleToStr(v:=f));
   6: // Bool
      jsonGetBool(o:=json, key:=key, idx:=idx, value :=b);
      IF b THEN
         DebugFmt(message:=prefix+txt+": TRUE");
      ELSE
         DebugFmt(message:=prefix+txt+": FALSE");
      END_IF;
   7: // NULL
      DebugFmt(message:=prefix+txt+": NULL");
   END_CASE;
END_FUNCTION;


VAR
   clock     : clockLinsecToTime;
   jsonStats : jsonHandleStats;
END_VAR;

FUNCTION linsecToStr : STRING;
VAR_INPUT
   linsec : DINT;
END_VAR;
VAR
   str : STRING;
END_VAR;
   clock(linsec:=linsec);
   str:=strFormat(format:="\1-\2-\3 ", v1:=clock.year, v2:=clock.month,
v3:=clock.day);
   str:=str+strFormat(format:="\1:\2:\3", v1:=clock.hour, v2:=clock.minute,
```

```
v3:=clock.second);

   linsecToStr:=str;
END_FUNCTION;

// Debug function that prints all the information about a certificate.
FUNCTION dumpCert;
VAR_INPUT
   req : SYSHANDLE;
   rip : DINT;
END_VAR;
VAR
   rc  : INT;
   str : STRING;
   d   : DINT;
   i   : INT;
   ip  : DINT;
END_VAR;
   rc := restReqClientCertPresent(req:=req);
   IF rc = 1 THEN
      DebugFmt(message:="Client cert present");
      rc := restReqClientCertSubjectGet(req:=req, str:=str);
      DebugFmt(message:=" Subject: "+str+": \1", v1:=rc);
      rc := restReqClientCertSubjectCNGet(req:=req, str:=str);
      DebugFmt(message:=" CN: "+str+": \1", v1:=rc);
      rc := restReqClientCertIssuerGet(req:=req, str:=str);
      DebugFmt(message:=" Issuer: "+str+": \1", v1:=rc);
      rc := restReqClientCertVersionGet(req:=req);
      DebugFmt(message:=" Version: \1", v1:=rc);
      d := restReqClientCertValidFrom(req:=req);
      DebugFmt(message:=" Valid from: \4, "+linsecToStr(linsec:=d), v4:=d);
      d := restReqClientCertValidTo(req:=req);
      DebugFmt(message:=" Valid to: \4, "+linsecToStr(linsec:=d), v4:=d);
      rc := restReqClientCertSerialGet(req:=req, str:=str);
      DebugFmt(message:=" Serial: "+str+": \1", v1:=rc);
      rc := restReqClientCertFingerprintGet(req:=req, type:=0, str:=str);
      DebugFmt(message:=" SHA1  : "+str+": \1", v1:=rc);
      rc := restReqClientCertFingerprintGet(req:=req, type:=1, str:=str);
      DebugFmt(message:=" MD51  : "+str+": \1", v1:=rc);

      rc := restReqClientCertCheckHostname(req:=req, hostname:="localhost");
      DebugFmt(message:=" Match localhost: \1", v1:=rc, v2:=i);

      rc := restReqClientCertCheckEmail(req := req, email :=
"test@example.com");
      DebugFmt(message:=" Match test@example.com: \1", v1 := rc);


      i := 0;
      REPEAT
         rc := restReqClientCertSANGet(req:=req, idx:=i, san:=str);
         DebugFmt(message:= "  SAN[\1]: \2: "+str, v1:=i, v2:=rc);
         IF rc = 4 THEN
            ip := soAddrToIP(address :=str);
            IF ip = rip THEN
               DebugMsg(message:="Matching IP found: "+str);
            END_IF;
         END_IF;
         i := i + 1;
      UNTIL rc = -21
      END_REPEAT;

   ELSE
      DebugFmt(message:="No client cert present");
```

```
   END_IF;
END_FUNCTION;



// Callback triggered when a POST request is sent to "/devices".
FUNCTION CALLBACK devicesPostCallback: INT;
VAR_INPUT
   req           : SYSHANDLE; | Request
   resp          : SYSHANDLE; | Response
   arg           : DINT;      | User argument
END_VAR;
VAR
   talq_version  : STRING;
   client_address: STRING;
   id            : STRING;
   json          : STRING;
   str           : STRING;
END_VAR;
   restReqHeaderGet(req:=req, name:="talq-version", value:=talq_version);
   restReqQueryGet(req:=req, name:="clientAddress", value:=client_address);
   restReqQueryGet(req:=req, name:="id", value:=id);
   DebugFmt(message:="Request: id: "+id+",  client address: "+client_address);
   restReqClientAddressGet(req:=req, address:=str);
   DebugFmt(message:="Address: "+str);
   // Dump all certificate info
   dumpCert(req:=req, rip := soAddrToIP(address :=str));

   // Get content of body as string
   restReqBodyGetString(req:=req, str := json);

   IF json <> "" THEN
      // Build response - just return the same JSON as received for now
      restRespBodySetString(resp:=resp, str := json);
      restRespHeaderSet(resp:=resp, name:="Content-Type",
value:="application/json");

      devicesPostCallback := 201;
   ELSE
      // Error response
      restRespBodySetString(resp:=resp, str := "{$"key$":$"payloadError$" }");
      restRespHeaderSet(resp:=resp, name:="Content-Type",
value:="application/json");

      devicesPostCallback := 400;
   END_IF;
END_FUNCTION;

FUNCTION GenerateJSONFromTemplate:INT;
VAR_INPUT
   json            : ACCESS SYSHANDLE;
   deviceName      : STRING;
   deviceClassName : STRING;
   displayName     : STRING;
   assetId         : STRING;
   lampTypeId      : STRING;
END_VAR;
VAR
   st    : SYSHANDLE;
   rc    : INT;
   str   : STRING;
   jStr  : STRING;
END_VAR;
   // Template string
```

```
   str := STRING_BEGIN
   [
    {
       "address": "00000000-0000-0000-0000-000000000000",
       "name": "<deviceName>",
       "class": "<deviceClassName>",
       "functions": [
         {
            "id": "basic001",
            "type": "BasicFunction",
            "displayName": {
              "value": "<displayName>"
            },
            "assetId": {
              "value": "<assetId>"
            }

         },
         {
            "id": "lampActuator001",
            "type": "LampActuatorFunction",
            "lampTypeId": {
              "value": "<lampTypeId>"
            },
            "maintenanceFactor": {
              "value": 50
            },
            "maintenancePeriod": {
              "value": 12
            }
         }
       ]
     }
   ]
STRING_END;

   // Create template
   rc := strTemplateCreate(st:=st, str:=str, sov := "<", eov := ">");
   DebugFmt(message:="strTemplateCreate: \1", v1:=rc);

   // Set template values
   rc := strTemplateSetVar(st:=st, name:="deviceName", value:=DeviceName);
   DebugFmt(message:="strTemplateSetVar: \1", v1:=rc);
   rc := strTemplateSetVar(st:=st, name:="deviceClassName",
value:=deviceClassName);
   DebugFmt(message:="strTemplateSetVar: \1", v1:=rc);
   rc := strTemplateSetVar(st:=st, name:="assetId", value:=assetId);
   DebugFmt(message:="strTemplateSetVar: \1", v1:=rc);
   rc := strTemplateSetVar(st:=st, name:="lampTypeId", value:=lampTypeId);
   DebugFmt(message:="strTemplateSetVar: \1", v1:=rc);

   // Generate string
   rc := strTemplateGenerateString(st:=st, str:=jStr);

   // Clean up
   rc := strTemplateFree(st:=st);
   DebugFmt(message:="strTemplateFree: \1", v1:=rc);

   rc := jsonFromString(o:=json, str:=jStr);
   DebugFmt(message:="jsonFromString: \1", v1:=rc);

   GenerateJSONFromTemplate := rc;
END_FUNCTION;
```

```
FUNCTION GenerateJSON:INT;
VAR_INPUT
   json : ACCESS SYSHANDLE;
END_VAR;
VAR
   rc : INT;
   devices, device, functions, func, tmp : SYSHANDLE;
   handles, data:DINT;
END_VAR;
   // devices array
   rc := jsonCreateArray(o:=devices);
   // First device
   rc := jsonCreateObject(o:=device);

   rc := jsonSetValueString(o:=device, key := "address",
         value := "00000000-0000-0000-0000-000000000000");
   rc := jsonSetValueString(o:=device, key := "name", value :=
"<deviceName>");
   rc := jsonSetValueString(o:=device, key := "class", value :=
"<deviceClassName>");

   // functions array
   rc := jsonCreateArray(o:=functions);

   // first function
   rc := jsonCreateObject(o:=func);
   rc := jsonSetValueString(o:=func, key := "id", value := "basic001");
   rc := jsonSetValueString(o:=func, key := "type", value := "basicFunction");
   // displayName
   rc := jsonCreateObject(o:=tmp);
   rc := jsonSetValueString(o:=tmp, key := "value", value := "<displayName>");
   rc := jsonSetValue(o:=func, key := "displayName", value := tmp);
   rc := jsonFree(o:=tmp);
   // assetId
   rc := jsonCreateObject(o:=tmp);
   rc := jsonSetValueString(o:=tmp, key := "value", value := "<assetId>");
   rc := jsonSetValue(o:=func, key := "assetId", value := tmp);


   rc := jsonFree(o:=tmp);
   // Add func to functions
   rc := jsonAddValue(o:=functions, value:=func);
   rc := jsonFree(o:=func);
   // second function
   rc := jsonCreateObject(o:=func);

   rc := jsonSetValueString(o:=func, key := "id", value := "lampActuator001");
   rc := jsonSetValueString(o:=func, key := "type", value :=
"LampActuatorFunction");
   // lampTypeId
   rc := jsonCreateObject(o:=tmp);
   rc := jsonSetValueString(o:=tmp, key := "value", value := "<lampTypeId>");
   rc := jsonSetValue(o:=func, key := "lampTypeId", value := tmp);
   rc := jsonFree(o:=tmp);
   // maintenanceFactor
   rc := jsonCreateObject(o:=tmp);
   rc := jsonSetValueInt(o:=tmp, key := "value", value := 50);
   rc := jsonSetValue(o:=func, key := "maintenanceFactor", value := tmp);
   rc := jsonFree(o:=tmp);
   // maintenancePeriod
   rc := jsonCreateObject(o:=tmp);
   rc := jsonSetValueInt(o:=tmp, key := "value", value := 12);
   rc := jsonSetValue(o:=func, key := "maintenancePeriod", value := tmp);
```

```
   rc := jsonFree(o:=tmp);
   // Add func to functions
   rc := jsonAddValue(o:=functions, value:=func);
   rc := jsonFree(o:=func);

   // Add functions to device
   rc := jsonSetValue(o:=device, key := "functions", value := functions);
   rc := jsonFree(o:=functions);
   // Add device to devices
   rc := jsonAddValue(o:=devices, value:=device);
   rc := jsonFree(o:=device);

   json := devices;
END_FUNCTION;



FUNCTION SendRequest;
VAR
   str  : STRING;
   req, resp : SYSHANDLE;
   json : SYSHANDLE;
   rc   : INT;
END_VAR;
   DebugMsg(message:="--------");

   // Create JSON:
   IF USE_ST = 1 THEN
      rc := GenerateJSONFromTemplate(json:=json, deviceName:="dev1",
assetId:="asset 1", lampTypeId:="lamp 1");
   ELSE
      rc := GenerateJson(json:=json);
   END_IF;


   // Perform request:
   IF USE_TLS = 1 THEN
      rc := restReqCreate(req:=req, method := "POST",
       url := "https://127.0.0.1/devices/1234",
       check_hostname:=OFF);
   ELSE
      rc := restReqCreate(req:=req, method := "POST",
       url := "http://127.0.0.1/devices/1234");
   END_IF;

   // Show usage of JSON handles, expected 1 handle
   jsonStats();
   IF jsonStats.status >0 THEN
      DebugFmt(message:="JSON status: \1 handle(s)",
v1:=jsonStats.no_handles);
   END_IF;

   rc := restReqQuerySet(req:=req, name := "clientAddress",
         value:="110a8321-e34c-112p5-b567-566655648453");
   rc := restReqHeaderSet(req:=req, name := "talq-version", value:="2.1.0");

   rc := restReqBodySetJSON(req:=req, json:=json);

   rc := jsonFree(o:=json);

   rc := restReqHeaderSet(req:=req, name:="Accept",
value:="application/json");
```

```
   IF USE_TLS = 1 THEN
      rc := restReqClientCertSet(req:=req, cert:="client");
      DebugFmt(message:="restReqClientCertSet: \1", v1:=rc);
   END_IF;

   // Send request
   rc := restClientRequest(req:=req, resp := resp);
   DebugFmt(message:="Send request: \1", v1:=rc);
   // Handle response
   IF rc = 201 THEN
      // success
      rc := restRespBodyGetString(resp:=resp, str := str);
      DebugFmt(message:="data: "+str);

      rc := restRespBodyGetJSON(resp:=resp, json := json);

      rc := jsonToString(o:=json, str:=str, indent:=0);

      // Parse JSON and dump it to the device output
      dumpStructure(json:=json);

      rc := jsonFree(o:=json);

      DebugFmt(message:="JSON: "+str);

      rc := restRespFree(resp:=resp);
   END_IF;

   rc := restReqFree(req:=req);

END_FUNCTION;



PROGRAM rest_1;
// These are the local variables of the program block
VAR
   serv : SYSHANDLE;
   rc   : INT;
END_VAR;
   // The next code will only be executed once after the program starts

   // Open ethernet to allow access to the server from the outside
   netOpen(iface:=2);

   // Create server
   IF USE_TLS = 1 THEN
      rc := restServerCreate(handle:=serv, port:=443, cert:="server",
accept_ca:="ca");
   ELSE
      rc := restServerCreate(handle:=serv, port:=80);
   END_IF;
   DebugFmt(message:="restServerCreate: \1", v1:=rc);
   // Register callback for /devices/<id>
   rc := restServerEndpointAdd(handle:=serv, method:="*",
url_prefix:="/devices",
      url_format:="/:id", cb_func:=@devicesPostCallback);
   // Start server
   rc := restServerStart(handle:=serv);
   DebugFmt(message:="restServerStart: \1", v1:=rc);
BEGIN
   // Code from this point until END will be executed repeatedly
```

```
    SendRequest();

    Sleep(delay:=30000);
END;
END_PROGRAM;
```

6.18 Examples - REST Basic Authentication Example

```
//---------------------------------------------------------------------------
-
// REST example using basic authentication
// Sending a request to /test will return a response containing information
// about the request as JSON.
// The client must provide the username "user" and the password "pass" to
//
//---------------------------------------------------------------------------
-
INCLUDE rtcu.inc

#DEFINE EP_AUTH      1
#DEFINE EP_CONTENT   2
#DEFINE EP_LOGOUT    3

// Authentication callback
// Validates the username and password.
// If called from /logout, it will just return Unauthorized
// to make the client forget its current authentication parameters.
FUNCTION CALLBACK devicesAuthCallback : INT;
VAR_INPUT
   req         : SYSHANDLE;
   resp        : SYSHANDLE;
   arg         : DINT; // User argument indicating endpoint.
END_VAR;
VAR
   str, user, pass : STRING;
   i  : INT;
   rc : INT;
END_VAR;
   restReqClientAddressGet (req:=req, address:=str);
   DebugFmt(message:="Auth Request from "+str);

   // Enable basic authentication in the response
   rc := restRespBasicAuthSet(resp:=resp);
   DebugFmt(message:="restRespBasicAuthSet: \1", v1:=rc);

   // Read current authentication
   rc := restReqBasicAuthGet(req:=req, user:=user, pass:=pass);
   DebugFmt(message:="restReqBasicAuthGet: \1, "+user+":"+pass, v1:=rc);

   IF arg = EP_LOGOUT THEN
      restRespBodySetString(resp:=resp, str := "Logged out");
      // unauthorized, request basic authentication
      devicesAuthCallback := -1;
   ELSE
      IF user = "user" AND pass = "pass" THEN
         // Valid user, continue with content callback
         devicesAuthCallback:=0;
      ELSE
         restRespBodySetString(resp:=resp, str := "Invalid user");
         // unauthorized, request basic authentication
         devicesAuthCallback := -1;
      END_IF;
   END_IF;
END_FUNCTION;


// Content callback
// Creates a response containing the request parameters converted into JSON
```

```
FUNCTION CALLBACK devicesGetCallback: INT;
VAR_INPUT
   req          : SYSHANDLE;
   resp         : SYSHANDLE;
   arg          : DINT;
END_VAR;
VAR
   str, name    : STRING;
   rc, i        : INT;
   json, o, a   : SYSHANDLE;
   pos, sz, len : DINT;
   buf          : ARRAY [1..32] OF SINT;
END_VAR;
   // Create main JSON object
   jsonCreateObject(o:=json);

   // Headers
   jsonCreateObject(o:=o);
   i := 0;
   rc := restReqHeaderKey(req:=req, idx:=i, name:=name);
   WHILE rc > 0 DO
      str := "";
      restReqHeaderGet(req:=req, name:=name, value := str);
      jsonSetValueString(o:=o, key:=name, value:=str);
      i := i + 1;
      rc := restReqHeaderKey(req:=req, idx:=i, name:=name);
   END_WHILE;
   jsonSetValue(o:=json, key:="headers", value:=o);
   jsonFree(o:=o);

   // Queries
   jsonCreateObject(o:=o);
   i := 0;
   rc := restReqQueryKey(req:=req, idx:=i, name:=name);
   WHILE rc > 0 DO
      restReqQueryGet(req:=req, name:=name, value := str);
      jsonSetValueString(o:=o, key:=name, value:=str);
      i := i + 1;
      rc := restReqQueryKey(req:=req, idx:=i, name:=name);
   END_WHILE;
   jsonSetValue(o:=json, key:="queries", value:=o);
   jsonFree(o:=o);

   // Post parameters
   jsonCreateObject(o:=o);
   i := 0;
   rc := restReqPostKey(req:=req, idx:=i, name:=name);
   WHILE rc > 0 DO
      restReqPostGet(req:=req, name:=name, value := str);
      jsonSetValueString(o:=o, key:=name, value:=str);
      i := i + 1;
      rc := restReqPostKey(req:=req, idx:=i, name:=name);
   END_WHILE;
   jsonSetValue(o:=json, key:="posts", value:=o);
   jsonFree(o:=o);

   // Body size
   sz := restReqBodySize(req:=req);
   jsonSetValueInt(o:=json, key:="body-size", value:=sz);

   // Body as text
   str := "";
   restReqBodyGetString(req:=req, str:=str);
   jsonSetValueString(o:=json, key:="body-text", value:=str);
```

```
   // Raw body, split into smaller hex-strings.
   rc := jsonCreateArray(o:=a);
   pos := 0;
   WHILE pos < sz DO
      len := restReqBodyGetRaw(req:=req, dst:=ADDR(buf), size := SIZEOF(buf),
start := pos);
      IF len <= 0 THEN
         DebugFmt(message:="failed to get body: \4", v4:=len);
         EXIT;
      END_IF;
      str := "";
      FOR i := 1 TO len DO
         str := str+sintToHex(v:=buf[i]);
      END_FOR;
      jsonAddValueString(o:=a, value:=str);

      pos := pos + len;
   END_WHILE;
   jsonSetValue(o:=json, key:="body-raw", value:=a);
   rc := jsonFree(o:=a);

   // Set response
   rc := restRespBodySetJSON(resp:=resp, json:=json);
   DebugFmt(message:="restRespBodySetJSON: \1, ", v1:=rc);

   rc := jsonFree(o:=json);
   DebugFmt(message:="jsonFree: \1, ", v1:=rc);

   devicesGetCallback := 200;
END_FUNCTION;



// Send a request to the local server and show the response.
FUNCTION SendRequest;
VAR
   str, name    : STRING;
   req, resp    : SYSHANDLE;
   rc, i        : INT;
   pos, sz, len : DINT;
   buf          : ARRAY[1..16] OF SINT;
END_VAR;
   DebugMsg(message:="--------");

   rc := restReqCreate(req:=req, method := "POST",
      url := "http://127.0.0.1/test");

   rc := restReqQuerySet(req:=req, name := "query", value:="value 1");

   rc := restReqBodySetString(req:=req, str:="{$"test$":42}");
   rc := restReqHeaderSet(req:=req, name:="Accept",
value:="application/json");

   // Set authentication
   rc := restReqBasicAuthSet(req:=req, user:="user", pass:="pass");

   // Send request
   rc := restClientRequest(req:=req, resp := resp);
   DebugFmt(message:="Send request: \1", v1:=rc);
   // Handle response
   IF rc = 200 THEN
      // success
```

```
      // Print headers
      i := 0;
      rc := restRespHeaderKey(resp:=resp, idx:=i, name:=name);
      WHILE rc > 0 DO
         restRespHeaderGet(resp:=resp, name:=name, value := str);
         DebugMsg(message:=" "+name+": "+str);
         i := i + 1;
         rc := restRespHeaderKey(resp:=resp, idx:=i, name:=name);
      END_WHILE;

      // Print body information
      sz := restRespBodySize(resp:=resp);
      DebugFmt(message:="body size: \4", v4:=sz);

      rc := restRespBodyGetString(resp:=resp, str := str);
      DebugFmt(message:="body: "+str);

      // Print raw body
      pos := 0;
      WHILE pos < sz DO
         len := restRespBodyGetRaw(resp:=resp, dst:=ADDR(buf), size :=
SIZEOF(buf), start := pos);
         IF len <= 0 THEN
            DebugFmt(message:="failed to get body: \4", v4:=len);
            EXIT;
         END_IF;
         str := dintToHex(v:=pos)+": ";
         FOR i := 1 TO len DO
            str := str+sintToHex(v:=buf[i])+" ";
         END_FOR;
         DebugMsg(message:=str);

         pos := pos + len;
      END_WHILE;

      rc := restRespFree(resp:=resp);
   ELSIF rc > 0 THEN
      rc := restRespBodyGetString(resp:=resp, str := str);
      DebugFmt(message:="Request failed: "+str);
   END_IF;

   rc := restReqFree(req:=req);

END_FUNCTION;



PROGRAM auth;
// These are the local variables of the program block
VAR
   rc   : INT;
   serv : SYSHANDLE;
   port : DINT := 80;

END_VAR;
   // Open LAN interface to provide access for external connections.
   netOpen(iface:=2);

   rc := restServerCreate(handle:=serv, port:=port);
   DebugFmt(message:="restServerCreate: \1", v1:=rc);


   // Add authentication endpoint with high priority
   rc := restServerEndpointAdd(handle:=serv, url_prefix:="/test",
```

```
      cb_func:=@devicesAuthCallback, cb_arg:=EP_AUTH);
   DebugFmt(message:="restServerEndpointAdd: \1", v1:=rc);

   // Add content endpoint with lower priority
   rc := restServerEndpointAdd(handle:=serv, url_prefix:="/test",
      cb_func:=@devicesGetCallback, cb_arg:=EP_CONTENT, prio:=1);
   DebugFmt(message:="restServerEndpointAdd: \1", v1:=rc);

   // Add logout endpoint, reusing authentication endpoint
   rc := restServerEndpointAdd(handle:=serv, url_prefix:="/logout",
      cb_func:=@devicesAuthCallback, cb_arg:=EP_LOGOUT);
   DebugFmt(message:="restServerEndpointAdd: \1", v1:=rc);

   // Start server
   rc := restServerStart(handle:=serv);
   DebugFmt(message:="restServerStart: \1", v1:=rc);


BEGIN

   // send local request
   SendRequest();

   Sleep(delay:=30000);

END;

END_PROGRAM;
```

6.19 Examples - Telnet server

```
//-------------------------------------------------------------------------
-
// Telnet.vpl, created 2003-03-04 20:05
//
// This is a VERY simple telnet server. It has only a few commands, but still
// shows how to establish a TCP/IP connection. The demo requires that the
// RTCU device has a GLOBAL accessible IP address on the internet ! When a
Telnet
// client connects to the RTCU, a welcome message is shown, together with a
// prompt, where the telnet client then can enter commands. The following
commnds
// are available: input, on n, off n, quit and help. Input will show the state
// of 4 digital inputs, on/off n will switch digital output n to either on or
off
// state, quit will close the connection, and help will show a list of
available
// commands.
//-------------------------------------------------------------------------
-
INCLUDE rtcu.inc

// Input variables that can be configured via the configuration dialog (These
are global)
VAR_INPUT
   Din : ARRAY[1..4] OF BOOL; | Digital inputs (can be read with "input"
command)
END_VAR;

// Output variables that can be configured via the configuration dialog (These
are global)
VAR_OUTPUT
   LED : BOOL; | LED that shows the program scan
   Dout : ARRAY[1..4] OF BOOL; | Digital outputs (can be set with the
"on"/"off" commands)
END_VAR;

// These are the global variables of the program
VAR
   sockrcv     : sockReceive;    // Receives data on a TCP/IP socket
   soccon      : sockConnection; // Manages a TCP/IP connection
   id          : SINT;           // TCP/IP connection id
   rxbuf       : ARRAY[0..512] OF SINT; // Receive buffer for TCP/IP data
   txbuf       : ARRAY[0..512] OF SINT; // Transmit buffer for TCP/IP data
   i           : INT;            // Temporary variable
   str         : STRING;         // Temporary variable
   extract     : strGetValues;   // Matching of commandstrings
   PartialCommand : STRING;      // Characters on TCP/IP socket gets assembled
in this
   Command     : STRING;         // Final command received (after CR)
   listenport : SINT:=23;        // Which port to listen on (port 23=standard
Telnet port)
END_VAR;


//-------------------------------------------------------------------------
// Send a STRING on a TCP/IP connection
//-------------------------------------------------------------------------
FUNCTION SendData;
VAR_INPUT
   id : SINT;
```

```
   str : STRING;
END_VAR
VAR
   txbuf : ARRAY[0..100] OF SINT;
END_VAR;
   strToMemory(dst:=ADDR(txbuf), str:=str, len:=strLen(str:=str));
   DebugFmt(message:="sockSend=\1", v1:=sockSend(id:=id, data:=ADDR(txbuf),
size:=strLen(str:=str)));
END_FUNCTION;


//-------------------------------------------------------------------------
// Start at TCP/IP socket listen on a specified port
// (and update the sockConnection() and sockReceive() functionblocks)
//-------------------------------------------------------------------------
FUNCTION StartListen;
   // Start listener on port
   id:=sockListen(port:=listenport);
   DebugFmt(message:="socListen()=\1",v1:=id);
   // Update id for sockConnection() and sockReceive()
   soccon(id:=id);
   sockrcv(id:=id,data:=ADDR(rxbuf),MaxSize:=SIZEOF(rxbuf));
END_FUNCTION;

//-------------------------------------------------------------------------
// Main program
//-------------------------------------------------------------------------
PROGRAM Telnet;
// Switch on power to the GSM module
gsmPower(power:=TRUE);
// Open the GPRS connection (connects the RTCU device to the Internet)
DebugFmt(message:="gprsOpen()=\1",v1:=gprsOpen());
// Wait for connection to the Internet
// (This is actually not needed, as the sockListen() can be called before
// the connection is established)
WHILE NOT gprsConnected() DO
   DebugMsg(message:="Waiting for cellular connection");
   Sleep(delay:=3000);
END_WHILE;
// Show our assigned IP address (this is the one the telnet client should
connect to)
DebugMsg(message:=strConcat(str1:="My IP
Address=",str2:=sockIPToName(ip:=sockGetLocalIP())));
// Start listening for TCP/IP connects on the Telnet port (port 23)
StartListen();
BEGIN
   soccon(); // Update status for sockConnection() functionblock
   sockrcv(); // Update status for sockReceive() functionblock
   // Connection status changed (someone connected or disconnected)
   IF soccon.changed THEN
      // Someone just connected to us...
      IF soccon.Connected THEN
         DebugMsg(message:=strConcat(str1:=sockIPToName(ip:=soccon.remoteip),
str2:=" has connected"));
         SendData(id:=id, str:="Welcome to the RTCU Telnet Server.$N$N");
         SendData(id:=id, str:="Available commands:$N");
         SendData(id:=id, str:="  <input> : Show status of digital inputs$N");
         SendData(id:=id, str:="  <on n> : Set digital output n to ON$N");
         SendData(id:=id, str:="  <off n> : Set digital output n to OFF$N");
         SendData(id:=id, str:="  <quit>  : Close connection$N");
         SendData(id:=id, str:="  <help>/<?> : Show available commands$N");
         SendData(id:=id, str:="$NEnter command> ");
      // Someone disconnected
      ELSE
```

```
         DebugMsg(message:=strConcat(str1:=sockIPToName(ip:=soccon.remoteip),
str2:=" is disconnecting !"));
         // Disconnect socket
         sockDisconnect(id:=id);
         // and start listening again
         StartListen();
      END_IF;
   END_IF;


   // We have received some data...
   IF sockrcv.ready THEN
      DebugFmt(message:="socket data received len=\1",v1:=sockrcv.size);
      // Echo the received data
      DebugFmt(message:="sockSend=\1", v1:=sockSend(id:=id, data:=ADDR(rxbuf),
size:=sockrcv.size));
      // Convert the received data to a string
      str:=strFromMemory(src:=ADDR(rxbuf), len:=sockrcv.size);
      DebugMsg(message:=str);
      // concatenate the received data to command string
      PartialCommand:=strConcat(str1:=PartialCommand, str2:=str);
      // If there is a Carriage return in the string...
      i:=strFind(str1:=PartialCommand, str2:="$R");
      IF i>0 THEN
         // We now have a complete command assembled in PartialCommand
         Command:=strLeft(str:=PartialCommand, length:=i-1);
         DebugMsg(message:=strConcat(str1:="Command=", str2:=Command));
         PartialCommand:="";
         // Check if it is a "QUIT" command (No prompt sent if it's quit)
         IF strCompare(str1:="QUIT", str2:=Command)=0 THEN
            SendData(id:=id, str:="$NGoodbye.$N");
            // Disconnect socket
            // (Note: this will NOT activate a soccon.changed, as it is
"ourself" who has disconnected !)
            sockDisconnect(id:=id);
            // and start listening again
            StartListen();
            // Check for other commands
         ELSE
            // Check if it is a "ON" command
            extract(format:="ON \1", str:=Command);
            IF extract.match AND extract.v1 >= 1 AND extract.v1 <= 8 THEN
               DOut[extract.v1] := ON;
               str:=strFormat(format:="Setting output \1 to ON",
v1:=extract.v1);
               DebugMsg(message:=str);
               SendData(id:=id, str:=strConcat(str1:=str, str2:="$N"));
            END_IF;

            // Check if it is a "OFF" command
            extract(format:="OFF \1", str:=Command);
            IF extract.match AND extract.v1 >= 1 AND extract.v1 <= 8 THEN
               Dout[extract.v1] := OFF;
               str:=strFormat(format:="Setting output \1 to OFF",
v1:=extract.v1);
               DebugMsg(message:=str);
               SendData(id:=id, str:=strConcat(str1:=str, str2:="$N"));
            END_IF;

            // Check if it is a "INPUT" command
            IF strCompare(str1:="INPUT", str2:=Command)=0 THEN
               FOR i:=1 TO 4 DO
                  str:=strFormat(format:="$NInput \1 is \2", v1:=i,
v2:=INT(Din[i]));
```

```
                    SendData(id:=id, str:=str);
                END_FOR;
            END_IF;

            // Check if it is a "HELP"/"?" command
            IF strCompare(str1:="HELP", str2:=Command)=0 OR
               strCompare(str1:="?", str2:=Command)=0 THEN
               SendData(id:=id, str:="Available commands:$N");
               SendData(id:=id, str:="  <input> : Show status of digital
inputs$N");
               SendData(id:=id, str:="  <on n> : Set digital output n to
ON$N");
               SendData(id:=id, str:="  <off n> : Set digital output n to
OFF$N");
               SendData(id:=id, str:="  <quit>  : Close connection$N");
               SendData(id:=id, str:="  <help>/<?> : Show available
commands$N");
            END_IF;

            // Send a new prompt
            SendData(id:=id, str:="$NEnter command> ");
         END_IF;
      END_IF;
   END_IF;

   Sleep(delay:=50);
   // Update the LED
   LED:=NOT LED;

END;
END_PROGRAM;
```

6.20 Examples - Telnet server2

```
//--------------------------------------------------------------------------
-
// telnet.vpl, created 2018-03-15 10:39
//
// This is a VERY simple telnet server. It has only a few commands, but still
// shows how to establish a TCP/IP connection.
// When a Telnet client connects to the RTCU, a welcome message is shown,
// together with a prompt, where the telnet client then can enter commands.
// The following commands are available:
// input, on n, off n, quit and help. Input will show the state
// of 4 digital inputs, on/off n will switch digital output n to either on or
off
// state, quit will close the connection, and help will show a list of
available
// commands.
//--------------------------------------------------------------------------
-
INCLUDE rtcu.inc

//  Input variables that can be configured via the configuration dialog (These
are global)
VAR_INPUT
   din      : ARRAY [1..4] OF BOOL; | Digital inputs
END_VAR;

//  Output variables that can be configured via the configuration dialog
(These are global)
VAR_OUTPUT
   dout     : ARRAY [1..4] OF BOOL; | Digital outputs
   led      : ARRAY [1..3] OF BOOL;
END_VAR;

//--------------------------------------------------------------------------
// Send a STRING on a TCP/IP connection
//--------------------------------------------------------------------------
FUNCTION SendData;
VAR_INPUT
   handle   : SYSHANDLE;
   str      : STRING;
END_VAR;
VAR
   txbuf    : ARRAY [0..100] OF SINT;
   len      : INT;
   size     : DINT;
END_VAR;
   len := strLen(str := str);
   strToMemory(dst := ADDR(txbuf), str := str, len := len);
   DebugFmt(message:="soSend = \1", v1 := soSend(socket:=handle,
data:=ADDR(txbuf), size:=len, sent:=size));
END_FUNCTION;

//--------------------------------------------------------------------------
// Thread which handles client connections
//--------------------------------------------------------------------------
THREAD_BLOCK TB_TELNET_CLIENT;
VAR_INPUT
   handle   : SYSHANDLE;
END_VAR;
VAR_OUTPUT
   inuse    : BOOL;
```

```
END_VAR;
VAR
   buf      : ARRAY [1..200] OF SINT;
   size     : DINT;
   sent     : DINT;
   rc, i    : INT;

   extract  : strGetValues;
   str      : STRING;
   Partial  : STRING;
   Command  : STRING;
   unknown  : BOOL;
END_VAR;

   // The thread is in use
   inuse := TRUE;

   // Send welcome message
   SendData(handle := handle, str := "Welcome to the RTCU Telnet
Server.$N$N");
   SendData(handle := handle, str := "Available commands:$N");
   SendData(handle := handle, str := "  <input>  : Show status of digital
inputs$N");
   SendData(handle := handle, str := "  <output> : Show status of digital
outputs$N");
   SendData(handle := handle, str := "  <on n>   : Set digital output n to
ON$N");
   SendData(handle := handle, str := "  <off n>  : Set digital output n to
OFF$N");
   SendData(handle := handle, str := "  <quit>   : Close connection$N");
   SendData(handle := handle, str := "  <help>   : Show available
commands$N");
   SendData(handle := handle, str := "$NEnter command> ");

// Main loop
WHILE TRUE DO
   // Wait for data
   rc := soRecv(
               socket  := handle,
               data    := ADDR(buf),
               maxsize := SIZEOF(buf),
               size    := size
              );
   IF rc < 1 THEN
      DebugFmt(message := "soRecv = \1", v1 := rc);
      soClose(socket := handle);
      inuse := FALSE;
      RETURN;
   END_IF;

   // Echo the received data
   rc := soSend(socket := handle, data:=ADDR(buf), size := size, sent :=
sent);
   IF rc < 1 THEN
      DebugFmt(message := "soRecv = \1", v1 := rc);
      soClose(socket := handle);
      inuse := FALSE;
      RETURN;
   END_IF;

   // Convert the received data to a string
   str := strFromMemory(src := ADDR(buf), len := INT(size));
   DebugMsg(message := str);
```

```
   // concatenate the received data to command string
   Partial := strConcat(str1 := Partial, str2 := str);
   i := strFind(str1 := Partial, str2 := "$R");
   IF i > 0 THEN
      // We now have a complete command assembled in PartialCommand
      Command := strLeft(str := Partial, length := i - 1);
      DebugMsg(message := strConcat(str1 := "Command=", str2 := Command));
      Partial := "";
      unknown := ON;

      // Check if it is a "ON" command
      extract(format := "ON \1", str := Command);
      IF extract.match AND extract.v1 >= 1 AND extract.v1 <= 8 THEN
         unknown := OFF;
         dout[extract.v1] := ON;
         str := strFormat(format := "Setting output \1 to ON", v1 :=
extract.v1);
         DebugMsg(message := str);
         SendData(handle := handle, str := strConcat(str1 := str, str2 :=
"$N"));
      END_IF;

      // Check if it is a "OFF" command
      extract(format := "OFF \1", str := Command);
      IF extract.match AND extract.v1 >= 1 AND extract.v1 <= 8 THEN
         unknown := OFF;
         dout[extract.v1] := OFF;
         str := strFormat(format := "Setting output \1 to OFF", v1 :=
extract.v1);
         DebugMsg(message := str);
         SendData(handle := handle, str := strConcat(str1 := str, str2 :=
"$N"));
      END_IF;

      // Check if it is a "INPUT" command
      IF strCompare(str1 := "INPUT", str2 := Command) = 0 THEN
         unknown := OFF;
         FOR i := 1 TO 4 DO
            str := strFormat(format := "$NInput \1 is \2", v1 := i, v2 :=
INT(din[i]));
            SendData(handle := handle, str := str);
         END_FOR;
      END_IF;

      // Check if it is a "OUTPUT" command
      IF strCompare(str1 := "OUTPUT", str2 := Command) = 0 THEN
         unknown := OFF;
         FOR i := 1 TO 4 DO
            str := strFormat(format := "$NOutput \1 is \2", v1 := i, v2 :=
INT(dout[i]));
            SendData(handle := handle, str := str);
         END_FOR;
      END_IF;

      // Check if it is a "HELP" command
      IF strCompare(str1 := "HELP", str2 := Command) = 0 THEN
         unknown := OFF;
         SendData(handle := handle, str := "Available commands:$N");
         SendData(handle := handle, str := "  <input>  : Show status of
digital inputs$N");
         SendData(handle := handle, str := "  <output> : Show status of
digital outputs$N");
         SendData(handle := handle, str := "  <on n>   : Set digital output n
to ON$N");
```

```
         SendData(handle := handle, str := "  <off n>  : Set digital output n
to OFF$N");
         SendData(handle := handle, str := "  <quit>   : Close connection$N");
         SendData(handle := handle, str := "  <help>   : Show available
commands$N");
      END_IF;

      // Check if it is a "QUIT" command
      IF strCompare(str1 := "QUIT", str2 := Command) = 0 THEN
         unknown := OFF;
         SendData(handle := handle, str := "$NGoodbye.$N");

         // Diconnect
         soClose(socket := handle);
      END_IF;

      // Unknown command
      IF unknown THEN
         SendData(handle := handle, str := "Unknown command$N");
      END_IF;

      // Send a new prompt
      SendData(handle := handle, str := "$NEnter command> ");
   END_IF;

END_WHILE;
END_THREAD_BLOCK;

//  These are the global variables of the program
VAR
   client   : ARRAY [1..10] OF TB_TELNET_CLIENT;
END_VAR;

//-------------------------------------------------------------------------------
// Show information about incoming connection
//-------------------------------------------------------------------------------
FUNCTION ShowInfo;
VAR_INPUT
   handle   : SYSHANDLE;
END_VAR;
VAR
   local    : STRING;
   remote   : STRING;
   host     : STRING;
   port     : DINT;
   iface    : SINT;
END_VAR;
   // Get information
   soStatus(socket := handle, local := local, remote := remote);
   soAddrInetGet(address := remote, host := host, port := port);
   iface := soAddrToInterface(address := local);

   // Show information
   DebugMsg(message := "Incoming connection");
   DebugFmt(message := "    interface  = \1", v1 := iface);
   DebugMsg(message := "    IP address = " + host);
   DebugFmt(message := "    IP port    = \4", v4 := port);
END_FUNCTION;

//-------------------------------------------------------------------------------
// Thread which listens for incoming connections
//-------------------------------------------------------------------------------
THREAD_BLOCK TB_TELNET_SERVER;
VAR_INPUT
```

```
   port     : DINT;
END_VAR;
VAR
   handle   : SYSHANDLE;
   remote   : SYSHANDLE;
   inuse    : BOOL;
   address  : STRING;

   rc       : INT;
   i        : INT;
   size     : DINT;
   buf      : ARRAY [1..10] OF SINT;
END_VAR;

// Setup
strToMemory(dst := ADDR(buf), str := "EXIT$N", len := 6);

// Build listen address
soAddrInetSet(address := address, port := port);

WHILE TRUE DO
   // Open listen socket
   IF NOT inuse THEN
      // Create socket
      rc := soCreate(socket := handle);
      IF rc < 1 THEN
         DebugFmt(message := "soCreate = \1", v1 := rc);
         RETURN;
      END_IF;

      // Bind socket
      rc := soBind(socket := handle, address := address);
      IF rc < 1 THEN
         DebugFmt(message := "soBind = \1", v1 := rc);
         soClose(socket := handle);
         RETURN;
      END_IF;

      // Start listening
      rc := soListen(socket := handle);
      IF rc < 1 THEN
         DebugFmt(message := "soListen = \1", v1 := rc);
         soClose(socket := handle);
         RETURN;
      END_IF;

      inuse := TRUE;
   END_IF;

   // Accept
   rc := soAccept(socket := handle, remote := remote);
   IF rc = 1 THEN
      // Find free thread
      FOR i := 1 TO 10 DO
         IF NOT client[i].inuse THEN
            EXIT;
         END_IF;
      END_FOR;

      // Start thread
      IF NOT client[i].inuse THEN
         // Show information
         ShowInfo(handle := remote);
```

```
            // Start thread
            client[i](handle := remote);
         ELSE
            // Send denied
            soSend(
                   socket  := remote,
                   data    := ADDR(buf),
                   size    := 6,
                   sent    := size
                  );
            soClose(socket := remote);
         END_IF;
      ELSE
         DebugFmt(message := "soAccept = \1", v1 := rc);
         soClose(socket := handle);
         inuse := FALSE;
      END_IF;

      Sleep(delay := 250);
   END_WHILE;
   END_THREAD_BLOCK;

   PROGRAM Telnet;
   VAR
      telnet      : TB_TELNET_SERVER;
      val_con     : BOOL;
      old_con     : BOOL;
      i           : INT;

      iface       : SINT;
      port        : DINT;
   END_VAR;

      // Welcome
      DebugMsg(message := "----------------------------------------");
      DebugMsg(message := "  RTCU Telnet Server.");
      DebugMsg(message := "----------------------------------------");

      // Initialize
      DebugMsg(message := "Initialize...");
      iface    := _NET_IFACE_LAN1;
      port     := 5020;

      // Open network interface
      netOpen(iface := iface);

      // Wait for connection to the Internet
      // (This is actually not needed, as the soListen() can be called before
      // the connection is established)
      WHILE NOT netConnected(iface := iface) DO
         DebugMsg(message := "   Waiting for network connection");
         Sleep(delay := 3000);
      END_WHILE;
      old_con := ON;

      // Start server
      telnet(port := port);

      // Show our assigned IP address (this is the one the telnet client should
   connect to)
      DebugMsg(message := strConcat(str1 := "My IP Address= ", str2 :=
   sockIPToName(ip := sockGetLocalIP(iface := iface))));
      DebugFmt(message := "My IP Port=    \4", v4 := port);
      DebugMsg(message := "----------------------------------------");
```

```
   DebugMsg(message := "Running.");

BEGIN
   // Update interface status
   val_con := netConnected(iface := iface);
   IF val_con <> old_con THEN
      IF val_con THEN
         DebugMsg(message := "Network: Connected");
      ELSE
         DebugMsg(message := "Network: Connection lost");
      END_IF;
   END_IF;
   old_con := val_con;

   // Update LEDS
   led[1]  := val_con;
   led[2]  := telnet._running;
   led[3]  := OFF;
   FOR i := 1 TO 10 DO
      IF client[i].inuse THEN led[3] := ON; END_IF;
   END_FOR;

   // Delay
   Sleep(delay := 250);
END;
END_PROGRAM;
```

6.21 Examples - Web client

```
//-------------------------------------------------------------------------
-
// web client.vpl, created 2018-03-16 09:58
//
// This is a VERY simple web client. It connects to a web server and downloads
// a web page, which is stored on the internal drive.
// The example shows how to establish a TCP/IP connection, how to enable
// a secure connection, and how to dictate which network interface is used.
//-------------------------------------------------------------------------
-
INCLUDE rtcu.inc

//  These are the global variables of the program
VAR
   buf      : ARRAY [1..1024] OF SINT;
   timer    : TON;
END_VAR;

////////////////////////////////////////////////////////////////////////////
// Opens a connection to the server
//
// Input:
//    host      - The server domain name
//    port      - The server port number
//    iface     - The network interface to use
//    https     - Use HTTP (FALSE) or HTTPS (TRUE) to get the page
//
// Returns:
//    0         - Success
//    -1        - Socket error
//    -2        - Could not connect to server
FUNCTION server_open : INT;
VAR_INPUT
   host     : STRING;
   port     : DINT;
   iface    : SINT;
   https    : BOOL;
   handle   : ACCESS SYSHANDLE;
END_VAR;
VAR
   address  : STRING;
   rc       : INT;
END_VAR;

   // Open socket
   rc := soCreate(socket := handle);
   IF rc < 1 THEN
      // Socket error
      DebugFmt(message := "soCreate = \1", v1 := rc);
      server_open := -1;
      RETURN;
   END_IF;

   // Bind to network interface
   IF iface > 0 THEN
      // Get the address of the network interface
      address := soAddrFromInterface(iface := iface);

      // Bind the socket
      rc := soBind(socket := handle, address := address);
```

```
      IF rc < 1 THEN
         // Socket error
         DebugFmt(message := "soBind = \1", v1 := rc);
         soClose(socket := handle);
         server_open := -1;
         RETURN;
      END_IF;
   END_IF;

   // Enable secure connection
   IF https THEN
      rc := soConfigTLS(
                        socket := handle,
                        enable := TRUE
                       );
      IF rc < 1 THEN
         // Socket error
         DebugFmt(message := "soConfigTLS = \1", v1 := rc);
         soClose(socket := handle);
         server_open := -1;
         RETURN;
      END_IF;
   END_IF;

   // Generate socket address of destination
   rc := soAddrInetSet(address := address, host := host, port := port);
   IF rc < 1 THEN
      // Not a valid address
      DebugFmt(message := "soAddrInetSet = \1", v1 := rc);
      soClose(socket := handle);
      server_open := -2;
      RETURN;
   END_IF;

   // Connect to server
   rc := soConnect(socket := handle, address := address);
   IF rc < 1 THEN
      // Not a valid address
      DebugFmt(message := "soConnect = \1", v1 := rc);
      soClose(socket := handle);
      server_open := -2;
      RETURN;
   END_IF;

   // Switch to non-blocking, as some servers leave the socket open
   rc := soConfigNonblock(socket := handle, enable := TRUE);
   IF rc < 1 THEN
      // Not a valid address
      DebugFmt(message := "soConfigNonblock = \1", v1 := rc);
      soClose(socket := handle);
      server_open := -1;
      RETURN;
   END_IF;
END_FUNCTION;
////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
// Read response header from server
//
// Input:
//    handle    - The socket connection
//
// Returns:
//    HTTP response code
```

```
FUNCTION read_line : INT;
VAR_INPUT
   handle   : SYSHANDLE;
   line     : ACCESS STRING;
END_VAR;
VAR
   ch       : SINT;
   size     : DINT;
   state    : INT;
   rc       : INT;
END_VAR;

   // Default
   line  := "";
   state := 0;
   timer(pt := 10000, trig := ON);

   // Iterate incoming data
   WHILE TRUE DO
      // Read character
      rc := soRecv(
                   socket  := handle,
                   data    := ADDR(ch),
                   maxsize := 1,
                   size    := size
                  );
      IF rc < 1 AND rc <> -4 THEN
         DebugFmt(message := "soRecv=\1", v1 := rc);
         RETURN;
      END_IF;

      // Handle character
      IF rc <> -4 THEN
         IF ch = 16#0A OR ch = 16#0D THEN
            state := state + 1;
         ELSE
            line  := line + strFromMemory(src := ADDR(ch), len := 1);
         END_IF;
         timer(trig := OFF);
      END_IF;

      timer(trig := ON);
      IF timer.q THEN
         // Timeout waiting for data
         RETURN;
      END_IF;
      IF state = 2 THEN
         EXIT;
      END_IF;
   END_WHILE;

   // Completed
   read_line := 1;
END_FUNCTION;
////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
// Read response header from server
//
// Input:
//    handle    - The socket connection
//
// Returns:
//    HTTP response code
```

```
FUNCTION reply_header : INT;
VAR_INPUT
   handle   : SYSHANDLE;
END_VAR;
VAR
   line     : STRING;
   http     : INT;
   rc       : INT;
END_VAR;

// Iterate response
WHILE TRUE DO
   // Read line
   rc := read_line(handle := handle, line := line);
   IF rc < 1 THEN
      RETURN;
   END_IF;

   // Check for header end
   IF strLen(str := line) = 0 THEN
      // The header is received
      EXIT;
   END_IF;

   // Check for HTTP reply
   IF strCompare(str1 := strLeft(str := line, length := 5), str2 := "http/") =
0 THEN
      http := strToInt(str := strMid(str := line, start := 10));
   END_IF;
END_WHILE;

   // Completed
   reply_header := http;
END_FUNCTION;
////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
// Read response body from server
//
// Input:
//    handle    - The socket connection
//
// Returns:
//    HTTP response code
FUNCTION reply_body : INT;
VAR_INPUT
   handle   : SYSHANDLE;
   filename : STRING;
END_VAR;
VAR
   rc       : INT;
   fd       : FILE;
   sent     : DINT;
END_VAR;

   // Open file
   fd := fsFileCreate(name := filename);

   // Restart receive timer
   timer(trig := OFF);
   timer(pt := 10000, trig := ON);

   // Loop until no more data
   WHILE TRUE DO
```

```
      rc := soRecv(socket := handle, data := ADDR(buf), maxsize :=
SIZEOF(buf), size := sent);
      IF rc < 1 AND rc <> -3 AND rc <> -4 THEN
         // Communication error
         DebugFmt(message := "soRecv=\1", v1 := rc);
         fsFileClose(fd := fd);
         RETURN;
      END_IF;
      IF rc = -3 OR sent = 0 THEN
         EXIT;
      END_IF;

      // Write to file
      IF rc <> -4 THEN
         fsFileWrite(fd := fd, buffer := ADDR(buf), length := INT(sent));
         timer(trig := OFF);
      END_IF;

      // Timer
      timer(trig := ON);
      IF timer.q THEN
         // Timeout waiting for data
         EXIT;
      END_IF;
   END_WHILE;

   // Close file
   fsFileClose(fd := fd);

   // Completed
   reply_body := 1;
END_FUNCTION;
////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
// Builds the HTTP GET command
//
// Input:
//    host      - The server domain name
//    page      - The path to the page on the server
//
// Returns:
//    String containing the HTTP GET command
FUNCTION build_http_get : STRING;
VAR_INPUT
   host     : STRING;
   page     : STRING;
END_VAR;
VAR
   cmd      : STRING;
END_VAR;
   // HTTP command
   cmd  := "GET ";

   // Add the page
   cmd  := cmd + page;

   // Add HTTP
   cmd  := cmd + " HTTP/1.1$N";

   // Add IP address
   cmd  := cmd + "Host: ";
   cmd  := cmd + host;
   cmd  := cmd + "$N";
```

```
   // Terminate command
   cmd  := cmd + "$N";

   build_http_get := cmd;
END_FUNCTION;
////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
// Get a WEB page and store it as a file
//
// Input:
//    host      - The server domain name
//    port      - The server port number
//    iface     - The network interface to use
//    page      - The path to the page on the server
//    https     - Use HTTP (FALSE) or HTTPS (TRUE) to get the page
//    filename  - The file name where the page is stored
//
// Returns:
//    0         - Success.
//    -1        - Could not connect to server
//    -2        - Failed to send request
//    -3        - Failed to receive web page
FUNCTION GetPage : INT;
VAR_INPUT
   host     : STRING;
   port     : DINT := 80;
   iface    : SINT;
   page     : STRING;
   filename : STRING;
   https    : BOOL;
END_VAR;
VAR
   handle   : SYSHANDLE;
   sent     : DINT;
   len      : INT;
   rc       : INT;
   request  : STRING;
END_VAR;

   // Generate HTTP request
   request := build_http_get(host := host, page := page);

   // Open a socket to the server
   DebugMsg(message := "Connecting...");
   rc := server_open(host := host, port := port, iface := iface, https :=
https, handle:= handle);
   IF rc < 0 THEN
      DebugFmt(message := "server_open=\1", v1 := rc);
      GetPage := -1;
      RETURN;
   END_IF;

   // Send request
   DebugMsg(message := "Send request...");
   len := strLen(str := request);
   strToMemory(dst := ADDR(buf), str := request, len := len);
   rc := soSend(socket := handle, data := ADDR(buf), size := len, sent :=
sent);
   IF rc < 1 THEN
      // Communication error
      DebugFmt(message := "soSend=\1", v1 := rc);
      soClose(socket := handle);
```

```
      GetPage := -2;
      RETURN;
   END_IF;

   // Receive response
   DebugMsg(message := "Wait for reply...");
   rc := reply_header(handle := handle);
   IF rc < 1 THEN
      // Communication error
      DebugFmt(message := "reply_header=\1", v1 := rc);
      soClose(socket := handle);
      GetPage := -3;
      RETURN;
   END_IF;
   DebugFmt(message := "Reply = \1", v1 := rc);

   // Receive web page
   rc := reply_body(handle := handle, filename := filename);
   IF rc < 1 THEN
      // Communication error
      DebugFmt(message := "reply_body=\1", v1 := rc);
      soClose(socket := handle);
      GetPage := -3;
      RETURN;
   END_IF;

   // Close
   soClose(socket := handle);
END_FUNCTION;
////////////////////////////////////////////////////////////////////////////////

PROGRAM web_client;
VAR
   rc : INT;
END_VAR;

   DebugMsg(message := "----------------------------------------");
   DebugMsg(message := "Initialize...");

   // Open filesystem
   fsMediaOpen(media := 1);
   Sleep(delay := 2000);
   fsDirChange(path := "B:/");

   // Open network interface
   netOpen(iface := _NET_IFACE_LAN1);

   // Wait for network connection
   WHILE NOT netConnected(iface := _NET_IFACE_LAN1) DO
      DebugMsg(message := "Wait for connection...");
      Sleep(delay := 3000);
   END_WHILE;

   DebugMsg(message := "----------------------------------------");
   rc := GetPage(
                 host      := "www.google.dk"
                 ,https    := ON
                 ,port     := 443
                 ,iface    := _NET_IFACE_LAN1
                 ,page     := "/"
                 ,filename := "index.htm"
                );
   DebugFmt(message := "GetPage=\1", v1 := rc);
```

```
BEGIN
END;
END_PROGRAM;
```

6.22 Examples - VSMS using the RTCU Communication Hub (simple)

```
//------------------------------------------------------------------------
-
// testvsms.vpl, created 2003-04-28 09:44
//
// Test of VSMS messages sent over the RTCU Gateway. In order to run this
// example, you need the following:
// 1. Install and configure the RTCU Gateway product
// 2. Insert a GPRS enabled SIM card in the RTCU device
// 3. Configure the TCP/IP and Gateway settings in the RTCU, using the RTCU
IDE program
// 4. Download this project into the RTCU device
//
// When running correctly the LED1 will turn on after a few seconds, and then
LED2
// should turn on after approx 30..50 seconds signalling that the device is
now connected
// to the GPRS network (connected to the internet)
// The program will then send a VSMS message every 60 seconds to it self.
//------------------------------------------------------------------------
-
INCLUDE rtcu.inc

//  Output variables that can be configured via the configuration dialog
(These are global)
VAR_OUTPUT
   gwOK   : BOOL; | Monitors connection to the RTCU Gateway
   gprsOK : BOOL; | Monitors connection to GPRS (The internet) connection
END_VAR;

PROGRAM testvsms;
VAR
   rc        : INT;
   sms       : gsmIncomingSMS; // Receives incoming VSMS messages
   ns        : DINT;   // Holds linsec time for next transmission
   sMyNodeID : STRING; // Contains this devices serialnumber (node ID) in the
                       // format of "@nnnn" where nnnn is the serialnumber
END_VAR;

// The next code will only be executed once after the program starts
gsmPower(power:=TRUE);

// Activate the GPRS support, and connect to the internet
DebugFmt(message:="gprsOpen=\1",v1:=gprsOpen());

// First VSMS will be sent in 60 seconds
ns:=clockNow()+60;

// Construct our own nodeid in the form of "@nnnn"
sMyNodeID:=strConcat(str1:="@", str2:=dintToStr(v:=boardSerialNumber()));
DebugMsg(message:=sMyNodeID);

// Code from this point until END will be executed repeatedly
BEGIN
   // Update function block
   sms();

   // If it's time to send another VSMS message (this message is destined for
this device)...
   if clockNow()>ns THEN
```

```
      DebugFmt(message:="gsmSendSMS()=\1",
v1:=gsmSendSMS(phonenumber:=sMyNodeID, message:="Hello world"));
      ns:=clockNow()+60; // Next transmit is in 60 seconds
   END_IF;

   // If any incoming VSMS messages, just show them
   if sms.status>0 THEN
      DebugMsg(message:="SMS received");
      DebugMsg(message:=sms.phonenumber); // In the case of VSMS messages,
this will be "@nnnn"
                                          // where nnnn is the nodenumber of
the sending node
      DebugMsg(message:=sms.message);
   END_IF;

   gprsOK:=gprsConnected();
   gwOK:=gwConnected();
END;

END_PROGRAM;
```

6.23 Examples - Mobile tracking (simple)

```
//--------------------------------------------------------------------------
-
// GPS Tracker.vpl, created 2003-04-28 13:02
//
// Program that sends a VSMS PDU message each 10 seconds, IF it has a valid
gps position
//
// In order to run this example, you need the following:
// 1. Install and configure the RTCU Gateway product
// 2. Insert a GPRS enabled SIM card in the RTCU device
// 3. Configure the TCP/IP and Gateway settings in the RTCU, using the RTCU
IDE program
// 4. Modify the variable "DestNodeID" to the correct nodeid for the receiver
// 5. Download this project into the RTCU device
//
// When running correctly, LED1 will turn on after 30..50 seconds, signalling
that
// the device is now connected to the RTCU Gateway. LED2 will blink every time
// the device gets a correct GPS position (2D or 3D)
// The program will then send a VSMS message every 10 seconds to the
"DestNodeID"
// with information about the current GPS time (in UTC) and
lattitude/longitude
//
// Protocol:
//   <gpstime> <lattitude> <longitude>
// where all 3 fields are 4 byte words, little endian format, 12 bytes in
total
//
//--------------------------------------------------------------------------
-
INCLUDE rtcu.inc

VAR_OUTPUT
   gpsOK : BOOL; | Monitors valid GPS fix
   gwOK  : BOOL; | Monitors connection to the RTCU Gateway
END_VAR;

// These are the global variables of the program
VAR
   DestNodeID : STRING:="@4711"; // The nodeid of the receiver
   TXBuf      : ARRAY[0..139] OF SINT; // Holds the transmit buffer
   tSend      : DINT; // Linsec value of next time to send a VSMS message

   gps        : gpsFix; // GPS position information
   Lat        : DINT; // Lattitude
   Lon        : DINT; // Longitude
   gpsTime    : DINT; // GPS time (in UTC !)
END_VAR;

//--------------------------------------------------------------------------
// Misc. helper functions
// Pack/Unpack INT and DINT's from an array of SINT
//--------------------------------------------------------------------------
FUNCTION getINT:INT;
VAR_INPUT
   adr : ptr;
END_VAR;
memcpy(dst:=ADDR(getInt),src:=adr,len:=SIZEOF(getINT));
END_FUNCTION;
```

```
FUNCTION getDINT:DINT;
VAR_INPUT
   adr : ptr;
END_VAR;
memcpy(dst:=ADDR(getDINT),src:=adr,len:=SIZEOF(getDINT));
END_FUNCTION;

FUNCTION setINT;
VAR_INPUT
   adr : ptr;
   v : INT;
END_VAR;
memcpy(dst:=adr,src:=ADDR(v),len:=SIZEOF(v));
END_FUNCTION;

FUNCTION setDINT;
VAR_INPUT
   adr : ptr;
   v : DINT;
END_VAR;
memcpy(dst:=adr,src:=ADDR(v),len:=SIZEOF(v));
END_FUNCTION;

//---------------------------------------------------------------------------


PROGRAM GPS_Tracker;

// Switch power on to the GSM and GPS modules
gpsPower(power:=TRUE);
gsmPower(power:=TRUE);

// Send next position in 10 seconds
tSend:=clockNow() + 10;

// Activate the GPRS support, and connect to the internet
DebugFmt(message:="gprsOpen=\1", v1:=gprsOpen());

BEGIN
   gps();
   // If time to send a new position...
   IF clockNow() > tSend THEN
      // If we have a 2D or 3D fix, go build buffer, and send as a PDU SMS
message
      IF gps.mode > 1 THEN
         // Calculate next time we need to send
         tSend:=clockNow() + 10;

         // Get GPS time as linsec value
         gpsTime:=gps.linsec;

         // Get Lat/Lon as two DINT values
         Lat:=gps.latitude;        //  ddmm.mmmm * 10000
         Lon:=gps.longitude;         // dddmm.mmmm * 10000

         // Show the calculated values in the debug window
         DebugFmt(message:="gpsTime=\4", v4:=gpsTime);
         DebugFmt(message:="Lattitude=\4", v4:=Lat);
         DebugFmt(message:="Longitude=\4", v4:=Lon);

         // Pack lattitude, longitude and GPS time into the TX buffer (as
little endians !)
         setDint(adr:=ADDR(TXBuf[0]), v:=gpsTime);
```

```
            setDint(adr:=ADDR(TXBuf[4]), v:=Lat);
            setDint(adr:=ADDR(TXBuf[8]), v:=Lon);

            // demonstrates how to extract data from a "raw" buffer (not needed
in this example)
            //DebugFmt(message:="gpsTime=\4", v4:=getDINT(adr:=ADDR(TXBuf[0])));
            //DebugFmt(message:="Lattitude=\4",
v4:=getDINT(adr:=ADDR(TXBuf[4])));
            //DebugFmt(message:="Longitude=\4",
v4:=getDINT(adr:=ADDR(TXBuf[8])));

            // Send a binary SMS message with the contents to the destination
node (via the RTCU Gateway)
            DebugFmt(message:="gsmSendPDU=\1",
v1:=gsmSendPDU(phonenumber:=DestNodeID, message:=ADDR(TXBuf), length:=12));

            // Enable the next line to get the PDU message in the Device->SMS
messages window also
            //gsmSendPDU(phonenumber:="9999", message:=ADDR(TXBuf), length:=12);
         END_IF;
      END_IF;

      gwOK:=gwConnected();

      // If valid fix from GPS, change state of LED
      IF gps.mode > 1 THEN
         gpsOK:=NOT gpsOK;
      END_IF;

END;
END_PROGRAM;
```

6.24 Examples - Socket communication (Advanced)

```
//------------------------------------------------------------------------
-
// test.vpl, created 2001-10-31 12:31
//
// Simple GPRS test program. The program tries to connect to an "echo host".
// The echo host will echo all data received on port 5005 back to the sender.
// The program connects to the echo host, and send a packet, which is then
// echoed back from the echo server. This is done 100 times, then the program
// will disconnect the socket, and try to connect again. It will then send
// another 100 frames and so on. If the 'restart' input is activated, the
// program will disconnect and then connect again to the echo host.
// Please note that the echo server program is located in the same directory
// as the "Socket Example" project, under the My documents\RTCU
Projects\Examples\
// directory, and is called echos.exe ("C" source also included in echos.c).
// This program must be started on a PC with a fixed IP address. This IP
address
// (or symbolic name) must then be put into the 'host' variable (see below,
// currently set to "rtcu.dk")
//------------------------------------------------------------------------
-
INCLUDE rtcu.inc

//  Input variables that can be configured via the configuration dialog (These
are global)
VAR_INPUT
   restart : BOOL R_EDGE; | Restart connection (disconnect/connect)
END_VAR;

//  Output variables that can be configured via the configuration dialog
(These are global)
VAR_OUTPUT
   toggle : BOOL; | LED that indicate when data is received from echo host
   gprs   : BOOL; | LED that indicates that a GPRS connection is present.
END_VAR;

//  These are the global variables of the program
VAR
   sockrcv  : sockReceive;    // Receive data on a socket
   soccon   : sockConnection; // Create a socket connection
   id       : SINT; // ID of the connection
   buffer   : ARRAY[0..300] OF SINT; // Receive buffer
   teststr1 : STRING :="Hello world. This is a test of the RTCU SOCKET
Interface";
   teststr2 : STRING :="Hello world. The RTCU Concept is the most powerfull
GSM Telemetry platform in the world!!";
   host     : STRING :="rtcu.dk"; // The echo host (can also be a dotted
format IP address)
   port     : INT :=5005; // The echo host port number.
   iter     : DINT; // Number of iterations
   rc       : INT;  // Return codes from socket calls
END_VAR;


//------------------------------------------------------------------------
-
//
//------------------------------------------------------------------------
-
PROGRAM test;
```

```
DebugMsg(message:="GPRS Socket test-program started.");

// Turn on power to GSM module
gsmPower(power:=ON);

// Open the GPRS connetion
rc:=gprsOpen();
DebugFmt(message:="gprsOpen()=\1",v1:=rc);

// Wait for GPRS connected (after this, we are connected to the Internet)
WHILE NOT gprsConnected() DO
   DebugMsg(message:="Waiting for cellular connection");
   Sleep(delay:=2500);
END_WHILE;

// Connect to the echo host, using port 5005
id:=sockConnect(ip:=sockIPFromName(str:=host),port:=port);

DebugFmt(message:="sockConnect()=\1",v1:=id);

// Initialize the sockConnection() functionblock
soccon.id:=id;

// And intialize the socket receiver
sockrcv.id:=id;
sockrcv.data:=ADDR(buffer);
sockrcv.maxsize:=SIZEOF(buffer);

BEGIN
   gprs:=gprsConnected();
   soccon();  // Update sockConnection() functionblock
   sockrcv(); // Update sockReceive() functionblock

   // If restart switch activated, disconnect the old connection, and make a
new one
   IF restart THEN
      // Disconnect old socket
      sockDisconnect(id:=soccon.id);
      // Lookup IP address of echo host, and make a connection to it
      id:=sockConnect(ip:=sockIPFromName(str:=host),port:=port);
      DebugFmt(message:="sockConnect()=\1",v1:=id);
      // Update sockConnection()/sockReceive() functionblocks with the new
connection ID
      soccon.id:=id;
      sockrcv.id:=id;
   END_IF;

   // If connection status changed on socket
   IF soccon.changed THEN
      // ...and we are now connected, go send data to echo host
      // (This will "prime" the conversation, after this the data are "re-
used"
      // as the echo server gets its own data back again (see sockrcv.ready
below))
      IF soccon.Connected THEN
         DebugMsg(message:="Socket connected");
         DebugMsg(message:=sockIPToName(ip:=soccon.remoteIP));
         strToMemory
(dst:=ADDR(buffer),str:=teststr1,len:=strLen(str:=teststr1));
         rc:=sockSend(id:=id,data:=ADDR(buffer),size:=strLen(str:=teststr1));
         DebugFmt(message:="sockSend rc=\1",v1:=rc);
         strToMemory
(dst:=ADDR(buffer),str:=teststr2,len:=strLen(str:=teststr2));
         rc:=sockSend(id:=id,data:=ADDR(buffer),size:=strLen(str:=teststr2));
```

```
            DebugFmt(message:="sockSend rc=\1",v1:=rc);
            iter:=0;
         ELSE
            // ...and we are now disconnected, go make a new connection attempt
            DebugMsg(message:="Socket disconnected");
            // Disconnect old socket
            sockDisconnect(id:=soccon.id);
            // Let the sockConnection() see that we are disconnected
            soccon();
            // Lookup IP address of echo host, and make a connection to it
            id:=sockConnect(ip:=sockIPFromName(str:=host),port:=port);
            DebugFmt(message:="sockConnect=\1",v1:=id);
            // Update sockConnection()/sockReceive() functionblocks with the new
connection ID
            soccon.id:=id;
            sockrcv.id:=id;
         END_IF;
      END_IF;

      // If received data on the socket (echo data from echo server)
      IF sockrcv.ready THEN
         iter:=iter+1;
         DebugFmt(message:="\4 - data received len=\1", v4:=iter,
v1:=sockrcv.size);
         toggle:=NOT toggle; // Change the state of the LED
         // After 100 iterations, we Diconnect the socket. This will activate
         // the code above (soccon.changed) and then try to connect the socket
again
         IF iter < 100 THEN
            // Send the data we just received from the echo server, back again
            sockSend(id:=id, data:=ADDR(buffer), size:=sockrcv.size);
         ELSE
            sockDisconnect(id:=soccon.id);
         END_IF;
      END_IF;
   END;

   END_PROGRAM;
```

6.25 Examples - RS485 Network - Master (Advanced)

```
//--------------------------------------------------------------------------
-
// Master.vpl, created 2003-04-07 15:44
//
// This is a small demonstration program, that shows how to communicate using
// the serial routines (via RS485) in the RTCU devices. Two projects are
involved,
// Master and Slave.
//
// Format:
// To Slave FROM Master:
// <STX> <SRC> <DST> <DO Data-LSB> <DO Data-MSB>
//                   <AO-1-LSB> <AO-1-MSB>
//                   <AO-2-LSB> <AO-2-MSB>
//                   <AO-3-LSB> <AO-3-MSB>
//                   <AO-4-LSB> <AO-4-MSB> <ETX>
//
//
// To Master FROM Slave:
// <STX> <SRC> <DST> <DI Data-LSB> <DI Data-MSB>
//                   <AI-1-LSB> <AI-1-MSB>
//                   <AI-2-LSB> <AI-2-MSB>
//                   <AI-3-LSB> <AI-3-MSB>
//                   <AI-4-LSB> <AI-4-MSB> <ETX>
//
//
// STX = 0xFF
// ETX = 0xFE
// STUFF = 0x1B
// <DO-Data-LSB>/<DO-Data-MSB> = New status for digital outputs
// <AO-n-LSB>/<AO-n-MSB> = New data for analog output n
//
// <DI-Data-LSB>/<DI-Data-MSB> = New status for digital inputs
// <AI-n-LSB>/<AI-n-MSB> = New data for analog input n
//
//--------------------------------------------------------------------------
-
INCLUDE rtcu.inc

// Input variables that can be configured via the configuration dialog (These
are global)
VAR_INPUT
   SerialPort : SINT; | Serial port to communicate on (0=service port, 1=port
2)
   MyNodeID   : SINT; | This devices nodeid
   DestNodeID : SINT; | NodeID of remote device
END_VAR;

// Output variables that can be configured via the configuration dialog (These
are global)
VAR_OUTPUT
   RXLed : BOOL; | changes state after each receive (with correct nodeid)
END_VAR;

// These are the global variables of the program
VAR
   xAI : ARRAY[1..4]  OF INT; // Analog inputs values from remote device
   xDI : ARRAY[1..12] OF BOOL;// Digital inputs values from remote device
   xAO : ARRAY[1..4]  OF INT; // Analog output values to remote device
   xDO : ARRAY[1..12] OF BOOL;// Digital output values to remote device
```

```
END_VAR;

//-------------------------------------------------------------------------------
// Misc. helper functions
// Pack/Unpack INT and DINT's from an array of SINT
//-------------------------------------------------------------------------------
FUNCTION getINT:INT;
VAR_INPUT
   adr : ptr;
END_VAR;
   memcpy(dst:=ADDR(getInt),src:=adr,len:=SIZEOF(getINT));
END_FUNCTION;

FUNCTION setINT;
VAR_INPUT
   adr : ptr;
   v : INT;
END_VAR;
   memcpy(dst:=adr,src:=ADDR(v),len:=SIZEOF(v));
END_FUNCTION;
//-------------------------------------------------------------------------------

PROGRAM Master;
VAR
   RX       : serFrameReceiver;
   Len      : SINT;
   RxBuffer : ARRAY[0..63] of SINT;
   TxBuffer : ARRAY[0..63] of SINT;
   Index    : SINT; // Temporary variable/counter
   tAnswer  : TON;
   str      : STRING;
END_VAR;

serOpen(Port:=SerialPort, Baud:=57600, bit:=8, Parity:=0, RS485:=TRUE);
RX(port:=SerialPort, enable:=TRUE, frame:=ADDR(rxbuffer),
maxsize:=SINT(SIZEOF(rxbuffer)), stuffch:=16#10, sof:=16#02, eof:=16#03);
DebugMsg(message:="Master running");
tAnswer.pt:=1000;

//-------------------------------------------------------------------------
// In this demo, just set the outputs of the remote node to something
xAO[1]:=128;
xAO[2]:=256;
xAO[3]:=512;
xAO[4]:=1023;
FOR index:=1 TO 12 BY 2 DO
   xDO[index]:=TRUE;
END_FOR;
//-------------------------------------------------------------------------

BEGIN
   // Update function block
   tAnswer();

   Sleep(delay:=100);

   // Build transmit frame
   TXBuffer[0]:=MyNodeID; // Our nodeID
   TXBuffer[1]:=DestNodeID; // Destination nodeID
   // Set values for the 12 digital outputs
   TXBuffer[2]:=0;
   TXBuffer[3]:=0;
   FOR index:=0 TO 7 DO
      IF xDO[index+1] THEN
```

```
            TXBuffer[2]:=TXBuffer[2] OR shl8(in:=1, n:=index);
         END_IF;
      END_FOR;
      FOR index:=0 TO 3 DO
         IF xDO[index+9] THEN
            TXBuffer[3]:=TXBuffer[3] OR shl8(in:=1, n:=index);
         END_IF;
      END_FOR;
      // Set values for the 4 analog outputs
      setInt(adr:=ADDR(TXBuffer[4]), v:=xAO[1]);
      setInt(adr:=ADDR(TXBuffer[6]), v:=xAO[2]);
      setInt(adr:=ADDR(TXBuffer[8]), v:=xAO[3]);
      setInt(adr:=ADDR(TXBuffer[10]),v:=xAO[4]);
      // Send the request
      serSendData(port:=SerialPort, data:=ADDR(TXBuffer), size:=12,
stuffch:=16#10, sof:=16#02, eof:=16#03);

      // Wait for response or a timeout (1 Sec)
      tAnswer.trig:=TRUE;
      WHILE NOT RX.ready AND NOT tAnswer.q DO
         RX();
         tAnswer();
      END_WHILE;
      tAnswer.trig:=FALSE;

      // If a response was received
      IF RX.Ready THEN
         RXLed:=NOT RXLed;

         // and it is for this node...
         IF MyNodeID = RXBuffer[1] THEN
            // Get the values of the analog inputs
            xAI[1]:=getINT(adr:=ADDR(RXBuffer[4]));
            xAI[2]:=getINT(adr:=ADDR(RXBuffer[6]));
            xAI[3]:=getINT(adr:=ADDR(RXBuffer[8]));
            xAI[4]:=getINT(adr:=ADDR(RXBuffer[10]));
            // Get the values of the digital inputs
            FOR index:=0 TO 7 DO
               xDI[index+1]:=(RXBuffer[2] AND shl8(in:=1, n:=index)) <> 0;
            END_FOR;
            FOR index:=0 TO 3 DO
               xDI[index+9]:=(RXBuffer[3] AND shl8(in:=1, n:=index)) <> 0;
            END_FOR;

            //-------------------------------------------------------------------
---
            // In this demo, just show the inputs from the remote node
            DebugFmt(message:="Analog inputs: \1,\2,\3,\4", v1:=xAI[1],
v2:=xAI[2], v3:=xAI[3], v4:=xAI[4]);
            str:="";
            FOR index:=1 to 12 DO
               str:=strConcat(str1:=str, str2:=intToStr(v:=INT(xDI[index])));
            END_FOR;
            DebugMsg(message:=strConcat(str1:="Digital inputs: ", str2:=str));
            //-------------------------------------------------------------------
---

         ELSE
            DebugMsg(message:="Message not for this node !");
         END_IF;
         serFrameReceiveDone(port:=SerialPort);
      ELSE
         DebugMsg(message:="Timeout waiting for response !");
      END_IF;
```

```
END;

END_PROGRAM;
```

6.26 Examples - RS485 Network - Slave (Advanced)

```
//--------------------------------------------------------------------------
-
// Slave.vpl, created 2003-04-07 15:44
//
// This is a small demonstration program, that shows how to communicate using
// the serial routines (via RS485) in the RTCU devices. Two projects are
involved,
// Master and Slave.
//
//
// Format:
// To Slave FROM Master:
// <STX> <SRC> <DST> <DO Data-LSB> <DO Data-MSB>
//                   <AO-1-LSB> <AO-1-MSB>
//                   <AO-2-LSB> <AO-2-MSB>
//                   <AO-3-LSB> <AO-3-MSB>
//                   <AO-4-LSB> <AO-4-MSB> <ETX>
//
//
// To Master FROM Slave:
// <STX> <SRC> <DST> <DI Data-LSB> <DI Data-MSB>
//                   <AI-1-LSB> <AI-1-MSB>
//                   <AI-2-LSB> <AI-2-MSB>
//                   <AI-3-LSB> <AI-3-MSB>
//                   <AI-4-LSB> <AI-4-MSB> <ETX>
//
//
// STX = 0xFF
// ETX = 0xFE
// STUFF = 0x1B
// <DO-Data-LSB>/<DO-Data-MSB> = New status for digital outputs
// <AO-n-LSB>/<AO-n-MSB> = New data for analog output n
//
// <DI-Data-LSB>/<DI-Data-MSB> = New status for digital inputs
// <AI-n-LSB>/<AI-n-MSB> = New data for analog input n
//
//--------------------------------------------------------------------------
-
INCLUDE rtcu.inc

// Input variables that can be configured via the configuration dialog (These
are global)
VAR_INPUT
   SerialPort : SINT;             | Serial port to communicate on (0=service
port, 1=port 2)
   Din    : ARRAY[1..12] OF BOOL; | Digital inputs
   Ain    : ARRAY[1..4] OF INT;   | Analog inputs
   NodeSW : ARRAY[1..3] OF BOOL;  | Dipswitches for node id (binary, SW1=LSB,
SW3=MSB)
END_VAR;

// Output variables that can be configured via the configuration dialog (These
are global)
VAR_OUTPUT
   Dout  : ARRAY[1..12] OF BOOL; | Digital outputs
   Aout  : ARRAY[1..4] OF INT;   | Analog outputs
   RXLed : BOOL;                 | changes state after each receive (with
correct nodeid)
END_VAR;
```

```
//-------------------------------------------------------------------------------
// Misc. helper functions
// Pack/Unpack INT and DINT's from an array of SINT
//-------------------------------------------------------------------------------
FUNCTION getINT:INT;
VAR_INPUT
   adr : ptr;
END_VAR;
   memcpy(dst:=ADDR(getInt),src:=adr,len:=SIZEOF(getINT));
END_FUNCTION;

FUNCTION setINT;
VAR_INPUT
   adr : ptr;
   v : INT;
END_VAR;
   memcpy(dst:=adr,src:=ADDR(v),len:=SIZEOF(v));
END_FUNCTION;
//-------------------------------------------------------------------------------


PROGRAM Slave;
VAR
   MyNodeID : SINT;
   RX       : serFrameReceiver;
   Len      : SINT;
   RxBuffer : ARRAY[0..63] of SINT;
   TxBuffer : ARRAY[0..63] of SINT;
   Index    : SINT; // Temporary variable/counter
END_VAR;

MyNodeID:=SINT(NodeSW[3])*4 + SINT(NodeSW[2])*2 + SINT(NodeSW[1]);
DebugFmt(message:="Slave running, nodeid=\1", v1:=MyNodeID);

serOpen(Port:=SerialPort, Baud:=57600, bit:=8, Parity:=0, RS485:=TRUE);
RX(port:=SerialPort, enable:=TRUE, frame:=ADDR(rxbuffer),
maxsize:=SINT(SIZEOF(rxbuffer)), stuffch:=16#10, sof:=16#02, eof:=16#03);

BEGIN
   RX();
   // If a frame is received
   IF RX.Ready THEN
      // and it is for this node...
      IF (MyNodeID = RXBuffer[1] AND RX.Size=12) THEN
         // Set the digital outputs
         FOR index:=0 TO 7 DO
            DOut[index+1]:=(RXBuffer[2] AND shl8(in:=1, n:=index)) <> 0;
         END_FOR;
         FOR index:=0 TO 3 DO
            DOut[index+9]:=(RXBuffer[3] AND shl8(in:=1, n:=index)) <> 0;
         END_FOR;
         // Set the analog outputs
         AOut[1]:=getINT(adr:=ADDR(RXBuffer[4]));
         AOut[2]:=getINT(adr:=ADDR(RXBuffer[6]));
         AOut[3]:=getINT(adr:=ADDR(RXBuffer[8]));
         AOut[4]:=getINT(adr:=ADDR(RXBuffer[10]));
         RXLed:=NOT RXLed;

         // Build response
         TXBuffer[0]:=MyNodeID;
         TXBuffer[1]:=RXBuffer[0];
         // Build 2 bytes with status of digital inputs
         TXBuffer[2]:=0;
         TXBuffer[3]:=0;
```

```
            FOR index:=0 TO 7 DO
               IF DIn[index+1] THEN
                  TXBuffer[2]:=TXBuffer[2] OR shl8(in:=1, n:=index);
               END_IF;
            END_FOR;
            FOR index:=0 TO 3 DO
               IF DIn[index+9] THEN
                  TXBuffer[3]:=TXBuffer[3] OR shl8(in:=1, n:=index);
               END_IF;
            END_FOR;
            // Set analog values in response
            setInt(adr:=ADDR(TXBuffer[4]), v:=Ain[1]);
            setInt(adr:=ADDR(TXBuffer[6]), v:=Ain[2]);
            setInt(adr:=ADDR(TXBuffer[8]), v:=Ain[3]);
            setInt(adr:=ADDR(TXBuffer[10]),v:=Ain[4]);
            // Send the response
            serSendData(port:=SerialPort, data:=ADDR(TXBuffer), size:=12,
stuffch:=16#10, sof:=16#02, eof:=16#03);
         ELSE
            DebugMsg(message:="Message not for this node !");
         END_IF;
         serFrameReceiveDone(port:=SerialPort);
      END_IF;
END;

END_PROGRAM;
```

6.27 Examples - Thread Example

```
//--------------------------------------------------------------------------
-
// thread example.vpl, created 2004-10-15 10:23
//
// Simple program to demonstrate the use of threads, mutexes and I-Button.
// The program creates 3 threads, 1 to read the GPS position and store it
// in persistent memory, 1 to read an I-Button and control I-Button LED and 1
to
// Parse incoming SMS messages.
// By using an I-Button or sending a SMS the program will monitor the 4 inputs
// and send a SMS when one of them is activated. The monitoring must restart
// before another alarm is sent.
// A tracking function is also included, to be activated by a SMS.
//
// The following SMS commands are supported:
//  OWI#,nnnnnnnnnnnn: Register an I-Button ID for validation. #=Index(1..5),
nnnnn=I-Button ID
//  AON: Activate alarm monitoring
//  AOFF: Deactivate alarm monitoring
//  TON: Start tracking
//  TOFF: Stop tracking
//  POS: Get position
//  PHONEnnnnnnnn: Tracking and Alarm response phone number
//--------------------------------------------------------------------------
-
INCLUDE rtcu.inc

VAR_INPUT
   in    : ARRAY[1..4] OF BOOL;
END_VAR;

VAR_OUTPUT
   led : BOOL;
END_VAR;

VAR
   AlarmState : BOOL := FALSE;
   TrackState : BOOL := FALSE;

   mxStateAl  : MUTEX;
   mxStateTr  : MUTEX;

   NextGps    : DINT;
END_VAR;

//--------------------------------------------------------------------------
// SetAlarm
// Set the state of the AlarmState flag.
//--------------------------------------------------------------------------
FUNCTION SetAlarm;
VAR_INPUT
   state : BOOL;
END_VAR;
   mxLock(mx:=mxStateAl);
      AlarmState := state;
   mxUnlock(mx:=mxStateAl);
END_FUNCTION;
//--------------------------------------------------------------------------

//--------------------------------------------------------------------------
```

```
// TestAlarm
// Test the state of the AlarmState flag.
//-------------------------------------------------------------------------------
FUNCTION TestAlarm : BOOL;
   mxLock(mx:=mxStateAl);
      TestAlarm := AlarmState;
   mxUnlock(mx:=mxStateAl);
END_FUNCTION;
//-------------------------------------------------------------------------------

//-------------------------------------------------------------------------------
// SetTracking
// Set the state of the TrackState flag.
//-------------------------------------------------------------------------------
FUNCTION SetTracking;
VAR_INPUT
   state : BOOL;
END_VAR;
   mxLock(mx:=mxStateTr);
      TrackState := state;
   mxUnlock(mx:=mxStateTr);
END_FUNCTION;
//-------------------------------------------------------------------------------

//-------------------------------------------------------------------------------
// TestTracking
// Test the state of the TrackState flag.
//-------------------------------------------------------------------------------
FUNCTION TestTracking : BOOL;
   mxLock(mx:=mxStateTr);
      TestTracking := TrackState;
   mxUnlock(mx:=mxStateTr);
END_FUNCTION;
//-------------------------------------------------------------------------------

//-------------------------------------------------------------------------------
// SetId
// Register or Unregister an I-Button ID
//-------------------------------------------------------------------------------
FUNCTION SetId;
VAR_INPUT
   index  : INT;
   ID     : STRING;
END_VAR;
VAR
   oldIDs : STRING;
   newIDs : STRING;
   i      : INT;
END_VAR;
   // Verify index
   IF index > 0 AND index < 6 THEN
      // Read Registered IDs
      oldIDs := LoadStringF(index:=3);
      // Insert IDs before new
      IF index > 1 THEN
         FOR i := 1 TO index - 1 BY 1 DO
            newIDs :=
strConcat(str1:=newIDs,str2:=strToken(str:=oldIDs,delimiter:=",",index:=i));
            newIDs := strConcat(str1:=newIDs,str2:=",");
         END_FOR;
      END_IF;
      // Insert new ID
      newIDs := strConcat(str1:=newIDs,str2:=ID);
      newIDs := strConcat(str1:=newIDs,str2:=",");
```

```
      // Insert IDs after new
      IF index < 5 THEN
         FOR i := index + 1 TO 5 BY 1 DO
            newIDs :=
strConcat(str1:=newIDs,str2:=strToken(str:=oldIDs,delimiter:=",",index:=i));
            newIDs := strConcat(str1:=newIDs,str2:=",");
         END_FOR;
      END_IF;
      // Save Registered IDs
      SaveStringF(index:=3,str:=newIDs);
   END_IF;
END_FUNCTION;
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// SendPos
// Sends the last position
//---------------------------------------------------------------------------
FUNCTION SendPos;
VAR_INPUT
   phone : STRING;
END_VAR;
VAR
   message : STRING;
END_VAR;
   // Retrieve last valid position from FRAM
   message := LoadStringF(index:=1);
   // Send position
   gsmSendSMS(phonenumber:=phone,message:=message);
END_FUNCTION;
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// OWIBUTTON
// Thread that reads and validates I-Button IDs, for alarm activation/
// deactivation.
//---------------------------------------------------------------------------
THREAD_BLOCK OWIBUTTON;
VAR
   strID  : STRING;
   Valid  : BOOL;
   Count  : INT := 0;
   Blink  : INT := 0;
END_VAR;
   // Init
   OWiButtonEnableLED(enable:=TRUE);
   OWiButtonSetLED(state:=OFF);
   // Do forever
   WHILE TRUE DO
      // Read I-Button ID
      strID := OWiButtonGetID();
      // Is there an I-Button present?
      IF strLen(str:=strID) > 0 THEN
         // Reset timer
         Count := 20;
         Blink := 10;
         // Repetition?
         IF NOT Valid THEN
            // Lookup ID
            IF strFind(str1:=LoadStringF(index:=3),str2:=strID) > 0 THEN
               // Change alarm state
               SetAlarm( state := NOT TestAlarm() );
            END_IF;
            // I-Button has been read
```

```
            OWiButtonSetLED(state:=ON);
            Valid := TRUE;
         END_IF;
      ELSE
         Valid := FALSE;
         // Delay for I-Button removed from reader
         IF Count > 0 THEN
            // Delay
            Count := Count - 1;
            Sleep(delay:=100);
         // Is alarm survailance activated?
         ELSIF TestAlarm() THEN
            // Led := ON for 100 ms.
            IF Blink = 0 THEN
               // Reset timer
               Blink := 10;
               // Turn LED on
               OWiButtonSetLED(state:=ON);
            ELSIF Blink = 9 THEN
               // Turn LED off
               OWiButtonSetLED(state:=OFF);
            END_IF;
            // Delay
            Blink := Blink - 1;
            Sleep(delay:=100);
         ELSE
            OWiButtonSetLED(state:=OFF);
            Sleep(delay:=100);
         END_IF;
      END_IF;
   END_WHILE;
END_THREAD_BLOCK;
//------------------------------------------------------------------------------

//------------------------------------------------------------------------------
// GPS
// Thread that reads valid GPS positions and save them in FRAM
//------------------------------------------------------------------------------
THREAD_BLOCK GPS;
VAR
   gpspos    : gpsFix;
   str       : STRING;
END_VAR;
   // Init
   gpsPower(power:=ON);
   // Do forever
   WHILE TRUE DO
      // Get GPS position
      gpspos();
      // Valid Position?
      IF gpspos.mode > 1 THEN
         // String
         str := strFormat(format:="Time: \1.\2.\3,
",v1:=gpspos.day,v2:=gpspos.month,v3:=gpspos.year);
         str := strConcat(str1:=str,str2:=strFormat(format:="\1:\2:\3
",v1:=gpspos.hour,v2:=gpspos.minute,v3:=gpspos.second));
         IF gpspos.latsouth THEN
            str := strConcat(str1:=str,str2:=strFormat(format:="Lat: S\1*\2.\3
",v1:=gpspos.latdeg,v2:=gpspos.latmin,v3:=gpspos.latdecmin));
         ELSE
            str := strConcat(str1:=str,str2:=strFormat(format:="Lat: N\1*\2.\3
",v1:=gpspos.latdeg,v2:=gpspos.latmin,v3:=gpspos.latdecmin));
         END_IF;
         IF gpspos.lonwest THEN
```

```
            str := strConcat(str1:=str,str2:=strFormat(format:="Long: W\1*\2.
\3 ",v1:=gpspos.londeg,v2:=gpspos.lonmin,v3:=gpspos.londecmin));
         ELSE
            str := strConcat(str1:=str,str2:=strFormat(format:="Long: E\1*\2.
\3 ",v1:=gpspos.londeg,v2:=gpspos.lonmin,v3:=gpspos.londecmin));
         END_IF;
         SaveStringF(index:=1,str:=str);
      END_IF;
      Sleep(delay:=1000);
   END_WHILE;
END_THREAD_BLOCK;
//--------------------------------------------------------------------------

//--------------------------------------------------------------------------
// SMS
// Thread that reads incoming SMS messages and executes commands
//--------------------------------------------------------------------------
THREAD_BLOCK SMS;
VAR
   incoming  : gsmIncomingSMS;
END_VAR;
   // Init
   gsmPower(power:=ON);
   DebugMsg(message:="Ready to receive SMS");
   // Do forever
   WHILE TRUE DO
      incoming();
      // GSM connection status
      led := gsmConnected();
      // Check for incoming SMS
      IF incoming.status > 0 THEN
         DebugMsg(message:=strConcat(str1:="SMS recieved =>
",str2:=strConcat
(str1:=incoming
.phonenumber,str2:=strConcat(str1:=",",str2:=incoming.message))));
         // Set I-Button ID
         IF
strCompare(str1:="OWI",str2:=strLeft(str:=incoming.message,length:=3)) = 0
THEN
            // Insert new ID
            SetId( index :=
strToInt(str:=strMid(str:=incoming.message,start:=4,length:=1)), ID :=
strMid(str:=incoming.message,start:=6) );
            gsmSendSMS
(phonenumber:=incoming
.phonenumber
,message:=strConcat(str1:=strMid(str:=incoming.message,start:=4,length:=1),
                                                                    st
r2:=strMid(str:=incoming.message,start:=6)));
         // Activate Alarm
         ELSIF strCompare(str1:="AON",str2:=incoming.message) = 0 THEN
            // Change alarm state
            SetAlarm(state:=ON);
            gsmSendSMS(phonenumber:=incoming.phonenumber,message:="Alarm=ON");
         // Deactivate Alarm
         ELSIF strCompare(str1:="AOFF",str2:=incoming.message) = 0 THEN
            // Change alarm state
            SetAlarm(state:=OFF);
            gsmSendSMS
(phonenumber:=incoming.phonenumber,message:="Alarm=OFF");
         // Activate Tracking
         ELSIF strCompare(str1:="TON",str2:=incoming.message) = 0 THEN
            // synchronize tracking timer
            NextGps := clockNow();
```

```
            // Change tracking state
            SetTracking(state:=ON);
            gsmSendSMS
(phonenumber:=incoming.phonenumber,message:="Tracking=ON");
         // Deactivate Tracking
         ELSIF strCompare(str1:="TOFF",str2:=incoming.message) = 0 THEN
            // Change tracking state
            SetTracking(state:=OFF);
            gsmSendSMS
(phonenumber:=incoming.phonenumber,message:="Tracking=OFF");
         // Retrieve Position
         ELSIF strCompare(str1:="POS",str2:=incoming.message) = 0 THEN
            // Send last position
            SendPos(phone:=incoming.phonenumber);
         // Set reply phone number
         ELSIF
strCompare(str1:="PHONE",str2:=strLeft(str:=incoming.message,length:=5)) = 0
THEN
            SaveStringF(index:=4,str:=strMid(str:=incoming.message,start:=6));
            gsmSendSMS
(phonenumber:=incoming
.phonenumber
,message:=strConcat
(str1:="Phone=",str2:=strMid(str:=incoming.message,start:=6)));
         END_IF;
      END_IF;
      Sleep(delay:=2000);
   END_WHILE;
END_THREAD_BLOCK;
//--------------------------------------------------------------------------------

PROGRAM thread_example;
VAR
   thIbutton  : OWIBUTTON;
   thGps      : GPS;
   thSms      : SMS;

   Valid      : BOOL;
END_VAR;

   // Init MUTEX
   DebugMsg(message:="Initializing Mutex");
   mxStateAl := mxInit();
   mxStateTr := mxInit();
   IF mxStatus(mx:=mxStateAl) = 1 THEN DebugMsg(message:="mxStateAl failed to
init!"); END_IF;
   IF mxStatus(mx:=mxStateTr) = 1 THEN DebugMsg(message:="mxStateTr failed to
init!"); END_IF;

   // Init THREAD
   DebugMsg(message:="Starting Threads");
   thIbutton();
   thGps();
   thSms();
   IF NOT thIbutton._running THEN DebugMsg(message:="thIbutton failed to
start!"); END_IF;
   IF NOT thGps._running THEN DebugMsg(message:="thGps failed to start!");
END_IF;
   IF NOT thSms._running THEN DebugMsg(message:="thSms failed to start!");
END_IF;

BEGIN
   // Are we monitoring Alarm inputs?
   IF TestAlarm() THEN
```

```
      // Test alarm inputs
      IF (in[1] OR in[2] OR in[3] OR in[4]) AND NOT Valid THEN
         gsmSendSMS(phonenumber:=LoadStringF(index:=4),message:="ALARM!");
         Valid := TRUE;
      END_IF;
   ELSE
      Valid := FALSE;
   END_IF;

   // Are we tracking device?
   IF TestTracking() THEN
      // Time for next position?
      IF clockNow() >= NextGps THEN
         // Send last position
         SendPos(phone:=LoadStringF(index:=4));
         // Time for next position
         NextGps := clockNow() + 60; // 1 Min.
      END_IF;
   END_IF;
END;

END_PROGRAM;
```

6.28 Examples - MX2 GPS log Example

```
//--------------------------------------------------------------------------
-
// MX2GPSLog.vpl, created 2006-05-31 13:15
//
// The application takes the GPS position every 15 seconds and saves it to
both
// the datalogger, and to a file. The file media must be ejected with
// dipswitch 1 before it is removed from the RTCU, otherwise the file will be
// corrupted. The status of the media is shown on the LED on the RTCU.
// When the device is no longer moving it goes into a power saving mode and
waits
// for movement.
// This example demonstrates the Filesystem and powermanagement features of
the
// MX2 device, and can only be used with a MX2 device
//--------------------------------------------------------------------------
-
INCLUDE rtcu.inc

VAR_INPUT
   dipEject   : BOOL R_EDGE;
END_VAR;

VAR_OUTPUT
   gw_conn : BOOL;
   gps_pos : BOOL;
END_VAR;


VAR
   log        : logWrite;
   clock      : clockLinsecToTime;
   gps        : gpsFix;
   gps_linsec : DINT;
   gps_pwr    : BOOL;
   gsm_pwr    : BOOL;
   media_open : BOOL;
   fd         : FILE;
   rc         : INT;
   str        : STRING;
   tSleep     : TON;
END_VAR;

PROGRAM MX2GPSLog;

// Use battery if power fails
pmPowerFail(bat:=TRUE);

// Set timer
tSleep.pt := 300000; // 5 min. delay

// Open media
fsMediaOpen(media:=0);
fsStatusLEDEnable(enable:=ON);

// Initialize datalogger system
IF NOT logIsInitialized(key:=4712) THEN
   logInitialize(key:=4712, numlogvalues:=2, numlogrecords:=0);
END_IF;
```

```
BEGIN
   // LED
   gw_conn := gwConnected();

   // Open log file
   IF fsMediaPresent(media:=0) AND NOT media_open THEN
      media_open := TRUE;
   ELSIF NOT fsMediaPresent(media:=0) AND media_open THEN
      media_open := FALSE;
   END_IF;

   // Eject media
   IF dipEject THEN
      fsMediaEject(media:=0);
      media_open := FALSE;
   END_IF;

   // Open log file
   IF media_open AND fsFileStatus(fd:=fd) <> 0 THEN
      IF fsFileExists(name:="\datalog.txt") THEN
         DebugMsg(message:="Open File");
         fd := fsFileOpen(name:="\datalog.txt");
      ELSE
         DebugMsg(message:="File Create");
         fd := fsFileCreate(name:="\datalog.txt");
      END_IF;
   END_IF;

   // Start GPS and/or GSM
   IF NOT gps_pwr THEN
      DebugMsg(message:="Start GPS");
      gpsPower(power:=ON);
      gps_pwr := ON;
   END_IF;
   IF NOT gsm_pwr THEN
      DebugMsg(message:="Start GSM");
      gsmPower(power:=ON);
      gprsOpen();
      gsm_pwr  := ON;
   END_IF;

   // Get GPS position
   gps();
   // GPS position valid?
   IF gps.mode > 1 THEN
      // Toggle led
      gps_pos   := NOT gps_pos;
      // log position
      IF clockNow() > gps_linsec THEN
         // Save to datalog
         log(value[1]:=gps.latitude,value[2]:=gps.longitude);
         // Save to file
         IF fsFileStatus(fd:=fd) = 0 THEN
            clock(linsec:=gps.linsec);
            str := strFormat(format:="\1.\2.\3,
",v1:=clock.year,v2:=clock.month,v3:=clock.day) +
                   strFormat(format:="\1:\2:\3,
",v1:=clock.hour,v2:=clock.minute,v3:=clock.second);
            IF gps.latsouth THEN str := str + "-"; END_IF;
            str := str + strFormat(format:="\1*\2.\3,
",v1:=gps.latdeg,v2:=gps.latmin,v3:=gps.latdecmin);
            IF gps.lonwest THEN str := str + "-"; END_IF;
            str := str + strFormat(format:="\1*\2.\3,
```

```
",v1:=gps.londeg,v2:=gps.lonmin,v3:=gps.londecmin);
            fsFileWriteStringNL(fd:=fd,str:=str);
         END_IF;
         // Set time for next logging
         gps_linsec := clockNow() + 15;
      END_IF;
   END_IF;

   // Has there been any movement?
   IF pmVibration() THEN
      tSleep(trig:=FALSE);
   ELSE
      tSleep(trig:=TRUE);
   END_IF;

   // No movement detected in 5 minutes
   IF tSleep.q THEN
      IF NOT batIsCharging() THEN
         // Debug to file
         IF fsFileStatus(fd:=fd) = 0 THEN
            fsFileWriteStringNL(fd:=fd,str:="--- Enter pmWaitEvent ---");
         END_IF;
         // Stop GPS and/or GSM
         IF gps_pwr THEN
            DebugMsg(message:="Stop GPS");
            gpsPower(power:=OFF);
            gps_pwr := OFF;
         END_IF;
         IF gsm_pwr THEN
            DebugMsg(message:="Stop GSM");
            gprsClose();
            gsmPower(power:=OFF);
            gsm_pwr := OFF;
         END_IF;
         // Wait for movement (vibration)
         rc := pmWaitEvent(vibration:=ON);
         DebugMsg(message:="pmWaitEvent - Movement");
         // Debug to file
         IF fsFileStatus(fd:=fd) = 0 THEN
            str := strFormat(format:="--- Exit pmWaitEvent (rc=\1)
---",v1:=rc);
            fsFileWriteStringNL(fd:=fd,str:=str);
         END_IF;
         // Wait for 5 minutes before Power down again
         tSleep(trig:=FALSE);
      END_IF;
   END_IF;

END;

END_PROGRAM;
```

6.29 Examples - Navigation Example

```
//--------------------------------------------------------------------------
-
// example.vpl, created 2009-10-21 12:00
//
// This application program demonstrates use of the new navigation system
//
// Commands:
//
// stopset <ID> <latitude> <longitude> <text>
//    -ID          : ID of destination
//    -latitude    : The latitude of the destination position.
//                   This must be in semicircles format.
//    -longitude   : The longitude of the destination position.
//                   This must be in semicircles format.
//    -text        : The text to describe the destination
//
// stopset 1 582902091 27369314 Eiffel Tower
// stopset 2 582920384 27860319 Louvre
// stopset 3 666381800 117509175 Holmboes Allé 14, Logic IO Aps
//
// stopdel <ID>
//    -ID          : ID of destination
//
// etaset <delay>
//    -delay       : The delay between ETA events in seconds.
//
// userreq
//
// userset <Status ID> <Driver ID>
//    -status ID   : The new status of user, if not exists then shown as 'not
set'
//    -Driver ID   : The new user ID.
//
// userdef <Status ID> <text>
//    -status ID   : The user status to define
//    -text        : The text describing the status.
//
// message <ID> <type> <text>
//    -ID          : The ID of the message
//    -type        : 1 = Normal, 2 = popup
//    -text        : The text of the message
//
// clear
//
// uitext <text>
//    -text        : The text of the UI text, empty deletes
//
// quicktxt <ID> <text>
//    -ID          : The ID of the quick text
//    -text        : The text of the quick message, empty deletes
//
//--------------------------------------------------------------------------
-
INCLUDE rtcu.inc

VAR
   navMsgAck     : navMessageReplyReceive;
   navMsg        : navMessageReceive;
   navMsgStatus  : navMessageStatusReceive;
   navETA        : navETAReceive;
```

```
   navStop       : navStopReceive;
   navUserID     : navUserIDReceive;
   navUserStatus : navUserStatusReceive;

   clock         : clockLinsecToTime;
   commands      : ARRAY[1..10] OF STRING :=
                           "stopset",
                           "stopdel",
                           "etaset",
                           "userreq",
                           "userset",
                           "message",
                           "userdef",
                           "clear",
                           "uitext",
                           "quicktxt";

   // phone number to receive notification on navigation events
   dst           : STRING := "9999";
END_VAR;

//-------------------------------------------------------------------------------
-
// FUNCTION    : command_parser
//    extract command and parameters from input message
// INPUT :
//    message  : message to pass for command
// OUTPUT :
//    cmd      : number identifying the command
//    param[]  : parameters extracted from message
//    no_param : max number of entries in param
//-------------------------------------------------------------------------------
-
FUNCTION_BLOCK command_parser;
VAR_INPUT
   message : STRING;
END_VAR;
VAR_OUTPUT
   cmd     : INT;
   param   : ARRAY[1..5] OF STRING;
   no_param: SINT := 5; // max number of entries in param
END_VAR;
VAR
   temp    : STRING;
   i       : INT;
END_VAR;
   // Clear outputs
   FOR i := 1 TO no_param DO
      param[i] := "";
   END_FOR;
   cmd      := 0;

   // Extract command word
   temp := strToken(str := message, delimiter := " ", index := 1);

   // Determine command
   FOR i := 1 TO 10 DO
      IF strCompare(str1 := temp, str2 := commands[i]) = 0 THEN
         cmd := i;
         EXIT;
      END_IF;
   END_FOR;

   // not a valid command string
```

```
   IF cmd = 0 THEN
      RETURN;
   END_IF;

   DebugFmt(message := "SMS received (command  ): '\1' (" + commands[cmd] +
")", v1 := cmd);

   // Unpack parameters
   CASE cmd OF
      //stopset <ID> <latitude> <longitude> <text>
      1: temp     := strRemoveSpaces(str := message);
         temp     := strMid(str := temp, start := strLen(str := commands[cmd])
+ 1);
         temp     := strRemoveSpaces(str := temp);
         param[1] := strToken(str := temp, delimiter := " ", index := 1);
         temp     := strMid(str := temp, start := strLen(str := param[1]) +
1);
         temp     := strRemoveSpaces(str := temp);
         param[2] := strToken(str := temp, delimiter := " ", index := 1);
         temp     := strMid(str := temp, start := strLen(str := param[2]) +
1);
         temp     := strRemoveSpaces(str := temp);
         param[3] := strToken(str := temp, delimiter := " ", index := 1);
         temp     := strMid(str := temp, start := strLen(str := param[3]) +
1);
         param[4] := strRemoveSpaces(str := temp);

      // stopdel <ID>
      2: param[1] := strToken(str := message, delimiter := " ", index := 2);

      // etaset
      3: param[1] := strToken(str := message, delimiter := " ", index := 2);

      // userset <user status id> <text>
      5: temp     := strRemoveSpaces(str := message);
         temp     := strMid(str := temp, start := strLen(str := commands[cmd])
+ 1);
         temp     := strRemoveSpaces(str := temp);
         param[1] := strToken(str := temp, delimiter := " ", index := 1);
         temp     := strMid(str := temp, start := strLen(str := param[1]) +
1);
         param[2] := strRemoveSpaces(str := temp);

      // message <ID> <type> <text>
      6: temp     := strRemoveSpaces(str := message);
         temp     := strMid(str := temp, start := strLen(str := commands[cmd])
+ 1);
         FOR i := 1 TO 2 DO
            temp     := strRemoveSpaces(str := temp);
            param[i] := strToken(str := temp, delimiter := " ", index := 1);
            temp     := strMid(str := temp, start := strLen(str := param[i]) +
1);
         END_FOR;
         param[3] := strRemoveSpaces(str := temp);

      //userdef <status id> <text>
      7: temp     := strRemoveSpaces(str := message);
         temp     := strMid(str := temp, start := strLen(str := commands[cmd])
+ 1);
         temp     := strRemoveSpaces(str := temp);
         param[1] := strToken(str := temp, delimiter := " ", index := 1);
         temp     := strMid(str := temp, start := strLen(str := param[1]) +
1);
         param[2] := strRemoveSpaces(str := temp);
```

```
      // uitext <text>
      9: temp     := strRemoveSpaces(str := message);
         temp     := strMid(str := temp, start := strLen(str := commands[cmd])
+ 1);
         param[1] := strRemoveSpaces(str := temp);

      // quicktxt <ID> <text>
      10:temp     := strRemoveSpaces(str := message);
         temp     := strMid(str := temp, start := strLen(str := commands[cmd])
+ 1);
         temp     := strRemoveSpaces(str := temp);
         param[1] := strToken(str := temp, delimiter := " ", index := 1);
         temp     := strMid(str := temp, start := strLen(str := param[1]) +
1);
         param[2] := strRemoveSpaces(str := temp);
   END_CASE;

   FOR i := 1 TO no_param DO
      DebugFmt(message := "SMS received (argument \1): '" + param[i] + "'", v1
:= i);
   END_FOR;
END_FUNCTION_BLOCK;

//---------------------------------------------------------------------------
-
// FUNCTION    : TimeString
//    extract command and parameters from input message
// INPUT :
//    linsec   : time in Linux seconds
// RETURN :
//    time     : a string with the time expressed in ASCII
//---------------------------------------------------------------------------
-
FUNCTION TimeString : STRING;
VAR_INPUT
   linsec : DINT;
END_VAR;
   clock(linsec := linsec);
   TimeString := strFormat(format := "\1.\2.\3, ", v1 := clock.day, v2 :=
clock.month, v3 := clock.year);
   TimeString := TimeString + strFormat(format := "\1:\2:\3", v1 :=
clock.hour, v2 := clock.minute, v3 := clock.second);
END_FUNCTION;

//---------------------------------------------------------------------------
-
// THREAD    : navMonitor
//    monitor events from the navigation device
//---------------------------------------------------------------------------
-
THREAD_BLOCK navMonitor;
VAR
   event : INT := 0;
   reply : STRING;
END_VAR;

WHILE event <> -1 DO
   event := navWaitEvent(timeout := -1);
   CASE event OF
      1:    // A reply for a text message is received
            navMsgAck();
            DebugMsg(message := "MON: message reply received");
            DebugFmt(message := "MON: -ID=\1", v1 := navMsgAck.ID);
```

```
            DebugMsg(message := "MON: -time=" + TimeString(linsec :=
navMsgAck.time));
            DebugFmt(message := "MON: -reply=\1", v1 := navMsgAck.reply);
            // Reply to server: reply <ID> <timestamp> <reply>
            reply := strFormat(format := "reply \1 \4 \2", v1 := navMsgAck.ID,
v2 := navMsgAck.reply, v4 := navMsgAck.time);
            gsmSendSMS(phonenumber := dst, message := reply);

      2:    // A text message is received
            navMsg();
            DebugMsg(message := "MON: message received");
            DebugFmt(message := "MON: -time=" + TimeString(linsec :=
navMsg.time));
            DebugMsg(message := "MON: -text=" + navMsg.message);
            // Reply to server: message <text>
            gsmSendSMS(phonenumber := dst, message := "message " +
navMsg.message);

      3:    // The status for a text message is received
            navMsgStatus();
            DebugMsg(message := "MON: message status received");
            DebugFmt(message := "MON: -ID=\1", v1 := navMsgStatus.ID);
            CASE navMsgStatus.status OF
               1: DebugMsg(message := "MON: -status=Unread");
               2: DebugMsg(message := "MON: -status=Read");
               3: DebugMsg(message := "MON: -status=Deleted/Not found");
            END_CASE;

      4:    // An ETA update is received
            navETA();
            DebugMsg(message := "MON: ETA received");
            DebugFmt(message := "MON: -stop=\1", v1 := navETA.id);
            DebugFmt(message := "MON: -time=" + TimeString(linsec :=
navETA.time));
            DebugFmt(message := "MON: -distance=\4", v4 := navETA.distance);
            DebugFmt(message := "MON: -latitude=\4", v4 := navETA.latitude);
            DebugFmt(message := "MON: -longitude=\4", v4 := navETA.longitude);
            // Reply to server: ETA <stop> <time> <distance>
            reply := strFormat(format := "ETA \1 \4 ", v1 := navETA.id, v4 :=
navETA.time);
            gsmSendSMS(phonenumber := dst, message := reply + dintToStr(v :=
navETA.distance));

      5:    // The status of a destination is received
            navStop();
            DebugMsg(message := "MON: stop status received");
            DebugFmt(message := "MON: -ID=\1", v1 := navStop.ID);
            DebugFmt(message := "MON: -index=\1", v1 := navStop.index);
            CASE navStop.status OF
               1: DebugMsg(message := "MON: -status=Active");
               2: DebugMsg(message := "MON: -status=Done");
               3: DebugMsg(message := "MON: -status=Inactive - Unread");
               4: DebugMsg(message := "MON: -status=Inactive - Read");
               5: DebugMsg(message := "MON: -status=Deleted/Not found");
            END_CASE;
            // Reply to server: ETA <stop> <time> <distance>
            reply := strFormat(format := "stop \1 \2 \3", v1 := navStop.ID, v2
:= navStop.index, v3 := navStop.status);
            gsmSendSMS(phonenumber := dst, message := reply);

      6:    // An user ID is received
            navUserID();
            DebugMsg(message := "MON: user ID received");
            DebugFmt(message := "MON: -time=" + TimeString(linsec :=
```

```
navUserID.time));
            DebugMsg(message := "MON: -user=" + navUserID.user);
            // Reply to server: user <ID>
            gsmSendSMS(phonenumber := dst, message := "user " +
navUserID.user);

      7:    // The status of an user is received
            navUserStatus();
            DebugMsg(message := "MON: user status received");
            DebugFmt(message := "MON: -time=" + TimeString(linsec :=
navUserStatus.time));
            DebugFmt(message := "MON: -status=\1", v1 :=
navUserStatus.status);
            // Reply to server: status <status>
            reply := strFormat(format := "status \1", v1 :=
navUserStatus.status);
            gsmSendSMS(phonenumber := dst, message := reply);

      129:  // A request to refresh the Quick messages
            DebugMsg(message := "MON: refresh of quick messages received,
sending defaults");
            navMessageQuickDefine(id := 1, text := "I have arrived at the
destination");
            navMessageQuickDefine(id := 2, text := "There are no persons to
receive the packages at destination");
            navMessageQuickDefine(id := 3, text := "Armed robbery in progress,
send help");
            navMessageQuickDefine(id := 4, text := "Packages delivered");
            navMessageQuickDefine(id := 5, text := "Finish with my route.");

      130:  // A request to refresh the message replies.
            DebugMsg(message := "MON: refresh of message replies received,
sending defaults");
            navMessageReplyDefine(id := 1, text := "Accept");
            navMessageReplyDefine(id := 2, text := "Reject");
            navMessageReplyDefine(id := 3, text := "Yes");
            navMessageReplyDefine(id := 4, text := "No");

      131:  // A request to refresh the user status list.
            DebugMsg(message := "MON: refresh of drivers status, sending
defaults");
            navUserStatusDefine(id := 1, text := "Awaiting new destination");
            navUserStatusDefine(id := 2, text := "On the way to a
destination");
            navUserStatusDefine(id := 3, text := "On the way to the office");
            navUserStatusDefine(id := 4, text := "At destination delivering
packages");

      132:  // Connection to navigation device established
            DebugMsg(message := "MON: navigation device present");
            DebugFmt(message := "MON: -serial=" + navDeviceSerial());

      133:  // Connection to navigation device lost
            DebugMsg(message := "MON: navigation device not present");
   ELSE
      DebugFmt(message := "MON: unhandled event form device '\1'", v1 :=
event);
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;

//---------------------------------------------------------------------------
-
// PROGRAM  : example
```

```
//    main application handling initialization, startup of navigation monitor
//    thread. Also handles incoming GSM connections.
//-------------------------------------------------------------------------------
-
PROGRAM example;
VAR
   PDU    : gsmIncomingPDU;
   SMS    : gsmIncomingSMS;
   pdu_rx : ARRAY[1..140] OF SINT;

   navMon : navMonitor;
   rc     : INT;
   cmd    : command_parser;

   id     : INT; // Message identifier
END_VAR;

   // Initializing
   rc := navOpen();
   IF rc = 0 THEN
      DebugMsg(message := "navigation open");
   ELSE
      DebugFmt(message := "navigation error (navOpen=\1)", v1 := rc);
   END_IF;
   navMon();
   gsmPower(power := ON);
   rc := gprsOpen();
   IF rc = 0 THEN
      DebugMsg(message := "GPRS initialized");
   ELSE
      DebugFmt(message := "GPRS error (gprsOpen=\1)", v1 := rc);
   END_IF;
   PDU.message := ADDR(pdu_rx);

   DebugMsg(message := "Initialization finish, ready for communication.");

BEGIN
   // Flush PDU messages
   PDU();

   // Receive SMS commands
   SMS();
   IF SMS.status > 0 THEN
      // Parse command
      cmd(message := SMS.message);
      CASE cmd.cmd OF
         1: // Set destination
            rc := navStopSet(ID := strToInt(str := cmd.param[1]),
                             latitude := strToDint(str := cmd.param[2]),
                             longitude := strToDint(str := cmd.param[3]),
                             text := cmd.param[4]);
            IF rc = 0 THEN
               DebugMsg(message := "Destination set");
            ELSE
               DebugFmt(message := "Destination error (navStopSet=\1)", v1 :=
rc);
            END_IF;

         2: // Delete destination
            rc := navStopDelete(ID := strToInt(str := cmd.param[1]));
            IF rc = 0 THEN
               DebugMsg(message := "Destination removed");
            ELSE
               DebugFmt(message := "Destination error (navStopDelete=\1)", v1
```

```
:= rc);
            END_IF;

         3: // Set ETA delay
            rc := navETAAutoSet(freq := strToInt(str := cmd.param[1]));
            IF rc = 0 THEN
               DebugMsg(message := "ETA delay set");
            ELSE
               DebugFmt(message := "ETA error (navETAAutoSet=\1)", v1 := rc);
            END_IF;

         4: // Request user info
            navUserIDRequest();
            navUserStatusRequest();

         5: // Set user info
            rc := navUserIDSet(user := cmd.param[2]);
            IF rc = 0 THEN
               rc := navUserStatusSet(id := strToInt(str := cmd.param[1]));
               IF rc = 0 THEN
                  DebugMsg(message := "User info set");
               ELSE
                  DebugFmt(message := "User info error (navUserStatusSet=\1)",
v1 := rc);
               END_IF;
            ELSE
               DebugFmt(message := "User info error (navUserIDSet=\1)", v1 :=
rc);
            END_IF;

         6: // Send message
            rc := navMessageSend(ID := strToInt(str := cmd.param[1]),
                                 text := cmd.param[3],
                                 type := strToSint(str := cmd.param[2]) - 1);
            IF rc = 0 THEN
               DebugMsg(message := "Message sent");
            ELSE
               DebugFmt(message := "Message error (navMessageSend=\1)", v1 :=
rc);
            END_IF;

         7: // Define user status
            rc := navUserStatusDefine(id := strToInt(str := cmd.param[1]),
text := cmd.param[2]);
            IF rc = 0 THEN
               DebugMsg(message := "User status defined");
            ELSE
               DebugFmt(message := "User status error
(navUserStatusDefine=\1)", v1 := rc);
            END_IF;

         8: // Clear data in Garmin
            navDeleteData(type := 0);
            navDeleteData(type := 1);
            navDeleteData(type := 2);
            navDeleteData(type := 3);
            navDeleteData(type := 4);
            navDeleteData(type := 6);

         9: // set UI text
            // Customize the text of the user interface in the navigation
device
            rc := navSetUIText(id := 0, text := cmd.param[1]);
            IF rc = 0 THEN
```

```
                DebugMsg(message := "The user interface is set");
            ELSE
                DebugFmt(message := "User interface text error
(navSetUIText=\1)", v1 := rc);
            END_IF;

        10: // set quick message
            id := strToInt(str := cmd.param[1]);
            IF strLen(str := cmd.param[2]) > 0 THEN
                // Define a quick message
                rc := navMessageQuickDefine(id := id, text := cmd.param[2]);
                IF rc = 0 THEN
                    DebugFmt(message := "The quick message \1 has been
set/updated.", v1 := id);
                ELSE
                    DebugFmt(message := "Updating/Setting quick message \1
failed (navMessageQuickDefine=\2)", v1 := id, v2 := rc);
                END_IF;
            ELSE
                // Delete a quick message
                rc := navMessageQuickDelete(id := id);
                IF rc = 0 THEN
                    DebugFmt(message := "The quick message \1 has been
deleted.", v1 := id);
                ELSE
                    DebugFmt(message := "Deletion of quick message \1 failed
(navMessageQuickDelete=\2)", v1 := id, v2 := rc);
                END_IF;
            END_IF;
        ELSE
            // Unknown command
            DebugMsg(message := "Unknown command=" + SMS.message);
        END_CASE;
    END_IF;
END;
END_PROGRAM;
```

6.30 Examples - NMP Example

```
//-------------------------------------------------------------------------
-
// example.vpl, created 2012-03-30 10:01
//
// This application demonstrates the use of the NMP.
//
// It creates two NMP buttons, one red with the text "Panic!" and a green
button
// with the text "Enter vehicle".
// The panic button will start flashing when pressed and print the text
// "!! ALERT !!" to the debug window.
// The text of the enter vehicle button will alternate between "Exit vehicle"
and
// "Enter vehicle" when pressed, and a message is printed to the debug window.
//
// The MDT is enabled, and by pressing the MDT button in the menu, the MDT
will be
// shown, with the text "Welcome!" on the display.
//
//-------------------------------------------------------------------------
-
INCLUDE rtcu.inc

VAR
   navMsgAck     : navMessageReplyReceive;
   navMsg        : navMessageReceive;
   navMsgStatus  : navMessageStatusReceive;
   navETA        : navETAReceive;
   navStop       : navStopReceive;
   navUserID     : navUserIDReceive;
   navUserStatus : navUserStatusReceive;
   nmpButtonRec  : nmpButtonPressedReceive;
   updProg       : nmpUpdateProgressReceive;
   gps           : gpsFix;
   mdtInfo       : mdtProfile;
   // flag to keep track of the location of the driver, according to the
   // enter/exit vehicle button
   driverInVehicle: BOOL;

END_VAR;


//-------------------------------------------------------------------------
-
// FUNCTION   : setupNMP
//    Handles the creation of buttons, etc. on the NMP when it is connected.
//-------------------------------------------------------------------------
-
FUNCTION setupNMP : INT;
VAR
   rc:INT;
END_VAR;
   //Add NMP buttons
   rc := nmpButtonsDefine(count:=2);
   IF rc <> 0  THEN
      DebugFmt(message:="Failed to define NMP buttons (nmpButtonsDefine=\1)",
v1:=rc);
      setupNMP := rc;
      RETURN;
   END_IF;
```

```
   rc := nmpButtonCreate(id := 1, weight:= 2, text := "Panic!", visible :=
TRUE,
enable := TRUE, color := nmpRGBToDint(r:= 16#FF, g := 16#00, b := 16#00));
   IF(rc <> 0 ) THEN
      DebugFmt(message:="Failed to create panic button (nmpButtonCreate=\1)",
v1:=rc);
      setupNMP := rc;
      RETURN;
   END_IF;

   IF driverInVehicle THEN
      rc := nmpButtonCreate(id := 2, weight:= 3, text := "Exit vehicle",
visible := TRUE,
enable := TRUE, color := nmpRGBToDint(r:= 16#66, g := 16#FF, b := 16#66));
   ELSE
      rc := nmpButtonCreate(id := 2, weight:= 3, text := "Enter vehicle",
visible := TRUE,
enable := TRUE, color := nmpRGBToDint(r:= 16#66, g := 16#FF, b := 16#66));

   END_IF;
   IF(rc <> 0 ) THEN
      DebugFmt(message:="Failed to create enter/exit button
(nmpButtonCreate=\1)", v1:=rc);
      setupNMP := rc;
      RETURN;
   END_IF;

   // Turn MDT on
   rc := mdtPower(power := ON);
   IF(rc = 0) THEN
      DebugMsg(message:="mdt powered on");
      mdtInfo();
      IF mdtInfo.type > 0 THEN
         DebugMsg(message:="MDT connected!");
         DebugFmt(message:="  Type=\1",v1:=mdtInfo.type);
         DebugFmt(message:="  Ver= \1",v1:=mdtInfo.version);
         DebugFmt(message:="  MaxX=\1",v1:=mdtInfo.pos_max_x);
         DebugFmt(message:="  MaxY=\1",v1:=mdtInfo.pos_max_y);
      END_IF;

   ELSE
      DebugFmt(message:="Failed to power MDT on (mdtPower=\1)", v1:=rc);
   END_IF;


   setupNMP:= rc;
END_FUNCTION;

//-------------------------------------------------------------------------------
-
// THREAD    : navMonitor
//    monitor events from the navigation device
//-------------------------------------------------------------------------------
-
THREAD_BLOCK navMonitor;
VAR
   event : INT := 0;
   rc    : INT;
END_VAR;

WHILE event <> -1 DO
   event := navWaitEvent(timeout := -1);
   CASE event OF
      1:    // A reply for a text message is received
```

```
            navMsgAck();
            DebugMsg(message := "MON: message reply received");
            DebugFmt(message := "MON: -ID=\1", v1 := navMsgAck.ID);
            DebugFmt(message := "MON: -reply=\1", v1 := navMsgAck.reply);

        2:  // A text message is received
            navMsg();
            DebugMsg(message := "MON: message received");
            DebugMsg(message := "MON: -text=" + navMsg.message);

        3:  // The status for a text message is received
            navMsgStatus();
            DebugMsg(message := "MON: message status received");
            DebugFmt(message := "MON: -ID=\1", v1 := navMsgStatus.ID);
            CASE navMsgStatus.status OF
               1: DebugMsg(message := "MON: -status=Unread");
               2: DebugMsg(message := "MON: -status=Read");
               3: DebugMsg(message := "MON: -status=Deleted/Not found");
            END_CASE;

        4:  // An ETA update is received
            navETA();
            DebugMsg(message := "MON: ETA received");
            DebugFmt(message := "MON: -stop=\1", v1 := navETA.id);
            DebugFmt(message := "MON: -distance=\4", v4 := navETA.distance);
            DebugFmt(message := "MON: -latitude=\4", v4 := navETA.latitude);
            DebugFmt(message := "MON: -longitude=\4", v4 := navETA.longitude);

        5:  // The status of a destination is received
            navStop();
            DebugMsg(message := "MON: stop status received");
            DebugFmt(message := "MON: -ID=\1", v1 := navStop.ID);
            DebugFmt(message := "MON: -index=\1", v1 := navStop.index);
            CASE navStop.status OF
               1: DebugMsg(message := "MON: -status=Active");
               2: DebugMsg(message := "MON: -status=Done");
               3: DebugMsg(message := "MON: -status=Inactive - Unread");
               4: DebugMsg(message := "MON: -status=Inactive - Read");
               5: DebugMsg(message := "MON: -status=Deleted/Not found");
            END_CASE;

        6:  // An user ID is received
            navUserID();
            DebugMsg(message := "MON: user ID received");
            DebugMsg(message := "MON: -user=" + navUserID.user);

        7:  // The status of an user is received
            navUserStatus();
            DebugMsg(message := "MON: user status received");
            DebugFmt(message := "MON: -status=\1", v1 :=
navUserStatus.status);

        10: // NMP update status received
            updProg();
            IF updProg.ready THEN
               DebugMsg(message := "*** Update progress received ***");
               DebugFmt(message := "-status=\1", v1 := updProg.status);
               DebugFmt(message := "-progress=\1", v1 := updProg.progress);
               IF updProg.status = 0 THEN
                  DebugMsg(message:="Update transferred to NMP");
               ELSIF updProg.status = 2 THEN
                  DebugMsg(message:="Update ready to install");
               ELSIF updProg.status < 0 THEN
                  DebugMsg(message:="Failed to transfer update");
```

```
            END_IF;
         END_IF;

   11:   // NMP button pressed
         nmpButtonRec();
         IF nmpButtonRec.ready THEN
            DebugMsg(message := "*** Button Pressed ***");
            DebugFmt(message := "-button=\1", v1 := nmpButtonRec.id);
            CASE nmpButtonRec.id  OF
               1: // Panic!
                  rc:=nmpButtonFlash
(id:=1,flash:=ON,duration:=30,period:=250,
color := nmpRGBToDint(r:=16#FF, g:=16#FF, b:=16#FF));
                  DebugMsg(message:="!! ALERT !!");

               2: // Enter/exit
                  driverInVehicle := NOT driverInVehicle;
                  IF driverInVehicle THEN
                     DebugMsg(message:="Driver entered vehicle");
                     rc := nmpButtonText(id := 2, text := "Exit vehicle");
                  ELSE
                     DebugMsg(message:="Driver exited vehicle");
                     rc := nmpButtonText(id := 2, text := "Enter vehicle");
                  END_IF;
            END_CASE;
         END_IF;

   129:  // A request to refresh the Quick messages
         DebugMsg(message := "MON: refresh of quick messages received,
sending
defaults");
         navMessageQuickDefine(id := 1, text := "I have arrived at the
destination");
         navMessageQuickDefine(id := 2, text := "There are no persons to
receive
the packages at destination");

   130:  // A request to refresh the message replies.
         DebugMsg(message := "MON: refresh of message replies received,
sending
defaults");
         navMessageReplyDefine(id := 1, text := "Yes");
         navMessageReplyDefine(id := 2, text := "No");

   131:  // A request to refresh the user status list.
         DebugMsg(message := "MON: refresh of drivers status, sending
defaults");
         navUserStatusDefine(id := 1, text := "Awaiting new destination");
         navUserStatusDefine(id := 2, text := "On the way to a
destination");

   132:  // Connection to navigation device established
         DebugMsg(message := "MON: navigation device present");
         DebugFmt(message := "MON: -serial=" + navDeviceSerial());

         IF nmpPresent() THEN
            DebugMsg(message := "NMP present");
            IF setupNMP() = 0 THEN
               DebugMsg(message := "NMP set up");
            ELSE
               DebugMsg(message := "Failed to set up NMP");
            END_IF;
         ELSE
            DebugMsg(message := "Not an NMP");
```

```
            END_IF;

      133:  // Connection to navigation device lost
            DebugMsg(message := "MON: navigation device not present");
   ELSE
      DebugFmt(message := "MON: unhandled event form device '\1'", v1 :=
event);
   END_CASE;
END_WHILE;
END_THREAD_BLOCK;


//-------------------------------------------------------------------------
-
// THREAD    : gpsRefresh
//    calls gpsFix repeatedly to send gps data to the NMP.
//-------------------------------------------------------------------------
-
THREAD_BLOCK gpsRefresh;
   WHILE TRUE DO
      gps();
      Sleep(delay:=100);
   END_WHILE;
END_THREAD_BLOCK;

//-------------------------------------------------------------------------
-
// THREAD    : mdtMonitor
//    Monitors the MDT for pressed buttons.
//-------------------------------------------------------------------------
-
THREAD_BLOCK mdtMonitor;
VAR
   key:INT;
   rc:INT;
END_VAR;
   WHILE TRUE DO
      key := mdtGetKey(timeout := 10000);
      CASE key OF
         127:  // PowerUP key
               rc:=mdtStandby(enable :=OFF);
               DebugFmt(message:="mdtStandby(OFF) = \1", v1:=rc);


               rc:=mdtGotoXY(x:=10,y:=5);
               DebugFmt(message:="mdtGotoXY() = \1", v1:=rc);
               rc:=mdtWrite(message:="Welcome!");
               DebugFmt(message:="mdtWrite() = \1", v1:=rc);

         126: // Power key
               rc:=mdtStandby(enable :=ON);
               DebugFmt(message:="mdtStandby(ON) = \1", v1:=rc);


      END_CASE;
   END_WHILE;
END_THREAD_BLOCK;


PROGRAM example;
// These are the local variables of the program block
VAR
   navMon : navMonitor;
   gpsRefr: gpsRefresh;
```

```
   mdtMon : mdtMonitor;
   rc     : INT;

END_VAR;
   // Initializing
   gpsPower(power:=TRUE);

   rc := navOpen(port := 1);
   IF rc = 0 THEN
      DebugMsg(message := "navigation open");
   ELSE
      DebugFmt(message := "navigation error (navOpen=\1)", v1 := rc);
   END_IF;
   // Power on NMP
   rc := nmpPower( power:= ON);
   IF rc = 0 THEN
      DebugMsg(message := "NMP powered on");
   ELSE
      DebugFmt(message := "failed to power on NMP (nmpPower=\1)", v1 := rc);
   END_IF;
   // Monitor for events from NMP
   navMon();
   // Start sending GPS data to NMP
   gpsRefr();
   // Open MDT
   rc := mdtOpen(port := 1);
   // Monitor MDT keys
   mdtMon();

BEGIN
   // Code from this point until END will be executed repeatedly
   Sleep(delay:=100);
END;

END_PROGRAM;
```

6.31 Examples - SMTP Example

```
//-------------------------------------------------------------------------
-
// smtp_example.vpl, created 2009-11-11 09:16
//
// An example of how SMTP can be used to monitor a liquid container
//
// Illustration:
//
//             |o|   <- supply flow monitor
//             +-+   <- top valve
//        |    | |    |
//        |           -| <- High level
//        |            |
//        |           -| <- Low level
//        +----+ +----+
//             +-+   <- bottom valve
//             |o|   <- drain flow monitor
//
// If any 1-Wire temperature sensors are connected, data will be collected and
// sent by email.
//-------------------------------------------------------------------------
-
INCLUDE rtcu.inc

VAR_INPUT
   level_high   : BOOL; | liquid is above high level
   level_low    : BOOL; | liquid is above low level
   flow_supply  : INT;  | amount of liquid running into tank
   flow_drain   : INT;  | amount of liquid running out of tank
END_VAR;

VAR_OUTPUT
   valve_top    : BOOL; | active open
   valve_bottom : BOOL; | active open
END_VAR;

VAR
   // Mail configuration
   receiver     : STRING := "mail@domain.com";

   clock        : clockGet;           // Reads the builtin Clock
   cnv_time     : clockLinsecToTime;  // Converts number to date format
END_VAR;

//-------------------------------------------------------------------------
// THREAD : tb_SendDataFile
//-------------------------------------------------------------------------
THREAD_BLOCK tb_SendDataFile;
VAR_INPUT
   filename     : STRING;
END_VAR;
VAR_OUTPUT
   status       : INT; // mail send status
   mail         : INT;
END_VAR;
VAR
   rc           : INT;
END_VAR;
   status := 1000;
```

```
   // Start a new e-mail
   mail := smtpNew(Receiver := receiver, Subject := "Data collected.");
   // Add text to the e-mail
   smtpAddText(Handle := mail, Message := "Data has been collected from ");

   cnv_time(linsec := fsFileGetCreateTime(name := filename));
   // date
   smtpAddText(Handle := mail, Message := intToStr(v := cnv_time.day));
   smtpAddText(Handle := mail, Message := "-" + intToStr(v :=
cnv_time.month));
   smtpAddText(Handle := mail, Message := "-" + intToStr(v := cnv_time.year));
   // time
   smtpAddText(Handle := mail, Message := " " + intToStr(v := cnv_time.hour));
   smtpAddText(Handle := mail, Message := ":" + intToStr(v :=
cnv_time.minute));
   smtpAddText(Handle := mail, Message := ":" + intToStr(v :=
cnv_time.second));

   smtpAddText(Handle := mail, Message := " to ");

   cnv_time(linsec := clockNow());
   // date
   smtpAddText(Handle := mail, Message := intToStr(v := cnv_time.day));
   smtpAddText(Handle := mail, Message := "-" + intToStr(v :=
cnv_time.month));
   smtpAddText(Handle := mail, Message := "-" + intToStr(v := cnv_time.year));
   // time
   smtpAddText(Handle := mail, Message := " " + intToStr(v := cnv_time.hour));
   smtpAddText(Handle := mail, Message := ":" + intToStr(v :=
cnv_time.minute));
   smtpAddText(Handle := mail, Message := ":" + intToStr(v :=
cnv_time.second));

   // Attach a file to the e-mail
   smtpAddAttachment(Handle := mail, Filename := filename);

   // Send the e-mail
   rc := smtpSendX(Handle := mail);
   IF rc <> 0 THEN
      DebugFmt(message := "Failed to send data file, error code '\1'.", v1 :=
rc);
   ELSE
      // monitor process
      REPEAT
         status := smtpStatus(Handle := mail);
      UNTIL status < 0
      END_REPEAT;
      DebugMsg(message := "data mail sent.");
   END_IF;
   // Delete file
   fsFileDelete(name := filename);
   DebugMsg(message := "file '" + filename + "' deleted.");
END_THREAD_BLOCK;

//-------------------------------------------------------------------------
// SendDataMail instant, used by temperature thread.
//-------------------------------------------------------------------------
VAR
   SendMail : tb_SendDataFile;
END_VAR;

//-------------------------------------------------------------------------
// THREAD : tb_temperature_monitor
//-------------------------------------------------------------------------
```

```
THREAD_BLOCK tb_temperature_monitor;
VAR
   next_check  : DINT := 0;
   fd          : FILE;
   fname       : STRING := "data.csv";
   next_send   : DINT;
   cur_row     : DINT := 0;
   owCount     : INT;
   value       : DINT;
   new_file    : BOOL := TRUE;
   i           : INT;
   str         : STRING;
END_VAR;

   DebugMsg(message:= "Temperature monitor started.");

   WHILE TRUE DO
      // Create new data file, on new collection.
      IF new_file THEN
         new_file := FALSE;

         // Delete any old file
         fsFileDelete(name := fname);

         // Create file
         fd :=fsFileCreate(name := fname);

         // Write header row followed by a new line to file
         fsFileWriteStringNL(fd := fd, str := "$"Time$";$"Average$";$"Values -
>$"");

         next_send := clockNow() + 300; // send each 5th minute
         cur_row := 2;
      END_IF;

      // Collect data
      IF next_check < clockNow() THEN
         next_check := clockNow() + 10; // monitor each 10 second.

         // Search for One-Wire devices
         owCount := owSearch(family := 1);

         // get system clock
         clock();
         // write check time in first column
         fsFileWriteString(fd := fd, str := "$"" + intToStr(v := clock.day));
         fsFileWriteString(fd := fd, str := "-" + intToStr(v := clock.month));
         fsFileWriteString(fd := fd, str := "-" + intToStr(v := clock.year));
         fsFileWriteString(fd := fd, str := " " + intToStr(v := clock.hour));
         fsFileWriteString(fd := fd, str := ":" + intToStr(v :=
clock.minute));
         fsFileWriteString(fd := fd, str := ":" + intToStr(v := clock.second)
+ "$"");

         // generate average cell recognized by Microsoft Excel and OpenOffice
Calc
         str := strFormat(format := ";$"=SUM(C\4:O\4)/\1$"", v1 := owCount, v4
:= cur_row);
         // write sum function
         fsFileWriteString(fd := fd, str := str);

         // Write found temperatures
         FOR i := 1 TO owCount DO
            value := owTempGet(device := SINT(i));
```

```
            // write data to file
            fsFileWriteString(fd := fd, str := ";$"");
            fsFileWriteString(fd := fd, str := dintToStr(v := value / 100));
            fsFileWriteString(fd := fd, str := ",");
            fsFileWriteString(fd := fd, str := dintToStr(v := value MOD 100));
            fsFileWriteString(fd := fd, str := "$"");
         END_FOR;

         // Write newline
         fsFileWriteString(fd := fd, str := "$N");
         cur_row := cur_row + 1;

         // wait 5 second before next check
         Sleep(delay := 5000);
      END_IF;

      // Time to send data
      IF next_send < clockNow() THEN
         fsFileClose(fd := fd);

         IF SendMail._running THEN
            DebugMsg(message := "Waiting for last mail to finish.");
            WHILE SendMail.status >= 0 AND SendMail._running DO
               DebugFmt(message := "Progress : \1", v1 := SendMail.status);
               Sleep(delay := 500);
            END_WHILE;
         END_IF;

         // Rename file
         fsFileRename(name_old := fname, name_new := "temp.csv");
         // send the file as an attachment
         SendMail(filename := "temp.csv");

         new_file := TRUE;
      END_IF;
   END_WHILE;
END_THREAD_BLOCK;

//---------------------------------------------------------------------------
// THREAD_BLOCK : tb_level_monitor
//---------------------------------------------------------------------------
THREAD_BLOCK tb_level_monitor;
VAR
   trig_supply : RF_TRIG;
   trig_drain  : RF_TRIG;
   trig_high   : RF_TRIG;
   trig_low    : RF_TRIG;

   message     : STRING;
   subject     : STRING;
   send_msg    : BOOL;

   rc          : INT;
END_VAR;

   DebugMsg(message := "Level monitor started.");

   // open vales according to current state
   IF NOT level_high THEN
      valve_top := ON;
   END_IF;

   IF level_low THEN
      valve_bottom := ON;
```

```
   END_IF;

   // skip initial level warnings
   trig_high(trig := level_high);
   trig_low(trig := level_low);
   trig_drain(trig := flow_drain > 10);
   trig_supply(trig := flow_supply > 10);

   WHILE TRUE DO
      send_msg := FALSE;

      trig_high(trig := level_high);
      trig_low(trig := level_low);

      trig_drain(trig := flow_drain > 10);
      trig_supply(trig := flow_supply > 10);

      // check high level
      IF trig_high.rq THEN
         subject := "WARNING: Liquid level critical high";
         message := subject + ", closing top valve.";
         valve_top := OFF;
         send_msg := TRUE;
      ELSIF trig_high.fq THEN
         subject := "NOTIFY: Liquid level below critical high";
         message := subject + ", opening top valve.";
         valve_top := ON;
         send_msg := TRUE;
      END_IF;

      // check low level
      IF trig_low.fq THEN
         subject := "WARNING: Liquid level critical low";
         message := subject + ", closing bottom valve.";
         valve_bottom := OFF;
         send_msg := TRUE;
      ELSIF trig_low.rq THEN
         subject := "NOTIFY: Liquid level above critical low";
         message := subject + ", opening bottom valve.";
         valve_bottom := ON;
         send_msg := TRUE;
      END_IF;

      IF send_msg THEN
         DebugMsg(message := message);
         // Send an e-mail
         rc := smtpSend(Receiver := receiver, Subject := subject, Message :=
message);
         IF rc <> 0 THEN
            DebugFmt(message := "Failed to send message, error code '\1'.", v1
:= rc);
         END_IF;
      END_IF;

      // monitor supply and drain pipes
      IF trig_supply.fq AND valve_top THEN
         subject := "WARNING: supply pipe clogged";
         message := subject + ", no action taken.";
         DebugMsg(message := message);
         // Send an e-mail
         rc := smtpSend(Receiver := receiver, Subject := subject, Message :=
message);
         IF rc <> 0 THEN
            DebugFmt(message := "Failed to send message, error code '\1'.", v1
```

```
:= rc);
            END_IF;
         END_IF;
         IF trig_drain.fq AND valve_bottom THEN
            subject := "WARNING: drain pipe clogged";
            message := subject + ", no action taken.";
            DebugMsg(message := message);
            // Send an e-mail
            rc := smtpSend(Receiver := receiver, Subject := subject, Message :=
message);
            IF rc <> 0 THEN
               DebugFmt(message := "Failed to send message, error code '\1'.", v1
:= rc);
            END_IF;
         END_IF;

         // wait 2 seconds before next scan
         Sleep(delay := 2000);
      END_WHILE;
END_THREAD_BLOCK;

//---------------------------------------------------------------------------
// PROGRAM: smtp_example
//---------------------------------------------------------------------------
PROGRAM smtp_example;
VAR
   rc                : INT;
   last_signal_time  : DINT := 0;
   temp_monitor      : tb_temperature_monitor;
   level_monitor     : tb_level_monitor;
END_VAR;

   DebugMsg(message := "Initializing.");
   // Open media
   rc := fsMediaOpen(media := 0);
   DebugFmt(message := "fsMediaOpen = \1", v1 := rc);

   // Controls power to the GSM module
   rc := gsmPower(power := ON);
   DebugFmt(message := "gsmPower = \1", v1 := rc);

   // Open/activate GPRS connection.
   rc := gprsOpen();
   DebugFmt(message := "gprsOpen = \1", v1 := rc);

   // Set SMTP configuration
   smtpSetConfig(
      IP :="mail.domain.com",
      Port := 25,
      Sender := "tank_monitor@domain.com",
      Authenticate := OFF
   );

   // Open the smtp interface
   rc := smtpOpen();
   DebugFmt(message := "smtpOpen = \1", v1 := rc);

   DebugMsg(message := "Starting threads.");
   temp_monitor();
   level_monitor();

   DebugMsg(message := "Ready");
BEGIN
   IF last_signal_time < clockNow() THEN
```

```
      last_signal_time := clockNow() + 10; // check again in 10sec

      // Check if GPRS connection is established
      IF NOT gprsConnected() THEN
         DebugMsg(message := "WARNING: no connection to GPRS, sending mails
not posible.");
      END_IF;

      IF NOT temp_monitor._running THEN
         DebugMsg(message := "ERROR: Temperature monitor not running,
starting.");
         temp_monitor();
      END_IF;

      IF NOT level_monitor._running THEN
         DebugMsg(message := "ERROR: Level monitor not running, starting.");
         level_monitor();
      END_IF;
   END_IF;
END;
END_PROGRAM;
```

6.32 Examples - SNMP - Manager

```
//--------------------------------------------------------------------------
-
// LEDmanager.vpl, created 2020-02-19 15:58
//
// This program demonstrates communication between two RTCU devices, where one
// is the manager (trap listener) and the other a client (publishing agent).
//
// To visualize the communication, the LEDs of both devices are manipulated
// by each other. The manager initiates by setting a published integer
variable
// on the client, which then updates it's LEDs according to the value written,
// and then sends a trap to the manager, that updates it's LEDs accordingly
and
// writes the next value to the other LED on the client, which again responds
// with a trap, which the manager uses to update it's LEDs with and so on.
//--------------------------------------------------------------------------
-
INCLUDE rtcu.inc

// Use LAN1
#DEFINE SNMP_INTERFACE 2

// Exchange this with your real enterprise OID base.
#DEFINE ENTERPRISE_OID "1.3.6.1.4.1.65500."

// Put the actual IP address and port of the client here.
#DEFINE SNMP_CLIENT_IP          "192.168.1.20" // example IP
#DEFINE SNMP_CLIENT_PORT        161

#DEFINE rwusername               "roed"
#DEFINE sysUpTime                "1.3.6.1.2.1.1.3.0"
#DEFINE usmStatsUnknownEngineIDs "1.3.6.1.6.3.15.1.1.4.0"

VAR_OUTPUT
   ledAgreen : BOOL;
   ledAred   : BOOL;
   ledBgreen : BOOL;
   ledBred   : BOOL;
END_VAR;

VAR
  netInfo    : netGetInformation;
  snmpvar    : snmpVariable;
  iface      : SINT := SNMP_INTERFACE;
  ledA       : INT; // 0=off, 1=green, 2=red, 3=yellow
  ledB       : INT; // 0=off, 1=green, 2=red, 3=yellow
END_VAR;

FUNCTION printNetInfo
  netInfo(iface:=iface);
  DebugFmt(message:="Network interface \1:", v1:=iface);
  CASE netInfo.status OF
     0 : DebugMsg(message:="Interface is not present.");
     1 : DebugMsg(message:="Interface is not opened.");
     2 : DebugMsg(message:="Interface is not clientconected.");
     3 : DebugMsg(message:="Interface is clientconected.");
     4 : DebugMsg(message:="Interface link is up and waiting for IP
configuration.");
  ELSE
     DebugFmt(message:="Illegal net status? (\1)", v1:=netInfo.status);
```

```
  END_CASE;
  IF netInfo.status > 1 THEN
    DebugMsg(message:=" Phy addr " + netInfo.phy_addr);
    IF netInfo.dhcp THEN
        DebugMsg(message:=" Dynamic IP address");
    ELSE
        DebugMsg(message:=" Static IP address");
    END_IF;
    DebugMsg(message:=" IP addr " + sockIPToName(ip := netInfo.ip));
    DebugMsg(message:=" Mask    " + sockIPToName(ip := netInfo.subnetMask));
    DebugMsg(message:=" Gateway " + sockIPToName(ip := netInfo.gateway));
    IF netInfo.AutoDNS THEN
        DebugMsg(message:=" Automatic DNS");
    ELSE
        DebugMsg(message:=" Manual DNS");
    END_IF;
    DebugMsg(message:=" DNS1    " + sockIPToName(ip := netInfo.DNS1));
    DebugMsg(message:=" DNS2    " + sockIPToName(ip := netInfo.DNS2));
  END_IF;
END_FUNCTION;

FUNCTION updateLeds;
VAR_INPUT
   ledA : INT;
   ledB : INT;
END_VAR;
   CASE ledA OF
      0:
      ledAgreen := OFF;
      ledAred := OFF;
      1:
      ledAgreen := ON;
      ledAred := OFF;
      2:
      ledAgreen := OFF;
      ledAred := ON;
      3:
      ledAgreen := ON;
      ledAred := ON;
   ELSE
   END_CASE;

   CASE ledB OF
      0:
      ledBgreen := OFF;
      ledBred := OFF;
      1:
      ledBgreen := ON;
      ledBred := OFF;
      2:
      ledBgreen := OFF;
      ledBred := ON;
      3:
      ledBgreen := ON;
      ledBred := ON;
   END_CASE;
   UPDATEIO;
END_FUNCTION;


PROGRAM LEDmanager;
VAR
   deleteuser : snmpUser;
   rwuser     : snmpUser;
```

```
   trapd      : SYSHANDLE;
   clientcon  : SYSHANDLE;
   str        : STRING;
   val        : DINT;
   rc         : INT;
   oid        : STRING;
   vars       : INT;
   ix         : SINT;
   event      : USINT;
   x          : DINT;
END_VAR;
ledA := 0;
ledB := 0;
updateLeds(ledA:=ledA, ledB:=ledB);

rc := netOpen(iface:=iface);
DebugFmt(Message:="netOpen (rc=\1)", v1:=rc);

WHILE NOT netConnected(iface:=iface) DO
   Sleep(Delay:=2000);
END_WHILE;

// Output network information
printNetInfo();

// Preemptively delete persistent users.
deleteuser.username := "";
FOR ix:=1 TO 25 DO
   snmpUserSet(index:=ix, user:=deleteuser);
END_FOR;

// Configure read-write user
rwuser.username := rwusername;
rwuser.password := "deor1234";
rwuser.cryptkey := "a0b3d543";
rwuser.authentication := _SNMP_USM_MD5;
rwuser.encryption := _SNMP_USM_AES;
rwuser.level := _SNMP_SEC_ENC;
rwuser.engineid := "12345678900001030507";

// Set USM users
rc := snmpUserSet(index:=1, user:=rwuser);
IF rc <> 1 THEN
   DebugFmt(message:="Failed to set snmp user (rc=\1)", v1:=rc);
END_IF;

// Start the trap listener
rc := snmpStartListen(
                      handle := trapd,
                      version:= _SNMP_VER_3
                     );
DebugFmt(message:="snmpStartListen (rc=\1)", v1:=rc);

// Register traps to listen for
rc := snmpRegisterTrap(oid:=ENTERPRISE_OID + "100");
DebugFmt(message:="snmpRegisterTrap (rc=\1)", v1:=rc);
rc := snmpRegisterTrap(oid:=ENTERPRISE_OID + "101");
DebugFmt(message:="snmpRegisterTrap (rc=\1)", v1:=rc);

// Setup connection to the client
rc := snmpConnect(
                  host    := SNMP_CLIENT_IP,
                  port    := SNMP_CLIENT_PORT,
                  version := _SNMP_VER_3,
```

```
                    usmuser := ADDR(rwuser),
                    handle  := clientcon
                   );
DebugFmt(message:="snmpConnect (rc=\1)", v1:=rc);

// Start the test by setting a variable on the client
snmpSetInteger(handle:=clientcon, OID:=ENTERPRISE_OID + "1.1.0", v:=1);
IF rc <> 1 THEN
   DebugFmt(message:="Failed to set variable! snmpSetInteger (rc=\1)",
v1:=rc);
END_IF;

BEGIN
// Handle events
   rc := snmpWaitEvent(timeout:=-1, event:=event, oid:=oid, vars:=vars);
   CASE event OF
      _SNMP_EVENT_NOEVENT :
         CASE rc OF
             0: DebugMsg(message:="ERROR: The function is not supported!");
            -2: DebugMsg(message:="ERROR: Invalid input");
            -5: DebugMsg(message:="WARNING: Event buffer overflow.");
         ELSE
            DebugFmt(message:="ERROR: unknown return code from snmpReadEvent
(rc=\1)", v1:=rc);
         END_CASE;
      _SNMP_EVENT_TIMEOUT :
         DebugMsg(message:="snmpWaitEvent: Timeout!");
      _SNMP_EVENT_TRAP :
         DebugMsg(message:="Trap received!");
         DebugMsg(message:="Trap oid is: " + oid);
         DebugFmt(message:="Vars to read : \1", v1:=vars);
      _SNMP_EVENT_SET :
         DebugMsg(message:="Set request received on OID " + oid);
   ELSE
      DebugFmt(message:="ERROR: snmpWaitEvent - unknown event type \1!",
v1:=event);
   END_CASE;
   rc := 0;
   WHILE NOT(rc = -11) DO
      rc := snmpGetNextVar(data:=snmpvar);
      CASE rc OF
       0: DebugMsg(message:="snmpGetNextVar: Function not implemented!");
      -2: DebugMsg(message:="Invalid parameter.");
      -9: DebugFmt(message:="Invalid variable type \1", v1:=snmpvar.type);
      -11:
      1:
         CASE snmpvar.type OF
            1: DebugFmt(message:="TimeTicks received = \4",
v4:=snmpvar.time_value);
            2: DebugFmt(message:="INTEGER received = \4",
v4:=snmpvar.int_value);
            3: DebugFmt(message:="UNSIGNED INTEGER received = \4",
v4:=snmpvar.uint_value);
            4: DebugFmt(message:="STRING received = " + snmpvar.str_value);
            5: DebugFmt(message:="Hex STRING received = " +
snmpvar.hex_value);
            6: DebugFmt(message:="OID received = " + snmpvar.oid_value);
            7: DebugFmt(message:="IP Address received = " + snmpvar.ip_value);
            8: DebugFmt(message:="FLOAT received = " +
floatToStr(v:=snmpvar.float_value) );
            9: DebugFmt(message:="DOUBLE received = " +
doubleToStr(v:=snmpvar.double_value) );
         ELSE
            DebugFmt(message:="Unknown var type received : \1",
```

```
v1:=snmpvar.type);
         END_CASE;
         IF strCompare(str1:=snmpvar.oid, str2:="." + ENTERPRISE_OID +
"100.1") = 0 THEN
            ledA := INT(snmpvar.int_value);
            updateleds(ledA:=ledA, ledB:=ledB);
            snmpGetString(handle:=clientcon, OID:=ENTERPRISE_OID + "2.1.0",
v:=str);
            DebugMsg(message:="Client LED " + str);
            x := (snmpvar.int_value + 1) MOD 4;
            snmpSetInteger(handle:=clientcon, OID:=ENTERPRISE_OID + "1.2.0",
v:=x);
         END_IF;
         IF strCompare(str1:=snmpvar.oid, str2:="." + ENTERPRISE_OID +
"100.2") = 0 THEN
            ledB := INT(snmpvar.int_value);
            updateleds(ledA:=ledA, ledB:=ledB);
            snmpGetString(handle:=clientcon, OID:=ENTERPRISE_OID + "2.2.0",
v:=str);
            DebugMsg(message:="Client LED "+str);
            x := (snmpvar.int_value + 1) MOD 4;
            snmpSetInteger(handle:=clientcon, OID:=ENTERPRISE_OID + "1.1.0",
v:=x);
         END_IF;
         DebugMsg(message:="Variable OID was : " + snmpvar.oid);
      ELSE
         DebugFmt(Message:="snmpGetNextVar (rc=\1)", v1:=rc);
      END_CASE;
   END_WHILE;
END;
END_PROGRAM;
```

6.33 Examples - SNMP - Client

```
//--------------------------------------------------------------------------
-
// LEDclient.vpl, created 2020-02-19 13:24
//
// This program demonstrates communication between two RTCU devices, where one
// is the manager (trap listener) and the other a client (publishing agent).
//
// To visualize the communication, the LEDs of both devices are manipulated
// by each other. The manager initiates by setting a published integer
variable
// on the client, which then updates it's LEDs according to the value written,
// and then sends a trap to the manager, that updates it's LEDs accordingly
and
// writes the next value to the other LED on the client, which again responds
// with a trap, which the manager uses to update it's LEDs with and so on.
//--------------------------------------------------------------------------
-
INCLUDE rtcu.inc

// Use LAN1
#DEFINE SNMP_INTERFACE 2

// Exchange this with your real enterprise OID base.
#DEFINE ENTERPRISE_OID "1.3.6.1.4.1.65500."

// Put the actual IP address of the manager here.
#DEFINE MANAGER_IP "192.168.1.10" // example IP

// Configure the LEDs 1-4 in the job configuration manager
VAR_OUTPUT
   ledAgreen : BOOL; // LED 1
   ledAred   : BOOL; // LED 2
   ledBgreen : BOOL; // LED 3
   ledBred   : BOOL; // LED 4
END_VAR;

VAR
  netInfo    : netGetInformation;
  snmpvar    : snmpVariable;
  trapvar    : snmpVariable;
  iface      : SINT := SNMP_INTERFACE;
  runthread  : BOOL := FALSE;
  ledA       : INT;
  ledB       : INT;
  mancon     : SYSHANDLE;
  rouser     : snmpUser;
  rwuser     : snmpUser;
END_VAR;

FUNCTION authToStr : STRING;
VAR_INPUT
   auth : SINT;
END_VAR;
   CASE auth OF
      0 : authToStr := "";
      _SNMP_USM_MD5 : authToStr := "MD5";
      _SNMP_USM_SHA : authToStr := "SHA";
   ELSE
      authToStr := "Unknown???";
   END_CASE;
```

```
END_FUNCTION

FUNCTION encToStr : STRING;
VAR_INPUT
   enc : SINT;
END_VAR;
   CASE enc OF
      0 : encToStr := "";
      _SNMP_USM_AES : encToStr := "AES";
      _SNMP_USM_DES : encToStr := "DES";
   ELSE
      encToStr := "Unknown???";
   END_CASE;
END_FUNCTION

FUNCTION levelToStr : STRING;
VAR_INPUT
   level : SINT;
END_VAR;
   CASE level OF
      _SNMP_SEC_NONE : levelToStr := "None";
      _SNMP_SEC_AUTH : levelToStr := "AuthOnly";
      _SNMP_SEC_ENC  : levelToStr := "Auth+Enc";
   ELSE
      levelToStr := "Unknown???";
   END_CASE;
END_FUNCTION

FUNCTION printUSMUsersInfo;
VAR
   rc : INT;
END_VAR
rc := snmpAgentUsersGet(readonly:=rouser, writable:=rwuser);
DebugFmt(message:="snmpAgentUsersGet (rc=\1)", v1:=rc);
DebugMsg(message:="----------------------------------------");
DebugMsg(message:="Read-Only user:");
DebugMsg(message:="-----------USM user profile-------------");
DebugMsg(message:="Username is  : "+rouser.username);
DebugMsg(message:="Auth is      : "+authToStr(auth:=rouser.authentication));
DebugMsg(message:="Auth pass is : "+rouser.password);
DebugMsg(message:="Enc is       : "+encToStr(enc:=rouser.encryption));
DebugMsg(message:="Cryptkey is  : "+rouser.cryptkey);
DebugMsg(message:="Sec level is : "+levelToStr(level:=rouser.level));
DebugMsg(message:="----------------------------------------");
DebugMsg(message:="");
DebugMsg(message:="Read-Write capable user:");
DebugMsg(message:="-----------USM user profile-------------");
DebugMsg(message:="Username is  : "+rwuser.username);
DebugMsg(message:="Auth is      : "+authToStr(auth:=rwuser.authentication));
DebugMsg(message:="Auth pass is : "+rwuser.password);
DebugMsg(message:="Enc is       : "+encToStr(enc:=rwuser.encryption));
DebugMsg(message:="Cryptkey is  : "+rwuser.cryptkey);
DebugMsg(message:="Sec level is : "+levelToStr(level:=rwuser.level));
DebugMsg(message:="----------------------------------------");
END_FUNCTION;

FUNCTION getledstring : STRING;
VAR_INPUT
   idx : SINT;
END_VAR;
VAR
   str: STRING;
END_VAR
CASE idx OF
```

```
   1 :
   IF ledAgreen THEN
      IF ledAred THEN
         str := "A is yellow";
      ELSE
         str := "A is green";
      END_IF;
   ELSE
      IF ledAred THEN
         str := "A is red";
      ELSE
         str := "A is off";
      END_IF;
   END_IF;
   2:
   IF ledBgreen THEN
      IF ledBred THEN
         str := "B is yellow";
      ELSE
         str := "B is green";
      END_IF;
   ELSE
      IF ledBred THEN
         str := "B is red";
      ELSE
         str := "B is off";
      END_IF;
   END_IF;
END_CASE;
getledstring := str;
END_FUNCTION;

FUNCTION updateLeds;
VAR_INPUT
   ledA : INT;
   ledB : INT;
END_VAR;
   CASE ledA OF
      0:
      ledAgreen := OFF;
      ledAred := OFF;
      1:
      ledAgreen := ON;
      ledAred := OFF;
      2:
      ledAgreen := OFF;
      ledAred := ON;
      3:
      ledAgreen := ON;
      ledAred := ON;
   END_CASE;

   CASE ledB OF
      0:
      ledBgreen := OFF;
      ledBred := OFF;
      1:
      ledBgreen := ON;
      ledBred := OFF;
      2:
      ledBgreen := OFF;
      ledBred := ON;
      3:
      ledBgreen := ON;
```

```
      ledBred := ON;
   END_CASE;
   UPDATEIO;
END_FUNCTION;

FUNCTION printNetInfo
  netInfo(iface:=iface);
  DebugFmt(message:="Network interface \1:", v1:=iface);
  CASE netInfo.status OF
     0 : DebugMsg(message:="Interface is not present.");
     1 : DebugMsg(message:="Interface is not opened.");
     2 : DebugMsg(message:="Interface is not connected.");
     3 : DebugMsg(message:="Interface is connected.");
     4 : DebugMsg(message:="Interface link is up and waiting for IP
configuration.");
  ELSE
     DebugFmt(message:="Illegal net status? (\1)", v1:=netInfo.status);
  END_CASE;
  IF netInfo.status > 1 THEN
    DebugMsg(message:=" Phy addr " + netInfo.phy_addr);
    IF netInfo.dhcp THEN
        DebugMsg(message:=" Dynamic IP address");
    ELSE
        DebugMsg(message:=" Static IP address");
    END_IF;
    DebugMsg(message:=" IP addr " + sockIPToName(ip := netInfo.ip));
    DebugMsg(message:=" Mask    " + sockIPToName(ip := netInfo.subnetMask));
    DebugMsg(message:=" Gateway " + sockIPToName(ip := netInfo.gateway));
    IF netInfo.AutoDNS THEN
        DebugMsg(message:=" Automatic DNS");
    ELSE
        DebugMsg(message:=" Manual DNS");
    END_IF;
    DebugMsg(message:=" DNS1    " + sockIPToName(ip := netInfo.DNS1));
    DebugMsg(message:=" DNS2    " + sockIPToName(ip := netInfo.DNS2));
  END_IF;
END_FUNCTION;

PROGRAM LEDclient;
VAR
   deleteuser : snmpUser;
   traphandle : SYSHANDLE;
   oid        : STRING;
   vars       : INT;
   eventtype  : USINT;
   rc         : INT;
   str        : STRING;
   ix         : SINT;
END_VAR;
ledAgreen := FALSE;
ledAred := FALSE;
ledBgreen := FALSE;
ledBred := FALSE;
ledA := 0;
ledB := 0;
UPDATEIO;

rc := netOpen(iface:=iface);
DebugFmt(message:="netOpen (rc=\1)",v1:=rc);

// Preemptively delete persistent users.
deleteuser.username := "";
FOR ix:=1 TO 25 DO
   snmpUserSet(index:=ix, user:=deleteuser);
```

```
END_FOR;

// Configure read-write user
rwuser.username := "roed";
rwuser.password := "deor1234";
rwuser.cryptkey := "a0b3d543";
rwuser.authentication := _SNMP_USM_MD5;
rwuser.encryption := _SNMP_USM_AES;
rwuser.level := _SNMP_SEC_ENC;
rwuser.engineid := "12345678900001030507";

// Set USM user with index 1
snmpUserSet(index:=1, user:=rwuser);

WHILE NOT netConnected(iface:=iface) DO
   Sleep(delay:=2000);
END_WHILE;

// Output network information
printNetInfo();

// Set USM users for the publishing agent
rc := snmpAgentUsersSet(writable:=rwuser, readonly:=rouser);
DebugFmt(message:="snmpAgentUsersSet (rc=\1)", v1:=rc);

// Output USM users for the publishing agent
printUSMUsersInfo();

// Setup connection to the manager
rc := snmpConnect(
                  handle:=mancon,
                  host:=MANAGER_IP,
                  port:=162,
                  version:=_SNMP_VER_3,
                  engineid := "12345678900001030507",
                  usmuser:=ADDR(rwuser)
                 );
DebugFmt(message:="snmpConnect (rc=\1)", v1:=rc);

// Start the publishing agent
rc := snmpStartAgent(
                     engineid := "a9382949f343be38",
                     dtls     := 0, // disable dtls as we use USM
                     secure   := TRUE // only allow secure connections
                    );
DebugFmt(message:="snmpStartAgent (rc=\1)", v1:=rc);

// Publish some variables
rc := snmpPublishInteger(v:=ledA, oid:=ENTERPRISE_OID + "1.1", subtype:=1,
writable:=TRUE);
DebugFmt(message:="snmpPublishInteger (rc=\1)", v1:=rc);
rc := snmpPublishInteger(v:=ledB, oid:=ENTERPRISE_OID + "1.2", subtype:=1,
writable:=TRUE);
DebugFmt(message:="snmpPublishInteger (rc=\1)", v1:=rc);
rc := snmpPublishString(v:=getledstring(idx:=1), oid:=ENTERPRISE_OID + "2.1",
subtype:=0, writable:=TRUE);
DebugFmt(message:="snmpPublishString (rc=\1)", v1:=rc);
rc := snmpPublishString(v:=getledstring(idx:=2), oid:=ENTERPRISE_OID + "2.2",
subtype:=0, writable:=TRUE);
DebugFmt(message:="snmpPublishString (rc=\1)", v1:=rc);
rc := snmpPublishTimeTicks(v:=boardTimeSinceReset(), oid:=ENTERPRISE_OID +
"3", writable:=TRUE);
DebugFmt(message:="snmpPublishTimeTicks (rc=\1)", v1:=rc);
rc := snmpPublishOID(v:="1.3.6.4.1.5.333.32.2", oid:=ENTERPRISE_OID + "4",
```

```
writable:=TRUE);
DebugFmt(message:="snmpPublishOID (rc=\1)", v1:=rc);
rc := snmpPublishIP(v:="192.168.1.255", oid:=ENTERPRISE_OID + "5",
writable:=TRUE);
DebugFmt(message:="snmpPublishIP (rc=\1)", v1:=rc);
rc := snmpPublishFloat(v:=3.1416, oid:=ENTERPRISE_OID + "6", writable:=TRUE);
DebugFmt(message:="snmpPublishFloat (rc=\1)", v1:=rc);
rc := snmpPublishDouble(v:=3.14159265359, oid:=ENTERPRISE_OID + "7",
writable:=TRUE);
DebugFmt(message:="snmpPublishDouble (rc=\1)", v1:=rc);

BEGIN
// Handle events
   rc := snmpWaitEvent(timeout:=-1, event:=eventtype, oid:=oid, vars:=vars);
   CASE eventtype OF
      _SNMP_EVENT_NOEVENT :
         CASE rc OF
             0: DebugMsg(message:="ERROR: The function is not supported!");
            -2: DebugMsg(message:="ERROR: Invalid input");
            -5: DebugMsg(message:="WARNING: Event buffer overflow.");
         ELSE
            DebugFmt(message:="ERROR: unknown return code from snmpReadEvent
(rc=\1)", v1:=rc);
         END_CASE;
      _SNMP_EVENT_TIMEOUT :
      DebugMsg(message:="snmpWaitEvent: Timeout!");
      _SNMP_EVENT_TRAP :
      DebugMsg(message:="Trap received!");
      DebugMsg(message:="Trap oid is: " + oid);
      DebugFmt(message:="Vars to read : \1", v1:=vars);
      _SNMP_EVENT_SET :
      DebugMsg(message:="Set request received on OID " + oid);
   ELSE
      DebugFmt(message:="ERROR: snmpWaitEvent - unknown event type \1!",
v1:=eventtype);
   END_CASE;
   rc := 0;
   WHILE NOT(rc = -11) DO
      rc := snmpGetNextVar(data:=snmpvar);
      CASE rc OF
        0: DebugMsg(message:="snmpGetNextVar: Function not implemented!");
       -2: DebugMsg(message:="Invalid parameter.");
       -9: DebugFmt(message:="Invalid variable type \1", v1:=snmpvar.type);
      -11: DebugMsg(message:="No more data available.");
        1: DebugFmt(message:="Type found is \1", v1:=snmpvar.type);
         CASE snmpvar.type OF
            1: DebugFmt(message:="TimeTicks received = \4",
v4:=snmpvar.time_value);
            2: DebugFmt(message:="INTEGER received = \4",
v4:=snmpvar.int_value);
            3: DebugFmt(message:="UNSIGNED INTEGER received = \4",
v4:=snmpvar.uint_value);
            4: DebugFmt(message:="STRING received = " + snmpvar.str_value);
            5: DebugFmt(message:="Hex STRING received = " +
snmpvar.hex_value);
            6: DebugFmt(message:="OID received = " + snmpvar.oid_value);
            7: DebugFmt(message:="IP Address received = " + snmpvar.ip_value);
            8: DebugFmt(message:="FLOAT received = " +
floatToStr(v:=snmpvar.float_value) );
            9: DebugFmt(message:="DOUBLE received = " +
doubleToStr(v:=snmpvar.double_value) );
         ELSE
            DebugFmt(message:="Unknown var type received : \1",
v1:=snmpvar.type);
```

```
            END_CASE;
            DebugMsg(message:="Variable OID was : " + snmpvar.oid);
            IF strCompare(str1:=snmpvar.oid, str2:=ENTERPRISE_OID + "1.1") = 0
THEN
               ledA := INT(snmpvar.int_value);
               updateLeds(ledA:=ledA, ledB:=ledB);
               rc := snmpPublishString(v:=getledstring(idx:=1),
oid:=ENTERPRISE_OID + "2.1", subtype:=0, writable:=TRUE);
               IF rc <> 1 THEN
                  DebugFmt(message:="snmpPublishString error, rc=\1", v1:=rc);
               END_IF;
               rc := snmpcreateTrap(trap:=traphandle, oid:=ENTERPRISE_OID +
"100", uptime:=100);
               trapvar.type := 2;
               trapvar.oid:=ENTERPRISE_OID + "100.1";
               trapvar.int_value := snmpvar.int_value;
               rc := snmpAddTrapVariable(trap:=traphandle, data:=trapvar);
               IF rc <> 1 THEN
                  DebugFmt(message:="snmpAddTrapVariable error, rc=\1", v1:=rc);
               END_IF;
               rc := snmpSendTrap(trap:=traphandle, connection:=mancon);
               IF rc <> 1 THEN
                  DebugFmt(message:="Sending of trap failed (rc=\1)", v1:=rc);
               ELSE
                  DebugFmt(message:="Successfully sendt trap!", v1:=rc);
               END_IF;
               rc := snmpDestroyTrap(trap:=traphandle);
               IF rc <> 1 THEN
                  DebugFmt(message:="snmpDestroyTrap failed (rc=\1)", v1:=rc);
               END_IF;
            END_IF;
            IF strCompare(str1:=snmpvar.oid, str2:=ENTERPRISE_OID + "1.2") = 0
THEN
               ledB := INT(snmpvar.int_value);
               updateLeds(ledA:=ledA, ledB:=ledB);
               rc := snmpPublishString(v:=getledstring(idx:=2),
oid:=ENTERPRISE_OID + "2.2", subtype:=0, writable:=TRUE);
               IF rc <> 1 THEN
                  DebugFmt(message:="snmpPublishString error, rc=\1", v1:=rc);
               END_IF;
               rc := snmpcreateTrap(trap:=traphandle, oid:=ENTERPRISE_OID +
"101", uptime:=boardTimeSinceReset());
               trapvar.type := 2;
               trapvar.oid:=ENTERPRISE_OID + "100.2";
               trapvar.int_value := snmpvar.int_value;
               rc := snmpAddTrapVariable(trap:=traphandle, data:=trapvar);
               IF rc <> 1 THEN
                  DebugFmt(message:="snmpAddTrapVariable error, rc=\1", v1:=rc);
               END_IF;
               rc := snmpSendTrap(trap:=traphandle, connection:=mancon);
               IF rc <> 1 THEN
                  DebugFmt(message:="Sending of trap failed (rc=\1)", v1:=rc);
               ELSE
                  DebugFmt(message:="Successfully sendt trap!", v1:=rc);
               END_IF;
               rc := snmpDestroyTrap(trap:=traphandle);
               IF rc <> 1 THEN
                  DebugFmt(message:="snmpDestroyTrap failed (rc=\1)", v1:=rc);
               END_IF;
            END_IF;
         ELSE
            DebugFmt(message:="snmpGetNextVar (rc=\1)", v1:=rc);
         END_CASE;
      END_WHILE;
```

```
END;
END_PROGRAM;
```

6.34 Examples - MODBUS Network - Slave (simple)

```
//------------------------------------------------------------------------
-
// SimpleSlave.vpl, created 2011-03-22 12:00
//
// This simple MODBUS slave is using the I/0 Extension to share its resources.
//
// IMPORTANT:
// As the memory, analog and digital in-/outputs are controlled by the
firmware
// these must not be mapped using the Job configuration.
//
// LED's and switches can be controlled by the application as these are not
// accessible through MODBUS requests.
//
// To see which MODBUS commands that are supported by the I/O Extensions
please
// see the "I/O Extension -> MODBUS commands" section of the online help
//
// Alternative communication with the master form the application:
//
// The only way to communicate with the MODBUS master from the application is
// through memory registers, as these can be accessed without mapping in the
// job, through the use of memioRead/memioReadX and memioWrite/memioWriteX
// functions.
//
// Note:
// When a memory register are accessed through MODBUS they are read as two
// 16bits addresses as they are 32bits long.
//
// The memory registers can be read as input register or read/written as
// holding registers.
//
// Start index of the memory registers are 0x1000 hex or 4096 decimal.
//
// MODBUS Configuration:
// Configuration of connection settings and node id is done through
// I/0 Extension, please see 'Using the I/O Extension' in the online help.
//------------------------------------------------------------------------
-
INCLUDE rtcu.inc

VAR_INPUT
   SW : ARRAY [1..3] OF BOOL;
END_VAR;

VAR_OUTPUT
   LED : ARRAY [1..4] OF BOOL;
END_VAR;

//------------------------------------------------------------------------
-
// THREAD_BLOCK: RegisterMonitor
//    Monitor IO memory for changes.
//
// INPUT
//    run   : Start/Stop monitoring
//    first : First index to monitor
//    last  : Last index to monitor
//------------------------------------------------------------------------
-
```

```
THREAD_BLOCK RegisterMonitor;
VAR_INPUT
   run   : BOOL := TRUE;
   first : INT := 1;
   last  : INT := 4096;
END_VAR;
VAR
   cur   : ARRAY [1..4096] OF DINT;
   old   : ARRAY [1..4096] OF DINT;
   msg   : STRING := "register \1 has changed from \4 to ";
   i     : INT;
   rc    : INT;
   rc_o  : INT;
   err   : STRING := "RegisterMonitor - Failed to read memory ";
END_VAR;
   DebugMsg(message := "RegisterMonitor - Started");
   rc := 0;
   rc_o := 0;
   WHILE run DO
      // read current memory
      rc := memioReadX(index:=first, mem:=ADDR(cur), len:=(last - first + 1));
      IF NOT (rc = 0) THEN
         IF NOT (rc = rc_o) THEN
            CASE (rc) OF
               1 : DebugMsg(message := err + "'Invalid parameters'");
               2 : DebugMsg(message := err + "'Index range was out of
bounds'");
            END_CASE;
            rc_o := rc;
         END_IF;
         Sleep(delay := 1000);
      ELSE
         FOR i := first TO last DO
            IF NOT ( cur[i] = old[i] ) THEN
               DebugFmt(message := msg + dintToStr(v := cur[i]),
                        v1 := i, v4 := old[i]);
               old[i] := cur[i];
            END_IF;
         END_FOR;
      END_IF;
   END_WHILE;
   DebugMsg(message := "RegisterMonitor - Ended");
END_THREAD_BLOCK;

//-------------------------------------------------------------------------
-
// PROGRAM: SimpleSlave
//   Monitor switches and LED's
//-------------------------------------------------------------------------
-
PROGRAM SimpleSlave;
VAR
   regMon   : RegisterMonitor;
   regMonSW : RF_TRIG;

   i        : INT;
   clk      : DINT;
   sn       : DINT;
END_VAR;

   DebugFmt(message := "Device \4 ready", v4 := boardSerialNumber());

   memioWrite(index:=1, value:=boardSerialNumber());
   DebugFmt(message := "Serial number stored in register 0x1000h + 0x1001h");
```

```
   // get initial message
   regMonSW(Trig := NOT SW[1]);

   LED[1] := ON;
BEGIN
   regMonSW(Trig := SW[1]);
   // Start / stop monitor thread
   IF regMonSW.rq THEN
      regMon.run := TRUE;
      DebugMsg(message := "Turn off switch 1 to stop register monitoring.");
   ELSIF regMonSW.fq THEN
      regMon.run := FALSE;
      DebugMsg(message := "Turn on switch 1 to start register monitoring.");
   END_IF;

   // Start thread if stopped and should be running
   IF regMon.run AND NOT regMon._running THEN
      regMon();
   END_IF;

   LED[2] := NOT regMon.run;
END;
END_PROGRAM;
```

6.35 Examples - Generic OneWire Example

```
//-------------------------------------------------------------------------
--
// DS2408.vpl, created 2012-05-09 10:19
//
// Demonstrate communication with Maxim DS2408 1-Wire device, using the
generic
// 1-Wire interface.
//
// DESCRIPTION of the DS2408
//    The DS2408 is an 8-channel, programmable I/O 1-Wire chip. PIO outputs
are
//    configured as open-drain and provide an on resistance of 100Ω max.
//    A robust PIO channel-access communication protocol ensures that PIO
//    output-setting changes occur error-free. A data-valid strobe output can
//    be used to latch PIO logic states into external circuitry such as a D/A
//    converter (DAC) or microcontroller data bus.
//
//
// HARDWARE configuration
//    The hardware used to communicate with is based on the reference design
//    found in the datasheet of the DS2408 (version 19-5702; 12/10), and an
//    RTCU AX9 pro, which supply the DS2408 with 3.3VDC from DCOUT33.
//
//    AX9 pro              DS2408            External
//
//    1-Wire      -->   IO         P0  -->   SW1
//    SGND        -->   GND        P1  -->   SW2
//    DCOUT33     -->   VCC        P2  -->   SW3
//                                 P3  -->   SW4
//                                 P4  -->   LED1
//                                 P5  -->   LED2
//                                 P6  -->   LED3
//                                 P7  -->   LED4
//
//-------------------------------------------------------------------------
--
INCLUDE rtcu.inc

//  Output variables that can be configured via the configuration dialog
VAR_OUTPUT
   Connected      : BOOL; | Connected to the DS2408
   NotConnected   : BOOL; | NOT connected to the DS2408
END_VAR;

//-------------------------------------------------------------------------
--
// Special 1-Wire function to read the registers of a
// Maxim DS2408 - 1-Wire 8-Channel Addressable Switch
//
// INPUT
//    device   : INT (default 1 (1st.))
//       device number withing the family is handling multiple DS2408 on same
net
//
// OUTPUT
//    P     : ARRAY[0..7] OF BOOL
//              PIO Logic State
//    PL    : ARRAY[0..7] OF BOOL
//              PIO Output Latch State Register
//    AL    : ARRAY[0..7] OF BOOL
```

```
//              PIO Activity Latch State Register
//    SM     : ARRAY[0..7] OF BOOL
//              Conditional Search Channel Selection Mask
//    SP     : ARRAY[0..7] OF BOOL
//              Conditional Search Channel Polarity Selection
//    PLS    : BOOL
//              Pin or Activity Latch Select
//    CT     : BOOL
//              Conditional Search Logical Term
//    ROS    : BOOL
//              RSTZ Pin Mode Control
//    PORL   : BOOL
//              Power-On Reset Latch
//    VCCP   : BOOL
//              VCC Power Status
//    ready  : BOOL
//              Last update successful
//
//-------------------------------------------------------------------------
--
FUNCTION_BLOCK owDS2408;
VAR_INPUT
   device    : INT     := 1;       // use device found with owSearchX()
END_VAR;
VAR_OUTPUT
   P         : ARRAY[0..7] OF BOOL; // PIO Logic State
   PL        : ARRAY[0..7] OF BOOL; // PIO Output Latch State Register
   AL        : ARRAY[0..7] OF BOOL; // PIO Activity Latch State Register
   SM        : ARRAY[0..7] OF BOOL; // Conditional Search Channel Selection
Mask
   SP        : ARRAY[0..7] OF BOOL; // Conditional Search Channel Polarity
Selection
   PLS       : BOOL    := FALSE;     // Pin or Activity Latch Select
   CT        : BOOL    := FALSE;     // Conditional Search Logical Term
   ROS       : BOOL    := FALSE;     // RSTZ Pin Mode Control
   PORL      : BOOL    := FALSE;     // Power-On Reset Latch
   VCCP      : BOOL    := FALSE;     // VCC Power Status
   ready     : BOOL    := FALSE;     // Last update successful
END_VAR;
VAR
   regs      : ARRAY[16#88..16#8F] OF SINT; // Register map
   crc       : INT;                 // inverted crc16 all data sent and
received.
   rc        : INT;
   i         : SINT;
END_VAR;
ready := FALSE;

// RST -> PD -> Select
rc := owAccess(family := 16#29, device := device);
IF rc <> 0 THEN
   DebugFmt(message := "owDS2408 : Failed to access device (\1)", v1 := rc);
   RETURN;
END_IF;

// RPR - Command "Read PIO Registers"
owWrite(data := 16#F0);
// TA - Target address
owWrite(data := 16#88); // TA1 (T7:T0)
owWrite(data := 16#00); // TA2 (T15:T8)
owReadData(data := ADDR(regs), size := SIZEOF(regs));
owReadData(data := ADDR(crc),  size := SIZEOF(crc));
owRelease();
```

```
// Validate received data against received CRC16
IF NOT (crcCalculate(
      buffer      := ADDR(regs),
      length      := SIZEOF(regs),
      order       := 16,
      polynom     := 16#0000_8005,  // X^16+X^15+X^2+1
      preset      := 16#0000_66CC,  // the CRC16 of RPR + TA using same
polynom
      direct      := TRUE,
      finalxor    := NOT (crc),     // check against received inverted crc
      reverseData := TRUE,
      reverseCRC  := TRUE
   ) = 0) THEN
   DebugMsg(message := "owDS2408 : Failed to read register (CRC error)");
   RETURN;
END_IF;

// Map register values to output values
FOR i := 0 TO 7 DO
   P[i] := (regs[16#88] AND shl8(in := 1, n := i))  <> 0;
END_FOR;
FOR i := 0 TO 7 DO
   PL[i] := (regs[16#89] AND shl8(in := 1, n := i)) <> 0;
END_FOR;
FOR i := 0 TO 7 DO
   AL[i] := (regs[16#8A] AND shl8(in := 1, n := i)) <> 0;
END_FOR;
FOR i := 0 TO 7 DO
   SM[i] := (regs[16#8B] AND shl8(in := 1, n := i)) <> 0;
END_FOR;
FOR i := 0 TO 7 DO
   SP[i] := (regs[16#8C] AND shl8(in := 1, n := i)) <> 0;
END_FOR;

PLS   := (regs[16#8D] AND 16#01) <> 0;
CT    := (regs[16#8D] AND 16#02) <> 0;
ROS   := (regs[16#8D] AND 16#04) <> 0;
PORL  := (regs[16#8D] AND 16#08) <> 0;
VCCP  := (regs[16#8D] AND 16#80) <> 0;

ready := TRUE; // all data received successful
END_FUNCTION_BLOCK;

//-------------------------------------------------------------------------
--
// Special 1-Wire function to set the Control/Status Register.
// This will also clear the power-on reset latch (PORL) status.
//
// INPUTS
//    device   : INT (default 1 (1st.))
//       device number withing the family is handling multiple DS2408 on same
net
//    PLS   : BOOL
//              Pin or Activity Latch Select configuration
//    CT    : BOOL
//              Conditional Search Logical Term configuration
//    ROS   : BOOL
//              RSTZ Pin Mode Control configuration
//-------------------------------------------------------------------------
--
FUNCTION owDS2408Cfg;
VAR_INPUT
   device   : INT    := 1;
   PLS      : BOOL   := FALSE;   // Pin or Activity Latch Select
```

```
   CT       : BOOL   := FALSE;   // Conditional Search Logical Term
   ROS      : BOOL   := FALSE;   // RSTZ Pin Mode Control
END_VAR;

VAR
   reg      : SINT   := 16#00;
   val      : SINT   := 16#00;
   rc       : INT;
END_VAR;

IF PLS THEN reg := reg OR 16#01; END_IF;
IF CT  THEN reg := reg OR 16#02; END_IF;
IF ROS THEN reg := reg OR 16#04; END_IF;

// RST -> PD -> Select
rc := owAccess(family := 16#29, device := device);
IF rc <> 0 THEN
   DebugFmt(message := "owDS2408Cfg : Failed to access device (\1)", v1 :=
rc);
   RETURN;
END_IF;

owWrite(data := 16#CC);
owWrite(data := 16#8D);
owWrite(data := 16#00);
owWrite(data := reg);
owRelease();

rc := owAccess(family := 16#29, device := device);
IF rc <> 0 THEN
   DebugFmt(message := "owDS2408Cfg (verify) : Failed to access device (\1)",
            v1 := rc);
   RETURN;
END_IF;

owWrite(data := 16#F0); // read reg
owWrite(data := 16#8D);
owWrite(data := 16#00);
owRead(data := val);
owRelease();

IF (val AND 16#0007) <> reg THEN
   DebugFmt(message := "owDS2408Cfg (validate) : Failed (\1 <> \2)",
            v1 := (val AND 16#0007), v2 := (reg AND 16#00FF));
END_IF;
END_FUNCTION;

//-------------------------------------------------------------------------
--
// Special 1-Wire function to reset the PIO Activity Latch State Register
//
// INPUTS
//    device   : INT (default 1 (1st.))
//       device number withing the family is handling multiple DS2408 on same
net
//-------------------------------------------------------------------------
--
FUNCTION owDS2408ResetAL : BOOL;
VAR_INPUT
   device   : INT := 1;
END_VAR;

VAR
   ack      : SINT;
```

```
   rc       : INT;
END_VAR;

rc := owAccess(family := 16#29, device := device);
IF rc <> 0 THEN
   DebugFmt(message := "owDS2408ResetAL : Failed to access device (\1)",
            v1 := rc);
   RETURN;
END_IF;

owWrite(data := 16#C3);
owRead(data := ack);
owRelease();

IF (ack AND 16#00FF) <> 16#00AA THEN
   DebugFmt(message := "owDS2408ResetAL : Failed to reset latch (ack = \1)",
            v1 := (ack AND 16#00FF));
   RETURN;
END_IF;
owDS2408ResetAL := TRUE;
END_FUNCTION;

//---------------------------------------------------------------------------
--
// Special 1-Wire function to set the PIO pins state.
//
// output pin status will be ignored when using pins as inputs.
//
// INPUTS
//    device   : INT (default 1 (1st.))
//       device number withing the family is handling multiple DS2408 on same
net
//
//    p0       : BOOL (default OFF)
//       pin 0 ON/OFF
//    p1       : BOOL (default OFF)
//       pin 1 ON/OFF
//    p2       : BOOL (default OFF)
//       pin 2 ON/OFF
//    p3       : BOOL (default OFF)
//       pin 3 ON/OFF
//    p4       : BOOL (default OFF)
//       pin 4 ON/OFF
//    p5       : BOOL (default OFF)
//       pin 5 ON/OFF
//    p6       : BOOL (default OFF)
//       pin 6 ON/OFF
//    p7       : BOOL (default OFF)
//       pin 7 ON/OFF
//---------------------------------------------------------------------------
--
FUNCTION owDS2408SetOutput;
VAR_INPUT
   device   : INT    := 1;
   p0       : BOOL   := OFF;
   p1       : BOOL   := OFF;
   p2       : BOOL   := OFF;
   p3       : BOOL   := OFF;
   p4       : BOOL   := OFF;
   p5       : BOOL   := OFF;
   p6       : BOOL   := OFF;
   p7       : BOOL   := OFF;
END_VAR;
```

```
VAR
   mask     : SINT := 0; // the pin stat after the update
   ack      : SINT;
   rc       : INT;
   i        : SINT;
   status   : SINT;
END_VAR;

IF p0 THEN mask := mask OR 16#01; END_IF;
IF p1 THEN mask := mask OR 16#02; END_IF;
IF p2 THEN mask := mask OR 16#04; END_IF;
IF p3 THEN mask := mask OR 16#08; END_IF;
IF p4 THEN mask := mask OR 16#10; END_IF;
IF p5 THEN mask := mask OR 16#20; END_IF;
IF p6 THEN mask := mask OR 16#40; END_IF;
IF p7 THEN mask := mask OR 16#80; END_IF;

rc := owAccess(family := 16#29, device := device);
IF rc <> 0 THEN
   DebugFmt(message := "owDS2408SetOutput : Failed to access device (\1)", v1
:= rc);
   RETURN;
END_IF;

owWrite(data := 16#5A);
owWrite(data := mask);
owWrite(data := NOT(mask));
owRead(data := ack);
owRead(data := status);
owRelease();

IF (ack AND 16#00FF) <> 16#00AA THEN
   DebugFmt(message := "owDS2408SetOutput : Failed to set new pin state (reply
\1)",
            v1 := (ack AND 16#00FF));
END_IF;
END_FUNCTION;


//-------------------------------------------------------------------------
-
// Application to run
//    Monitor the state of the DS2408 inputs (pin 0-3), and maps it to
//    the outputs (pin 4-7).
//-------------------------------------------------------------------------
-
PROGRAM DS2408;
VAR
   DS2408   : owDS2408;
   id       : STRING;
   cnt      : INT;
   i        : INT;
   upd      : BOOL;
END_VAR;

// Enable/Disable 3.3VDC output to supply external LEDs
boardDCOut(enable:=ON);

BEGIN
   IF connected THEN
      Sleep(delay := 500); // allow DS2408 to detect changes on inputs
      DS2408(device := 1);
   END_IF;
   IF DS2408.ready THEN
```

```
      owDS2408ResetAL(device := 1); // reset the latches as we got them
      upd := FALSE;
      IF DS2408.porl THEN
         DebugMsg(Message := "Power-On Reset detected.");
         owDS2408Cfg(device := 1, ros := ON);
         upd := TRUE;
      ELSE
         FOR i := 0 TO 3 DO
            IF DS2408.al[i] THEN
               DebugFmt(message := "DS2408 : PIN \1 = \2",
                        v1 := i, v2 := SINT(DS2408.p[i]));
               upd := TRUE;
            END_IF;
         END_FOR;
      END_IF;
      IF upd THEN
         owDS2408SetOutput(
            device := 1,
            p4 := DS2408.p[0],
            p5 := DS2408.p[1],
            p6 := DS2408.p[2],
            p7 := DS2408.p[3]);
      END_IF;
   ELSE
      IF NOT (id = "") THEN
         DebugMsg(message := "Lost connection to DS2408 device 1 '" + id
+"'");
         id := "";
      END_IF;
      cnt := owSearchX(first := 16#29, last := 16#29); // only search for
DS2408 devices
      IF cnt > 0 THEN
         DebugFmt(message := "Found \1 DS2408 1-Wire devices.", v1 := cnt);
         id := owGetID(device := 1, family := 16#29);
         DebugFmt(message := "Using DS2408 1-Wire device = " + id);
      ELSIF cnt < 0 THEN
         DebugMsg(message := "Failed to search for DS2408 1-Wire devices.");
      END_IF;
   END_IF;
   Connected    := (id <> "");
   NotConnected := NOT Connected;
END;
END_PROGRAM;
```

6.36 Examples - Wired M-Bus

```
//------------------------------------------------------------------------
-
//
// This example scans for devices on wired M-Bus and requests data from the
found devices.
// By sending an SMS containing a filter, a new scan will be done.
//
//------------------------------------------------------------------------
-
INCLUDE rtcu.inc
// Uncomment math.inc to add math library support.
INCLUDE math.inc

//  Input variables that can be configured via the configuration dialog (These
are global)
VAR_INPUT

END_VAR;

//  Output variables that can be configured via the configuration dialog
(These are global)
VAR_OUTPUT

END_VAR;

//  These are the global variables of the program
VAR
   mb        : SYSHANDLE;
   // secondary address of found devices
   devices   : ARRAY [1..10] OF STRING;
   // Number of found devices
   dev_count : INT:=0;

   clock     : clockLinsecToTime;
   recInfo   : mbusRecordGetInfo;
END_VAR;


// Convert linsec to time string
FUNCTION GetTime:STRING;
VAR_INPUT
   linsec:DINT;
END_VAR;

   clock(linsec:=linsec);
   GetTime:=strFormat(format:="\1:\2:\3", v1:=clock.hour, v2:=clock.minute,
v3:=clock.second);
END_FUNCTION;

// Convert linsec to date string
FUNCTION GetDate:STRING;
VAR_INPUT
   linsec:DINT;
END_VAR;
   clock(linsec:=linsec);
   GetDate:=strFormat(format:="\1-\2-\3", v1:=clock.year, v2:=clock.month,
v3:=clock.day);
END_FUNCTION;
```

```
FUNCTION DumpRecords;
VAR
   rc    : INT;
   count : INT;
   i     : INT;
   str, unit : STRING;
   scale : FLOAT;
   type  : INT;
   d     : DINT;
END_VAR;
   count := mbusRecordCount(handle:=mb);
   DebugFmt(message:=" Records: \1", v1:=count);


   FOR i := 0 TO count DO
      recInfo(handle:=mb, index:=i);
      IF recInfo.ready THEN
         str := "  Record "+intToStr(v:=i)+", Type: " +
intToStr(v:=recInfo.type);

         scale:=1.0;
         unit:="";

         str := str + ", VIF: "+sintToHex(v:=recInfo.VIF)+", VIFE: "
            + sintToHex(v:=recInfo.VIFE[1])+" "+sintToHex(v:=recInfo.VIFE[2]);
         DebugFmt(message:=str);
         DebugFmt(message:="   Addr: "+recInfo.address);
         DebugFmt(message:="   Dev: \1, Func: \2, Tar \4, Stor:
"+dintToStr(v:=recInfo.storage),
                           v1:=recInfo.device, v2:=recInfo.func,
v4:=recInfo.tariff);

         DebugFmt(message:="   Type: "+recInfo.text);

         type := mbusRecordGetType(handle:=mb, index:=i);
         DebugFmt(message:="   Data Type: \1", v1:=type);

         IF recInfo.VIF = 16#14 THEN
            // Volume [1e-2  m^3]
            unit := "m^3";
            scale := 0.01;
         END_IF;


         IF type = _MBUS_TYPE_INT32 THEN
            rc := mbusRecordGetInt(handle:=mb, index:=i, value:=d);
            DebugMsg(message:="   Value:     "+floatToStr(v:=FLOAT(d)*scale)+"
"+unit);
         END_IF;
         IF type = _MBUS_TYPE_TIME THEN
            rc := mbusRecordGetLinsec(handle:=mb, index:=i, value:=d);
            IF recInfo.VIF = 16#6D THEN
               // time + date
               DebugMsg(message:="   Value:     "+GetDate(linsec:=d)+"
"+GetTime(linsec:=d));
            ELSE
               // Just date
               DebugMsg(message:="   Value:     "+GetDate(linsec:=d));
            END_IF;
         END_IF;

      END_IF;
   END_FOR;
END_FUNCTION;
```

```
// mbusScanCallback
//
// Description:
//   Declaration of the function called when a slave is detected while
scanning
//
// Input:
//   handle           - The handle to the M-bus connection.
//   primary          - The primary address (1..250)
//   secondary        - The secondary address
//
// Returns:
//   none
//
FUNCTION CALLBACK OnScan;
VAR_INPUT
   handle      : SYSHANDLE;
   secondary   : STRING;
   primary     : INT;
END_VAR;
VAR
   tmp:INT;
END_VAR;
   DebugFmt(message:="device found: \1: "+secondary, v1:=primary);
   IF secondary <> "" THEN
      IF dev_count < 10 THEN
         dev_count:=dev_count+1;
         devices[dev_count]:= secondary;
      END_IF;
   END_IF;

END_FUNCTION;

FUNCTION CALLBACK OnProgress;
VAR_INPUT
   handle      : SYSHANDLE;
   secondary   : STRING;
   primary     : INT;
END_VAR;
VAR
   tmp:INT;
END_VAR;
   DebugFmt(message:="scanning: \1: "+secondary, v1:=primary);
END_FUNCTION;

PROGRAM master;
// These are the local variables of the program block
VAR
   rc    : INT;
   i     : INT;
   count : INT := 0;
   sms   : gsmIncomingSMS;
   info  : mbusRecordSlaveInfo;
END_VAR;
// The next code will only be executed once after the program starts
   rc := mbusOpen(handle:=mb, type:=1, baud:=2400);
   DebugFmt(message:="mbusOpen(): \1", v1:=rc);

   dev_count:=0;
   rc := mbusScan(handle:=mb, cb_found:=@OnScan, cb_progress:=@OnProgress,
smode:=2, mask:="FFFFFFFFFFFFFFFF");
```

```
   DebugFmt(message:="mbusScan(): \1", v1:=rc);


BEGIN
// Code from this point until END will be executed repeatedly
   sms();
   IF sms.status > 0 THEN
      // Use SMS message as filter for scan for secondary addresses
      DebugFmt(message:="Scan: "+sms.message);
      dev_count:=0;
      rc := mbusScan(handle:=mb, cb_found:=@OnScan, cb_progress:=@OnProgress,
smode:=2, mask:=sms.message);
      DebugFmt(message:="mbusScan(): \1", v1:=rc);

      FOR i := 1 TO dev_count DO
         DebugFmt(message:="Device \1: "+devices[i], v1:=i);
      END_FOR;
   END_IF;

   IF count > 10 THEN

      FOR i:= 1 TO dev_count DO
         // Request data from device
         rc := mbusDataRequest(
                               handle   := mb,
                               sec_addr := devices[i]
                              );
         DebugFmt(message := "  mbusDataRequest("+devices[i]+")=\1", v1 :=
rc);

         info(handle:=mb);
         IF info.ready THEN
            DebugFmt(message:="Info for \4:", v4:=info.id);
            DebugFmt(message:=" Enc:  \1", v1:=info.enc_state);
            DebugFmt(message:=" Man:  "+info.manufacturer);
            DebugFmt(message:=" Ver:  \1", v1:=info.version);
            DebugFmt(message:=" Med:  \1", v1:=info.medium);
            DebugFmt(message:=" AN :  \1", v1:=info.accessnumber);
            DebugFmt(message:=" Sta:  \1", v1:=info.status);
            DebugFmt(message:=" Addr: "+info.sec_addr);
            DebugFmt(message:=" Sig:  \1", v1:=info.signal);

            DumpRecords();
         ELSE
            DebugMsg(message:="no response");
        END_IF;
      END_FOR;
      count := 0;
   ELSE
      count := count + 1;
      Sleep(delay:=1000);
   END_IF;

END;

END_PROGRAM;
```

6.37 Examples - Wireless M-Bus Receiver

```
//--------------------------------------------------------------------------
-
// Wireless M-Bus receiver example
// This example listens for data from the device from the OMS specification
Annex N.2.3.
//--------------------------------------------------------------------------
-
INCLUDE rtcu.inc
// Uncomment math.inc to add math library support.
INCLUDE math.inc

//  Input variables that can be configured via the configuration dialog (These
are global)
VAR_INPUT

END_VAR;

//  Output variables that can be configured via the configuration dialog
(These are global)
VAR_OUTPUT

END_VAR;

//  These are the global variables of the program
VAR
   mb      : SYSHANDLE;
   clock   : clockLinsecToTime;
   recInfo : mbusRecordGetInfo;
END_VAR;

FUNCTION RegisterDevices;
VAR
   rc  : INT;
   key : ARRAY[1..16] OF USINT;
END_VAR;

   // wM-Bus Meter with integrated radio and Security profile B from OMS
specification Annex N.2.3.
   key[1] := 16#00;
   key[2] := 16#01;
   key[3] := 16#02;
   key[4] := 16#03;
   key[5] := 16#04;
   key[6] := 16#05;
   key[7] := 16#06;
   key[8] := 16#07;
   key[9] := 16#08;
   key[10] := 16#09;
   key[11] := 16#0a;
   key[12] := 16#0b;
   key[13] := 16#0c;
   key[14] := 16#0d;
   key[15] := 16#0e;
   key[16] := 16#0f;

   rc := mbusSlaveRegister(handle:=mb, sec_addr:="1234567893153303", index:=1,
key:=ADDR(key));
   DebugFmt(message:="mbusSlaveRegister(): \1", v1:=rc);
END_FUNCTION;
```

```
FUNCTION UnRegisterDevices;
VAR
   rc : INT;
   i  : SINT;
END_VAR;
   FOR i:=1 TO 64 DO
      rc := mbusSlaveUnRegister(handle:=mb, index:=i);
      DebugFmt(message:="mbusSlaveUnRegister(): \1", v1:=rc);
   END_FOR;
END_FUNCTION;


FUNCTION GetTime:STRING;
VAR_INPUT
   linsec : DINT;
END_VAR;
   clock(linsec:=linsec);
   GetTime:=strFormat(format:="\1:\2:\3", v1:=clock.hour, v2:=clock.minute,
v3:=clock.second);
END_FUNCTION;

FUNCTION GetDate:STRING;
VAR_INPUT
   linsec : DINT;
END_VAR;
   clock(linsec:=linsec);
   GetDate:=strFormat(format:="\1-\2-\3", v1:=clock.year, v2:=clock.month,
v3:=clock.day);
END_FUNCTION;


FUNCTION DumpRecords;
VAR
   rc, type  : INT;
   count, i  : INT;
   str, unit : STRING;
   scale     : FLOAT;
   d         : DINT;
END_VAR;
      count := mbusRecordCount(handle:=mb);
      DebugFmt(message:=" Records: \1", v1:=count);


      FOR i := 0 TO count DO
         recInfo(handle:=mb, index:=i);
         IF recInfo.ready THEN
            str := "  Record "+intToStr(v:=i)+", Type: " +
intToStr(v:=recInfo.type);

            scale:=1.0;
            unit:="";

            str := str + ", VIF: "+sintToHex(v:=recInfo.VIF)+", VIFE: "
                + sintToHex(v:=recInfo.VIFE[1])+"
"+sintToHex(v:=recInfo.VIFE[2]);
            DebugFmt(message:=str);
            DebugFmt(message:="   Addr: "+recInfo.address);
            DebugFmt(message:="   Dev: \1, Func: \2, Tar \4, Stor:
"+dintToStr(v:=recInfo.storage),
               v1:=recInfo.device, v2:=recInfo.func, v4:=recInfo.tariff);

            DebugFmt(message:="   Type: "+recInfo.text);
```

```
            type := mbusRecordGetType(handle:=mb, index:=i);
            DebugFmt(message:="   Data Type: \1", v1:=type);

            IF recInfo.VIF = 16#14 THEN
               // Volume [1e-2  m^3]
               unit := "m^3";
               scale := 0.01;
            END_IF;


            IF type = _MBUS_TYPE_INT32 THEN
               rc := mbusRecordGetInt(handle:=mb, index:=i, value:=d);
               DebugMsg(message:="   Value:     "+floatToStr(v:=FLOAT(d)
*scale)+" "+unit);
            END_IF;
            IF type = _MBUS_TYPE_TIME THEN
               rc := mbusRecordGetLinsec(handle:=mb, index:=i, value:=d);
               IF recInfo.VIF = 16#6D THEN
                  // time + date
                  DebugMsg(message:="   Value:     "+GetDate(linsec:=d)+"
"+GetTime(linsec:=d));
               ELSE
                  // Just date
                  DebugMsg(message:="   Value:     "+GetDate(linsec:=d));
               END_IF;
            END_IF;

         END_IF;
      END_FOR;


END_FUNCTION;


PROGRAM wmbus_rec;
// These are the local variables of the program block
VAR
   rc   : INT;
   info : mbusRecordSlaveInfo;
END_VAR;
// The next code will only be executed once after the program starts

   rc := mbusOpen(type:=2, handle:=mb, mode:=_MBUS_MODE_T1_C);
   DebugFmt(message:="mbusOpen(): \1", v1:=rc);


   // Enable this line to clear the registrations
   //UnRegisterDevices();

   // Create device registrations
   RegisterDevices();


   // Only receive from registered devices
   rc := mbusFilterEnable(handle:=mb, enable:=TRUE);
   DebugFmt(message:="mbusFilterEnable(): \1", v1:=rc);

BEGIN
// Code from this point until END will be executed repeatedly

   // Wait for 15 seconds for data from 1234567893153303
   rc := mbusDataReceive(handle:=mb, filter:="1234567893153303", timeout:=15);
   DebugFmt(message:="mbusDataReceive: \1", v1:=rc);
```

```
    IF rc = 1 THEN
       info(handle:=mb);
       IF info.ready THEN
          DebugFmt(message:="Info for \4:", v4:=info.id);
          DebugFmt(message:=" Enc:  \1", v1:=info.enc_state);
          DebugFmt(message:=" Man:  "+info.manufacturer);
          DebugFmt(message:=" Ver:  \1", v1:=info.version);
          DebugFmt(message:=" Med:  \1", v1:=info.medium);
          DebugFmt(message:=" AN :  \1", v1:=info.accessnumber);
          DebugFmt(message:=" Sta:  \1", v1:=info.status);
          DebugFmt(message:=" Addr: "+info.sec_addr);
          DebugFmt(message:=" Sig:  \1", v1:=info.signal);
       END_IF;
       DumpRecords();
    END_IF;

END;

END_PROGRAM;
```

6.38 Examples - Wireless M-Bus sender

```
//--------------------------------------------------------------------------
-
// Wireless M-Bus sender example
// This example sends the data from the OMS specification Annex N.2.3.
//--------------------------------------------------------------------------
-
INCLUDE rtcu.inc
// Uncomment math.inc to add math library support.
//INCLUDE math.inc

//  These are the global variables of the program
VAR
   mb  : SYSHANDLE;
END_VAR;

// Convert hex string into SINT array
FUNCTION ParseString:INT;
VAR_INPUT
   str : STRING;
   dst : PTR;
END_VAR;
VAR
   i   : INT;
   pos : INT;
   arr : ARRAY[0..300] OF SINT;
END_VAR;
   i:=1;
   pos := 0;
   WHILE i < strLen(str := str) DO
      IF strMid(str := str, start := i, length := 1) <> " " THEN
         arr[pos]:=hexToSint(hex:=strMid(str:=str, start:=i, length:=2));
         i := i + 2;
         pos := pos + 1;
      ELSE
         i := i + 1;
      END_IF;
   END_WHILE;
   memcpy(dst:=dst, src:=ADDR(arr), len:=pos);
   ParseString:=pos;
END_FUNCTION;

// Send packet from OMS Annex N.2.3: gas meter with internal radio, security
profile B
FUNCTION SendN23;
VAR
   rc       : INT;
   len      : INT;
   tx_frame : ARRAY [1..300] OF USINT;
END_VAR;
   // Hex string with data from example N.2.3:
   len := ParseString(dst := ADDR(tx_frame), str:=
      // C
      "44"+
      // ELL
      "8c2075"+
      // AFL
      "900f002c25b30a000021924d4f2fb66e01"+
      // TPL
      "7a7500200710"+
      // Encrypted TPL/APL
```

```
      " 9058475f4bc91df878b80a1b0f98b629 "+
      // Encrypted APL
      "024aac727942bfc549233c0140829b93"
   );

   // Set sender address to address from DLL
   rc := mbusSetSenderAddress(handle:=mb, sec_addr:="1234567893153303");
   DebugFmt(message:="mbusSetSenderAddress(): \1", v1:=rc);

   rc := mbusSend(handle:=mb, frame := ADDR(tx_frame), size := len);
   DebugFmt(message:="mbusSend(): \1", v1:=rc);

END_FUNCTION;


PROGRAM sender;
// These are the local variables of the program block
VAR
   rc : INT;
END_VAR;
// The next code will only be executed once after the program starts
   rc := mbusOpen(type:=2, handle:=mb);
   DebugFmt(message:="mbusOpen(): \1", v1:=rc);

BEGIN
// Code from this point until END will be executed repeatedly

   SendN23();

   Sleep(delay:=10000);

END;
END_PROGRAM;
```

* * * * THIS PAGE IS INTENTIONALLY LEFT BLANK * * *

Tutorial

7 Tutorial

7.1 Tutorial

This tutorial is a quick crash course in the use of the RTCU IDE and the RTCU Emulator. This tutorial presents a step-by-step guide that will teach you how to create your very first program - from programming, building and all the way to configuring and transferring your program to an RTCU device. If you do not have access to an RTCU device, do not panic, as the RTCU Emulator will show you all aspects of the RTCU, although everything will happen on your screen and not in the "real world". This tutorial also assumes that you have installed the RTCU IDE and RTCU Emulator programs, and that all settings in the program are the same as when it was first installed.

The RTCU IDE and RTUC Emulator programs can be downloaded for free here:
www.logicio.com

So, lets get started with the exciting world of programming the RTCU...

Should you encounter any difficulties during the tutorial, please contact support.

7.2 Chapter 1, The problem we want to solve

Your greenhouse

You are the proud owner of a greenhouse. However, you have some trouble with managing the climate inside the greenhouse. You would like to have some sort of alarm system to alert you when the temperature either rises too high or falls too low. But wait - that is not all! You want the greenhouse to try to adjust the temperature automatically, too. You own both a heater that will raise the temperature and a motorised door that will lower the temperature inside your greenhouse. It is only when the greenhouse is unable to keep the temperature within the limits you have set forth, and it has ceased trying to do so, that you want the greenhouse to alert you. Realizing, as you are always busy, a simple audible alarm is not enough. You need to be alerted even when you are far away from the greenhouse.

This is where the RTCU comes to the rescue!

We want the RTCU device to do the following:

- Turn the heater on when the temperature falls too low.
- Open the motorised door when the temperature rises too high.
- Call you on your mobile phone when the RTCU is unable to keep the temperature within limits.

The interface between the greenhouse and the RTCU is as follows:

Digital inputs:

- `Low temperature`
- `High temperature`

Digital outputs:

- `Heater`
- `Open door`

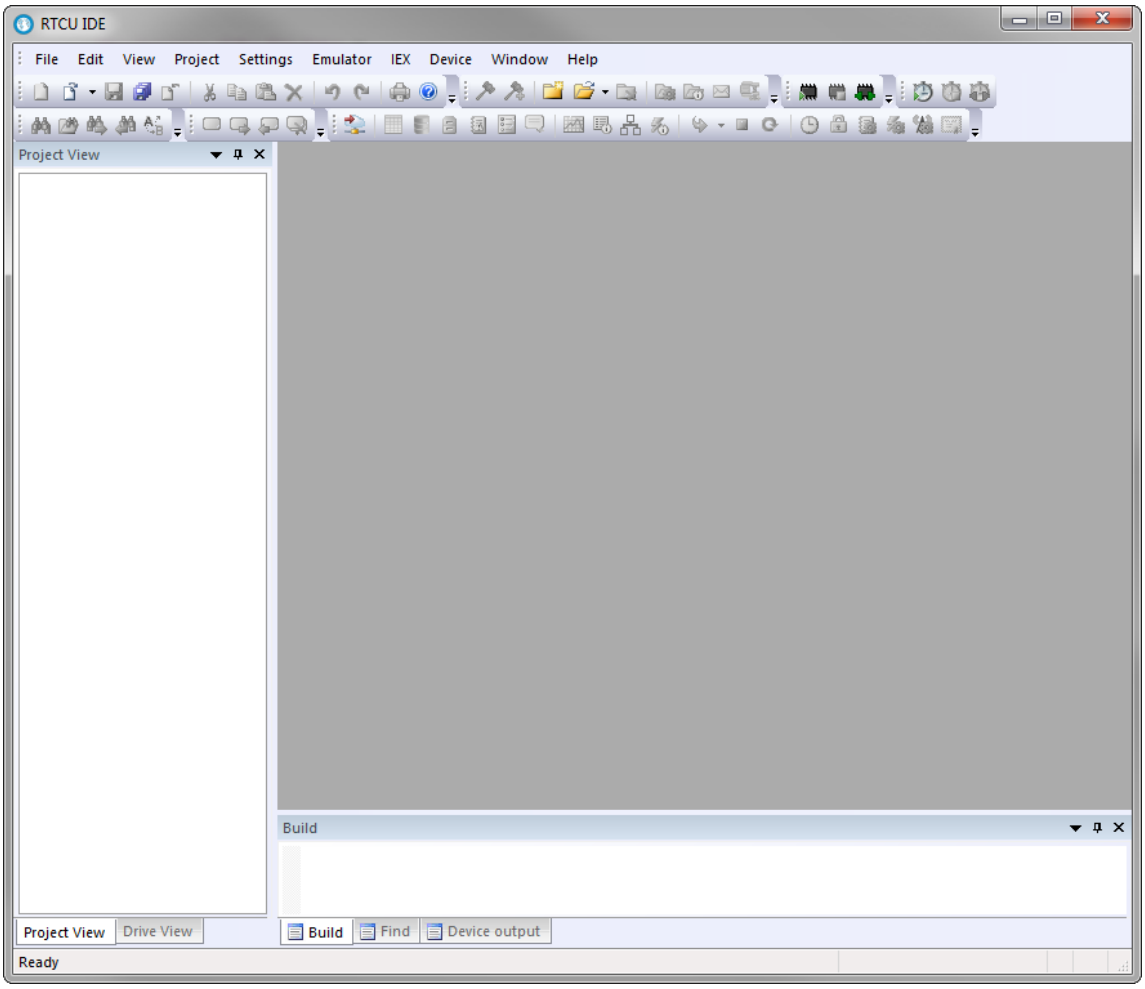
The next chapters will help you to develop this application, to test the program, and finally to transfer it to an RTCU device.

Chapter 2 Create a project

Should you encounter any difficulties during this tutorial, please contact support.

7.3 Chapter 2, Create a Project

We begin this tutorial by starting up the RTCU IDE program. Find the shortcut for RTCU IDE in the Start menu and run the RTCU IDE program. You should see something that looks like this:

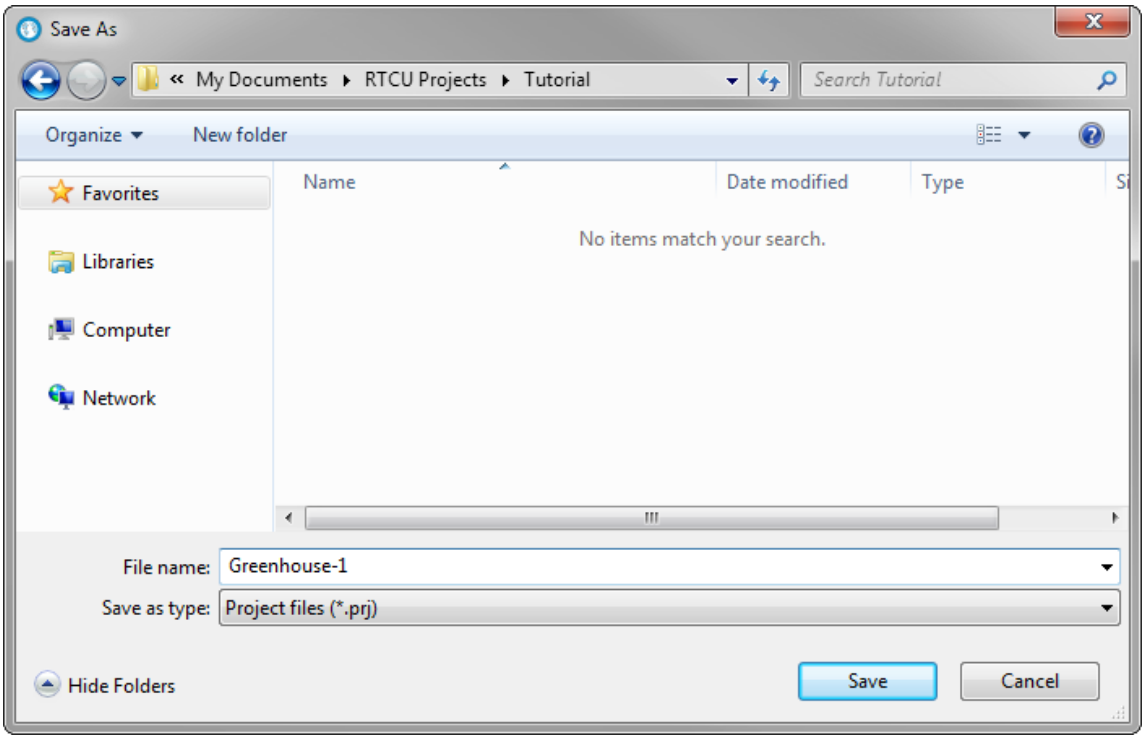


This is the RTCU IDE environment we will use throughout this tutorial. The RTCU IDE has a lot of features and functions, and some of these will be used in this tutorial, whereas some will not be needed. You will always be able to seek help in the online help system. At the end of this tutorial, you will be well-prepared to experiment and learn more about all the many features of the RTCU IDE environment.

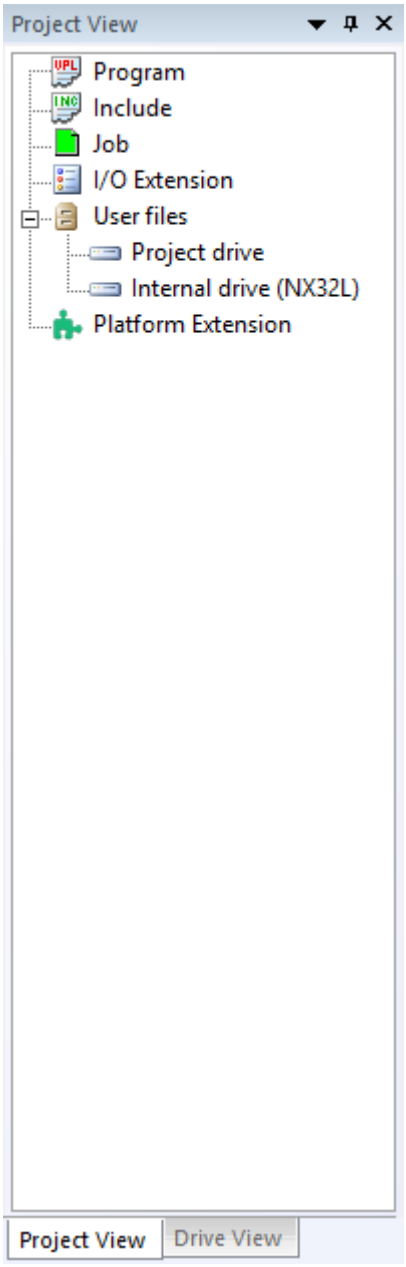
So, the RTCU IDE program is now running. We start by creating a new project. The project encapsulates all the different components of an application - namely program code, voice messages, configuration etc.

Click on the "New Project" icon on the Project toolbar to create a new project:

You will then see a dialog asking you the name of your new project. It is generally a good idea to create a new directory on your hard disk for each new project, and then keep all the files related to the project in this directory. In the following example, we create a directory named "Tutorial" (by clicking on the "New Directory" icon in the dialogue), select it by double-clicking on the name "Tutorial". We name the new project "Greenhouse-1". You should see something very similar to this:



Press the "Save" button, and the RTCU IDE project tree should look like this:



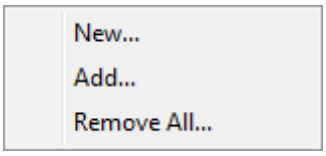
It tells you which Program files, Job files, and Voice messages the project currently consists of. In our case, the project is pretty much empty, because we have yet to add any program code etc. to it. That is next!

Chapter 3 Create the program file

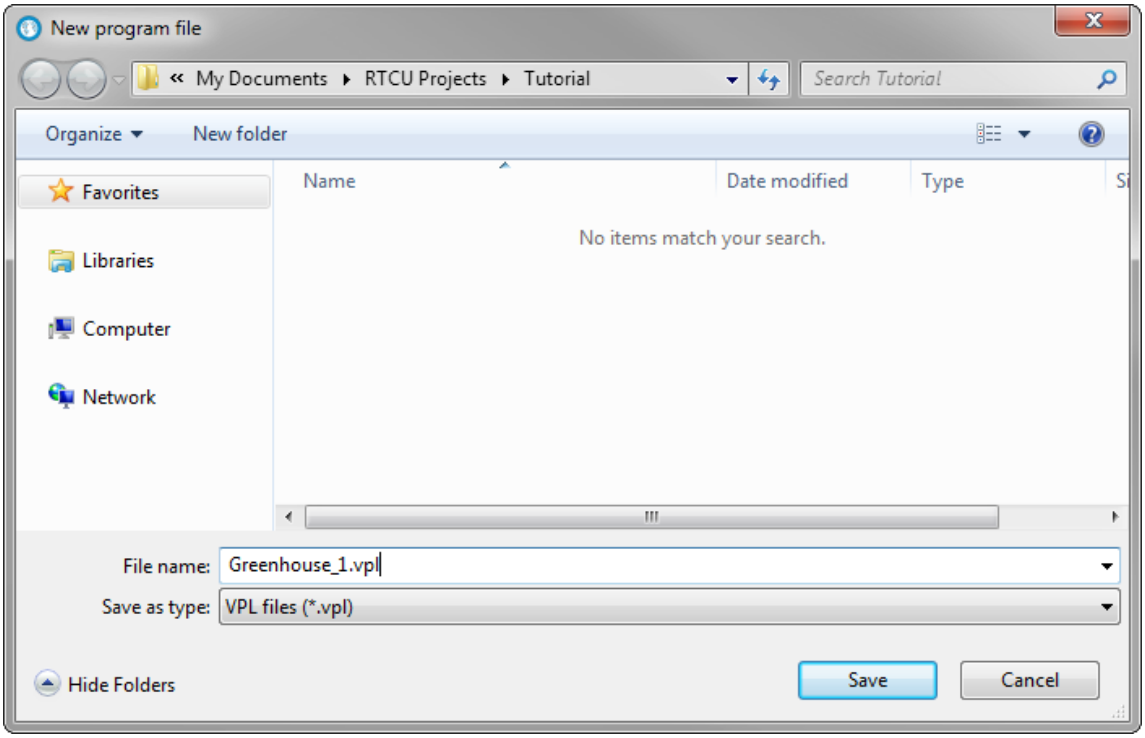
7.4 Chapter 3, Create the program file

We now have a empty project named "Greenhouse-1". Now we will add some program code to this empty project.

First, right-click on the "Program" in the project tree. This will show a little drop-down menu with three items in it:



Click on "New". Now you should see a "save dialog" like this one:



Name the new file "Greenhouse_1". This will create a file with the name "Greenhouse_1.VPL". The ".VPL" is the suffix used for VPL programs, and it is fixed in the RTCU IDE environment and cannot be changed.
The RTCU IDE has now created the new file for you. This file is open for you in the editor. In addition to this, it has created a program template. The file should look like this now:

```
//------------------------------------------------------------------------------
-
// Greenhouse_1.vpl, created 2000-12-31 14:45
//
//------------------------------------------------------------------------------
-
INCLUDE rtcu.inc

//  Input variables that can be configured via the configuration dialog
VAR_INPUT

END_VAR;
```

```
//  Output variables that can be configured via the configuration dialog
VAR_OUTPUT

END_VAR;

// The global variables of the program
VAR

END_VAR;

PROGRAM Greenhouse_1;

// The next code will only be executed once after the program starts

BEGIN
// Code from this point until END will be executed repeatedly


END;

END_PROGRAM;
```

The only difference will be the text "created xxxx-xx-xx xx:xx" in the comment field at the very top of the file. This will, of course, show the current date and time.
We will not try to explain anything about the different statements in the program, as this will be covered in great detail in the online manual, but you still might want to have a look at the file as it is.

You now have a basic program that will do absolutely nothing. This will be of great help, however, as you do not have to read a lot about syntax in VPL programs and the like. You just get a basic template, onto which you can immediately begin writing some useful code. This will be the subject of the next chapter.

Chapter 4 Write the program

7.5 Chapter 4, Write the program

In this chapter, we will add some useful code to the "Greenhouse_1.VPL" file.

To accelerate this task it is possible to mark the code in this section, select "Copy", and then use "Paste" in the RTCU IDE program editor to move it there. This will save you some work at the keyboard.

We begin by defining the various signals we need either to monitor or to control from the program:

In the file, find the following code:

```
//  Input variables that can be configured via the configuration dialog
VAR_INPUT

END_VAR;
```

Then modify it to read:

```
//  Input variables that can be configured via the configuration dialog
VAR_INPUT
   Sensor_L     : BOOL;    | Sensor input for low temperature
   Sensor_H     : BOOL;    | Sensor input for high temperature
   Alarm_time   : INT;     | time, 0..32767 minutes, time before SMS message is
sent
   phone_number : STRING; | The number the SMS message should be sent to
END_VAR;
```

This defines the two signals we need to monitor from the outside world, as well as the "Alarm_time" and "phone_number", which will be set from the configuration dialog.

Now, locate the following code in the file:

```
//  Output variables that can be configured via the configuration dialog
VAR_OUTPUT

END_VAR;
```

And change it to look like this:

```
//  Output variables that can be configured via the configuration dialog
VAR_OUTPUT
   Out_Heater      : BOOL; | Output to activate heater
   Out_Ventilation : BOOL; | Output to activate ventilation (Open door)
END_VAR;
```

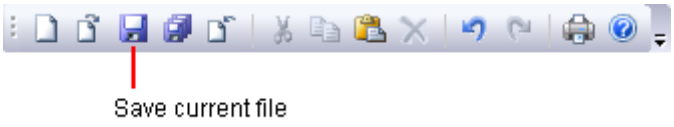
This defines the two signals we need to control in the outside world.

Your program now looks like this:

```
//-------------------------------------------------------------------------
-
// Greenhouse_1.vpl, created 2000-12-31 14:45
//
//-------------------------------------------------------------------------
-
INCLUDE rtcu.inc

//  Input variables that can be configured via the configuration dialog
VAR_INPUT
   Sensor_L     : BOOL;   | Sensor input for low temperature
   Sensor_H     : BOOL;   | Sensor input for high temperature
   Alarm_time   : INT;    | time, 0..32767 minutes, time before SMS message is
sent
   phone_number : STRING; | The number the SMS message should be sent to
END_VAR;

//  Output variables that can be configured via the configuration dialog
VAR_OUTPUT
   Out_Heater      : BOOL; | Output to activate heater
   Out_Ventilation : BOOL; | Output to activate ventilation
END_VAR;

// The global variables of the program
VAR

END_VAR;


PROGRAM Greenhouse_1;

// The next code will only be executed once after the program starts

BEGIN
// Code from this point until END will be executed repeatedly


END;

END_PROGRAM;
```

Please note that we have not in any way connected the four signals to any specific physical in- or outputs, we have not even written anything about whether they are digital in- or output signals at all. This will be done in the Configuration, which will be covered in Chapter 6.

To save your work, you can press the "Save current file" on the standard toolbar:



Now we will add the variables of the program.
Locate the following:

```
// The global variables of the program
VAR

END_VAR;
```

And modify it to read:

```
// The global variables of the program
VAR
   timer      : TON;    // ON-delay timer is declared.
                        // A On-delay timer goes active when 'trig' has
                        // been active for 'pt' seconds.
   send_alarm : R_TRIG; // detects leading edge on alarm
END_VAR;
```

Now it is time to add some code to the program.
First we will add some initialisation code.

Locate the following:

```
// The next code will only be executed once after the program starts

BEGIN
```

And modify it to read:

```
// The next code will only be executed once after the program starts
// Set the timer
// (Convert to seconds)
timer(pt := Alarm_time*60*1000);

// turn on GSM-module
gsmPower(power := ON);

BEGIN
```

Now we will add the rest of the program.
Locate the following:

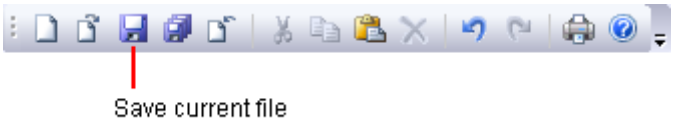
```
BEGIN
// Code from this point until END will be executed repeatedly


END;

END_PROGRAM;
```

And modify it to read:

```
BEGIN
   // Code from this point until END will be executed repeatedly
   // Update timer
   timer();

   // If the temperature is to high then activate ventilation:
   IF Sensor_H THEN
      Out_Ventilation:= ON;
```

```
   ELSE
      Out_Ventilation:= OFF;
   END_IF;

   // If the temperature is to low then activate the heater:
   IF Sensor_L THEN
      Out_Heater:= ON;
   ELSE
      Out_Heater:= OFF;
   END_IF;

   // Start timer on the leading edge of the sensor-inputs:
   timer(trig:= Sensor_L OR Sensor_H);

   // Detect leading edge on alarm:
   send_alarm(trig:=timer.q);

   IF send_alarm.q THEN
      IF Sensor_L THEN
         // send sms for temperature to low
         gsmSendSMS(phonenumber:=phone_number, message:="Temperature too
low");
      ELSIF Sensor_H THEN
         // send sms for temperature to high
         gsmSendSMS(phonenumber:=phone_number, message:="Temperature too
high");
      END_IF;
   END_IF;

END;

END_PROGRAM;
```

To save your work, you can press the "Save current file" on the standard toolbar:



Now your complete program looks like this:

```
//-------------------------------------------------------------------------
---
// Greenhouse_1.vpl, created 2000-12-31 14:45
//
//-------------------------------------------------------------------------
---
INCLUDE rtcu.inc

//  Input variables that can be configured via the configuration dialog
VAR_INPUT
   Sensor_L     : BOOL;    | Sensor input for low temperature
   Sensor_H     : BOOL;    | Sensor input for high temperature
   Alarm_time   : INT;     | time, 0..32767 minutes, time before SMS message is
sent
   phone_number : STRING; | The number the SMS message should be sent to
```

```
END_VAR;

//  Output variables that can be configured via the configuration dialog
VAR_OUTPUT
   Out_Heater      : BOOL; | Output to activate heater
   Out_Ventilation : BOOL; | Output to activate ventilation
END_VAR;

// The global variables of the program
VAR
   timer      : TON;    // ON-delay timer is declared.
                        // A On-delay timer goes active when 'trig' has
                        // been active for 'pt' seconds.
   send_alarm : R_TRIG; // detects leading edge on alarm
END_VAR;

PROGRAM Greenhouse_1;

// The next code will only be executed once after the program starts
// Set the timer
// (Convert to seconds)
timer(pt := Alarm_time*60*1000);

// turn on GSM-module
gsmPower(power := ON);

BEGIN
// Code from this point until END will be executed repeatedly
   // Update timer
   timer();

   // If the temperature is to high then activate ventilation:
   IF Sensor_H THEN
      Out_Ventilation:= ON;
   ELSE
      Out_Ventilation:= OFF;
   END_IF;

   // If the temperature is to low then activate the heater:
   IF Sensor_L THEN
      Out_Heater:= ON;
   ELSE
      Out_Heater:= OFF;
   END_IF;

   // Start timer on the leading edge of the sensor-inputs:
   timer(trig:= Sensor_L OR Sensor_H);

   // Detect leading edge on alarm:
   send_alarm(trig:=timer.q);

   IF send_alarm.q THEN
      IF Sensor_L THEN
         // send sms for temperature to low
         gsmSendSMS(phonenumber:=phone_number, message:="Temperature too
low");
      ELSIF Sensor_H THEN
         // send sms for temperature to high
         gsmSendSMS(phonenumber:=phone_number, message:="Temperature too
high");
      END_IF;
   END_IF;

END;
```

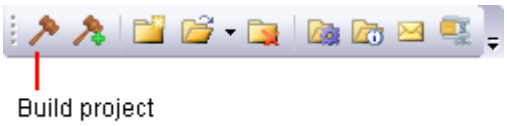
```
END_PROGRAM;
```

Congratulations! You have already come a long way, but now it is off to:

Chapter 5 Build the project

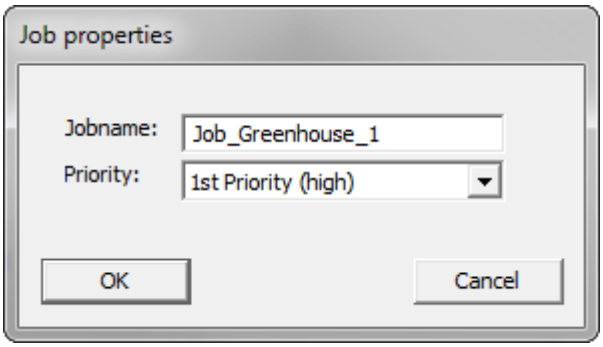
7.6 Chapter 5, Build the Project

Now the time has come to build the project. During the build process, all the parts of a project will be translated to a format that the RTCU device can understand and execute. The build step is a very simple step, because the only thing there is to do is to press "Build Project" on the Project toolbar:



When you press the "Build Project" button, all VPL files are translated and examined for errors, and the right files are built, so that the project can be transferred to an RTCU device or executed in the RTCU Emulator.

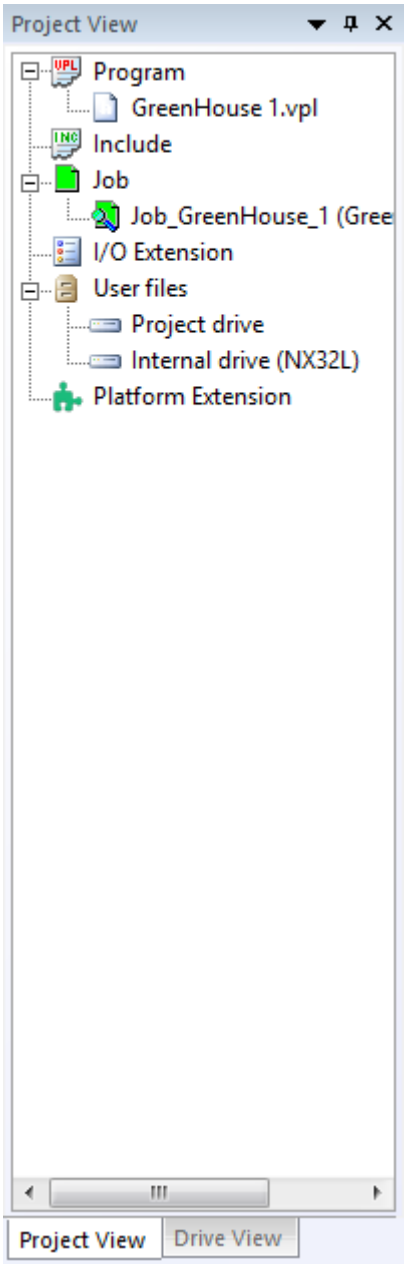
If you have written the program as you were meant to, you will see the following message:



This message pops up, because the RTCU IDE program has detected that you have added a new program file, but there is currently no Job defined that uses this program file. Therefore, the RTCU IDE suggests both Jobname and Priority for this new Job.
Just click on the "OK" button to accept the default names and values. You should then see the following output in the build window:

This means that you have built the project, and that no errors were found.

The project tree in the left part of the RTCU IDE program now looks like this:



And as you can see, there is now one Job defined - "Job_Greenhouse_1". This is what actually will be executed on the RTCU platform.
Where to go from here? Do we have a complete project that is ready to be transferred to the RTCU platform?
As a matter of fact, yes, but nothing much will come from it. If you remember, we did not specify where the two in- and output signals should come from.

So, if we transfer the project to the RTCU device, and watch the results, what will happen? Even if we watch very closely, nothing will be visible to us - no outputs will change and so on. This is because we need to configure the Job we have defined in the project.

As you go on with the online help, you will eventually come to know that we can have more than one Job for a project. You can even define more Jobs using the same program file (".VPL" file), and this is where the configuration process is really useful. Well, there is actually many other reasons to have the configuration process - more on that in the online help.

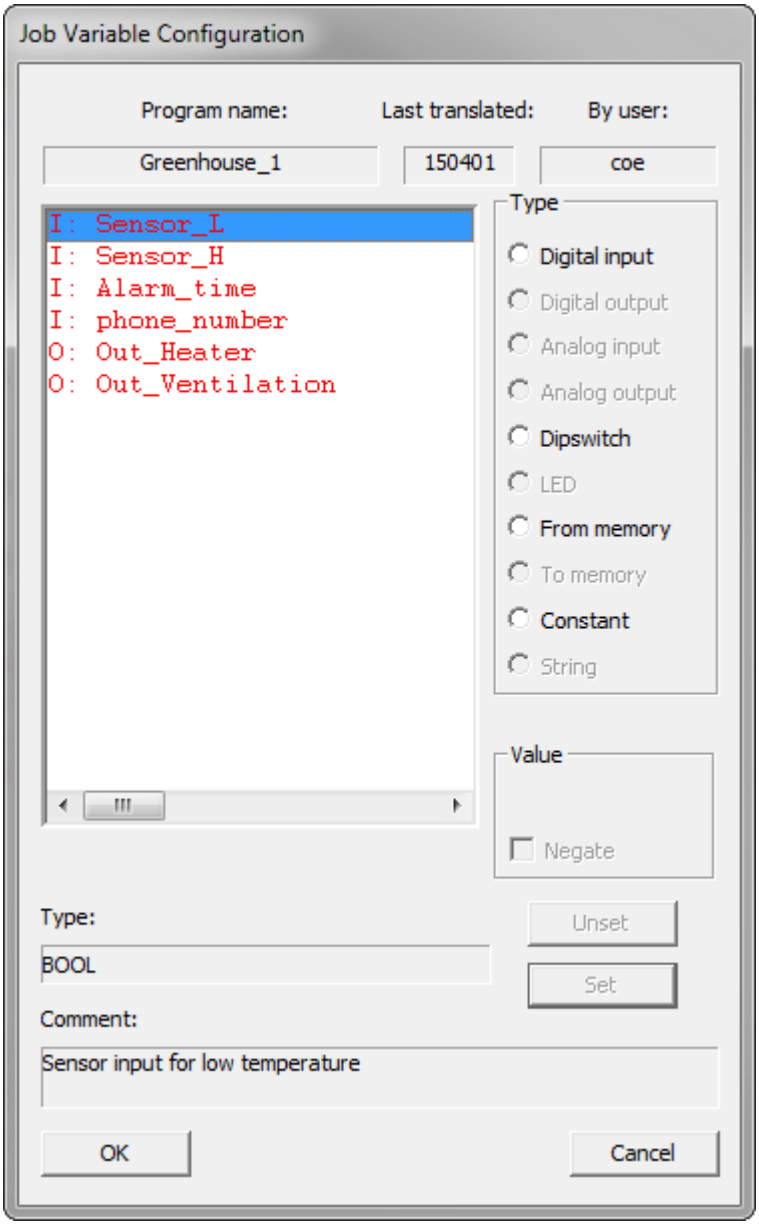
This is why it is now time for:

Chapter 6 Configure the project

7.7 Chapter 6, Configure the Project

Now the time has come to do the configuration of the project. In the configuration process, all variables defined in the VAR_INPUT or VAR_OUTPUT sections of your program will be assigned to in- or outputs, numerical values, strings, and voice messages. During program development, you do not need to think about which specific in- or outputs to use, because you delay that decision until the configuration process.

Now let us configure the Greenhouse project:
First, double-click on the Job "Job_Greenhouse_1" in the project tree on the left part of the screen. This should bring up the configuration dialogue:

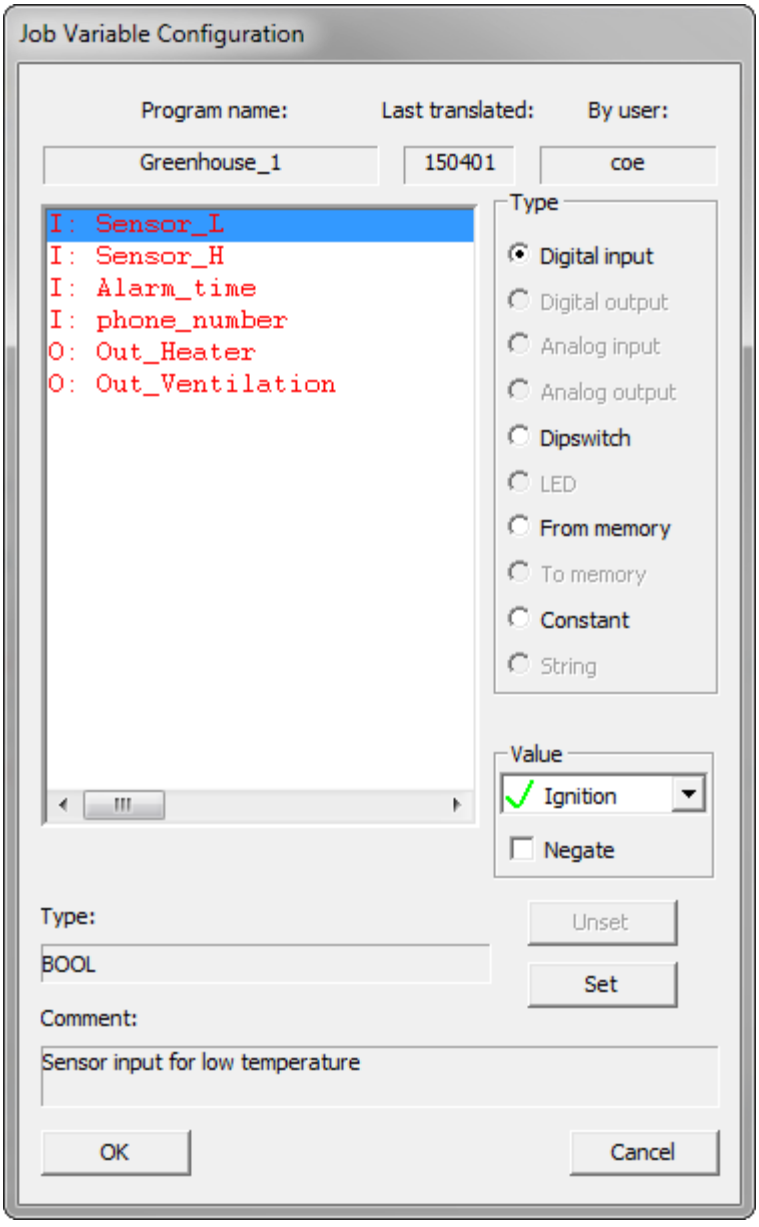


This is the configuration dialogue. The list to the left shows you all the variables defined in the VAR_INPUT and VAR_OUTPUT sections. The variables with an "I:" in front of them are inputs, and the variables with an "O:" in front of them are outputs. When you select one of the variables, the two fields at the bottom of the dialogue show you the type and comment field of the variable (the text written with orange in the editor when you enter the program). To the right, you see a list

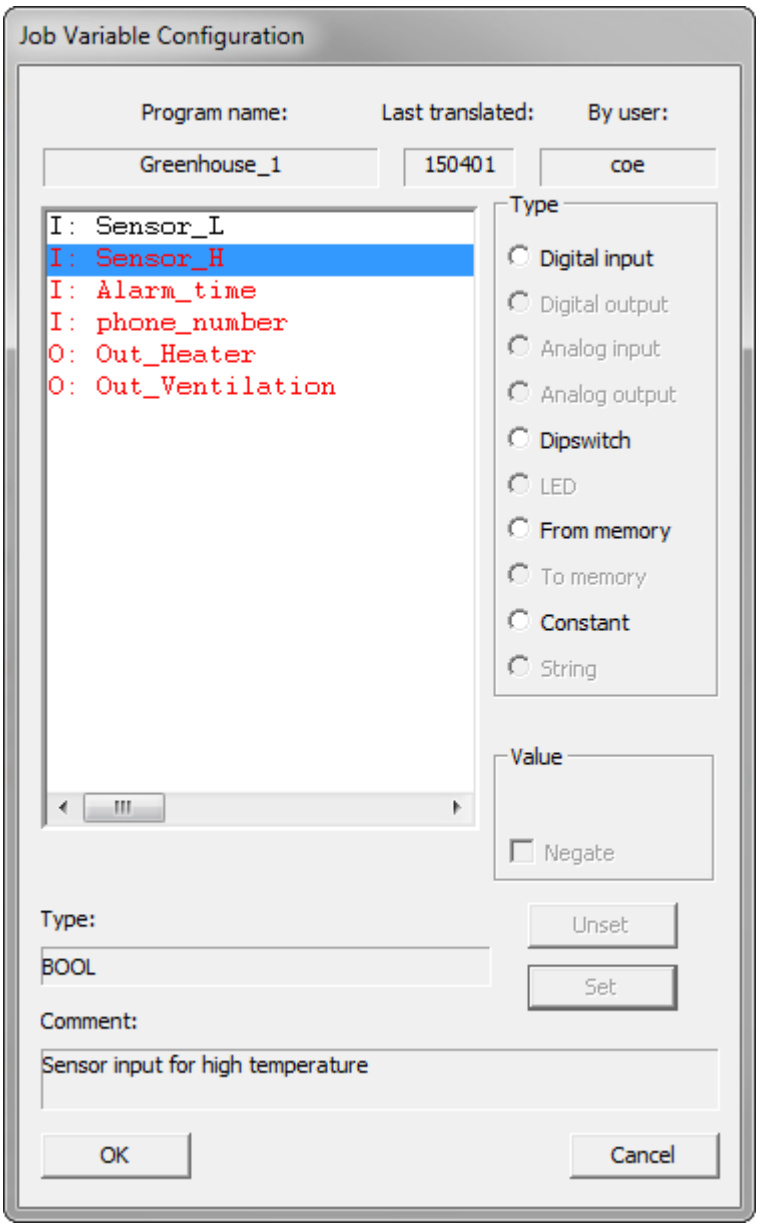
of the different types you can assign the variables to. Some are grayed out, which means they are not active for this particular type of variable.

So, let us assign something to our variables.

First, make sure the uppermost line is selected - Sensor_L.
Next, click on "Digital input". Now the configuration dialogue looks like this:

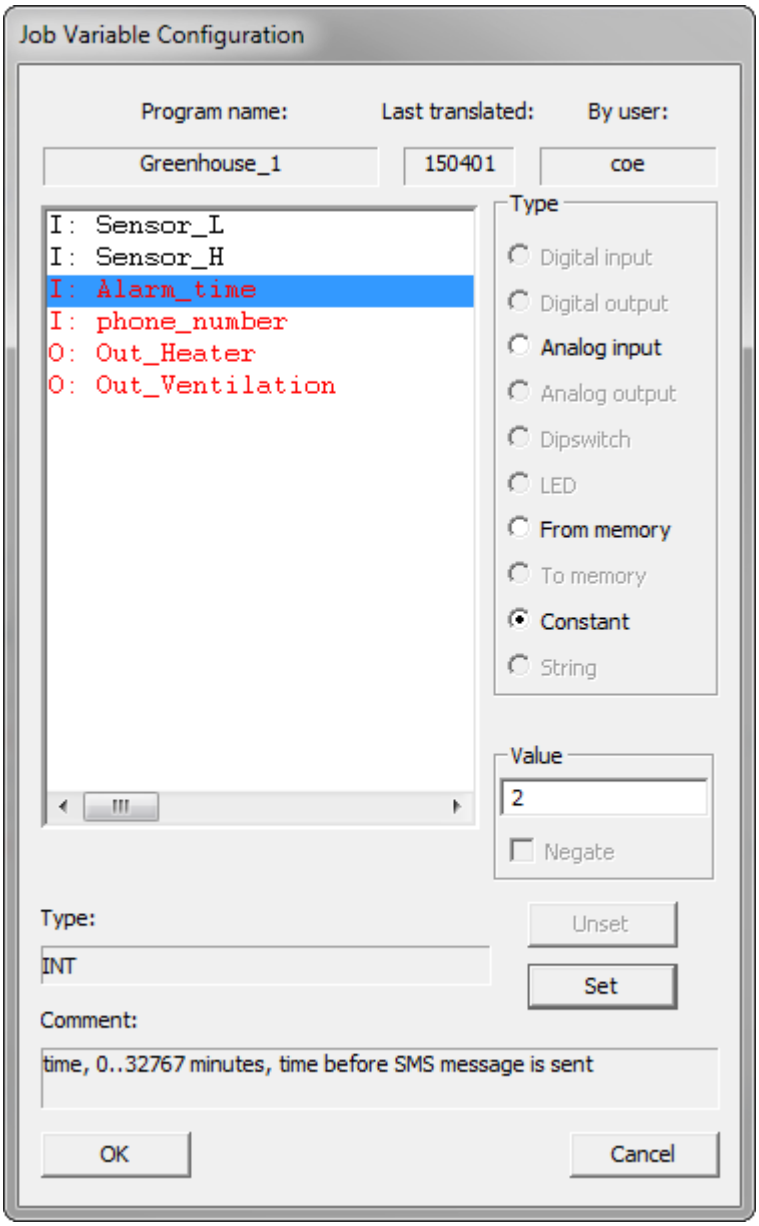


The next free item in the "Item" group is, in this case: "Ignition". It will be selected automatically. Select "Input 1" instead and after that, you want to click the "Set" button. The assignment is not done before you click the "Set" button. After you click the "Set" button, the next non-configured item in the list will automatically be selected. After you click the "Set" button, the configuration dialogue looks like this:

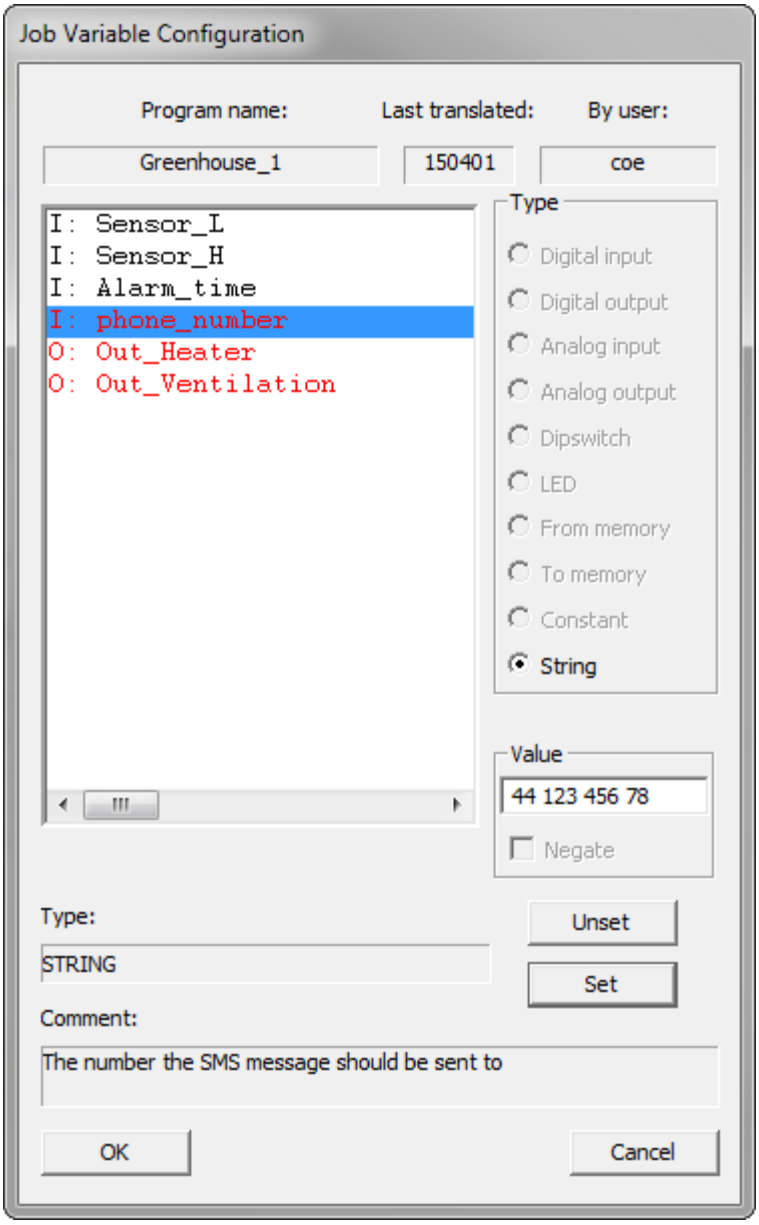


Then click on the "Digital input" and select the "Input 2" item. Press the "Set" button.

The "Alarm_time" is next. This variable must be assigned a value. Press the "Constant" type, and then write the number "2" in the entry field in the "Item" group box. The dialogue now looks like this:



Press the "Set" button. Now the "phone_number" variable will be selected from the list. This is a STRING data type that holds the phone number of your GSM mobile phone. This is the number of the phone the program will send SMS text messages to, when the greenhouse is unable keep the temperature within limits. Click on "String". Type the number in the entry field in the "Item" group box. The dialogue now looks like this:



Press the "Set" button.

Now you probably have a firm grip on the configuration process, so we will quickly move over the last two items that need to be configured.

The "Out_Heater" variable is selected. Select "Digital output" and accept the "Output 1" item shown. Press the "Set" button.
Finally, the "Out_Ventilation" variable is selected. Select "Digital output" and accept the "Output 2" item shown. Press the "Set" button.

All the configurable variables of the program are now configured. You may have noticed that when a variable is configured, it will change its color from red to black. When you press the "OK" button, the configuration data is saved for the project.

Press the "OK" button.

Now you have a complete project that is ready to be transferred to an RTCU device, or tested in the RTCU Emulator.
And testing in the emulator is just what is next.

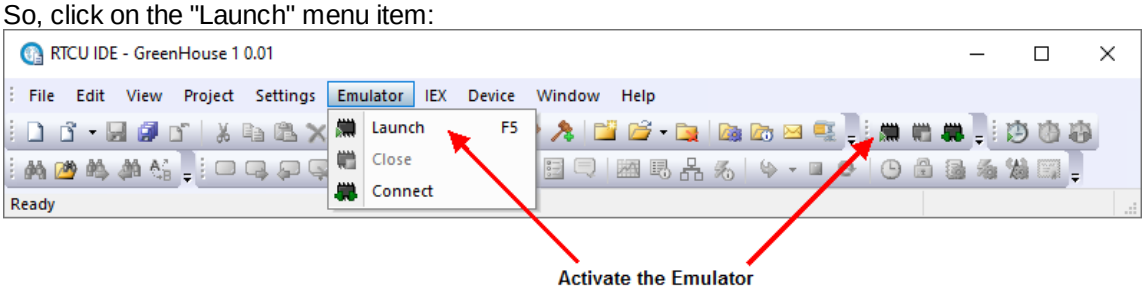
Chapter 7 Run the project in the RTCU Emulator

This is a good place if you want to take a break from the tutorial and come back later.

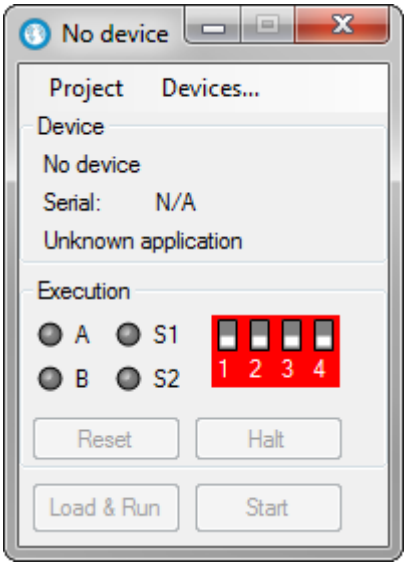
7.8 Chapter 7, Run the project in the RTCU Emulator

Now you have developed the Greenhouse project and are probably wondering whether it works or not. Well, this is the right chapter for you. In this chapter, we will test the program, and the RTCU Emulator will save you time, as you will not have to fix errors after you have transferred your program to an RTCU device. In the emulator, the "transfer time" is reduced to sub-seconds, and you have all the different hardware resources available on your screen, which greatly simplifies the process of testing your program - a process that would involve switches, wires and the like in the real world.

So, click on the "Launch" menu item:

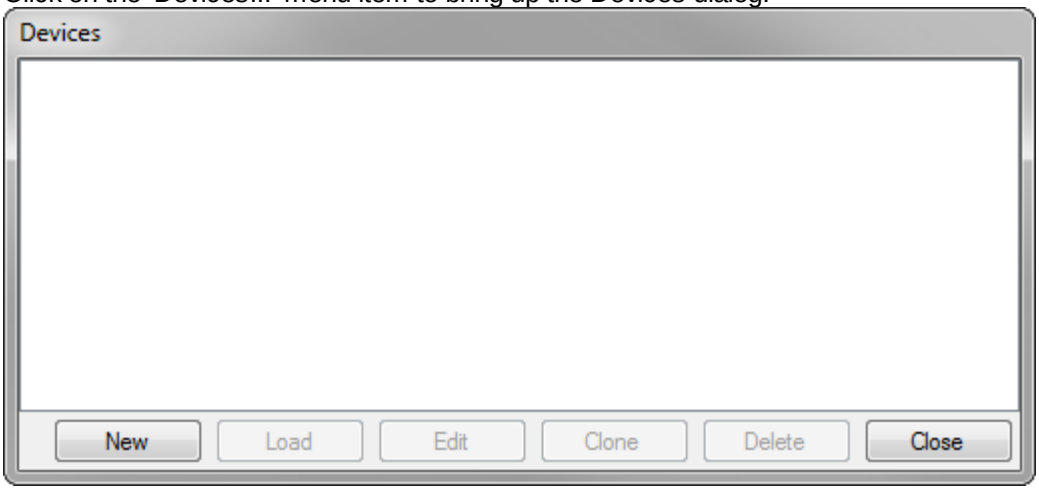


This will launch the RTCU Emulator and bring up the main window:



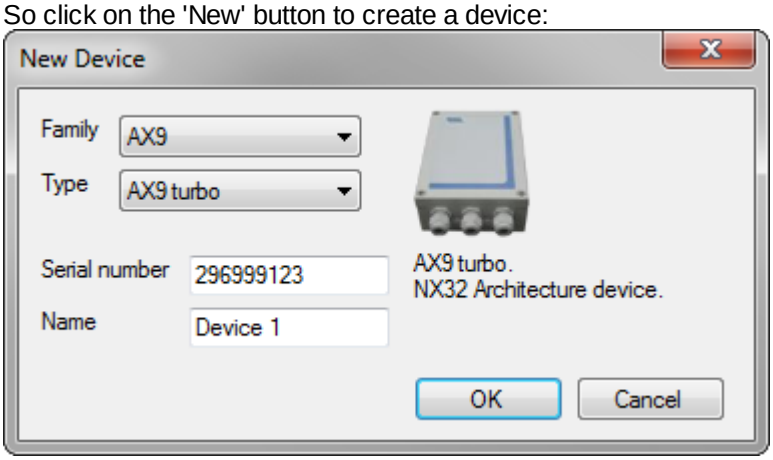
The first time the RTCU Emulator is launched there is no virtual devices present and the first device must therefore be created.

Click on the 'Devices...' menu item to bring up the Devices dialog:



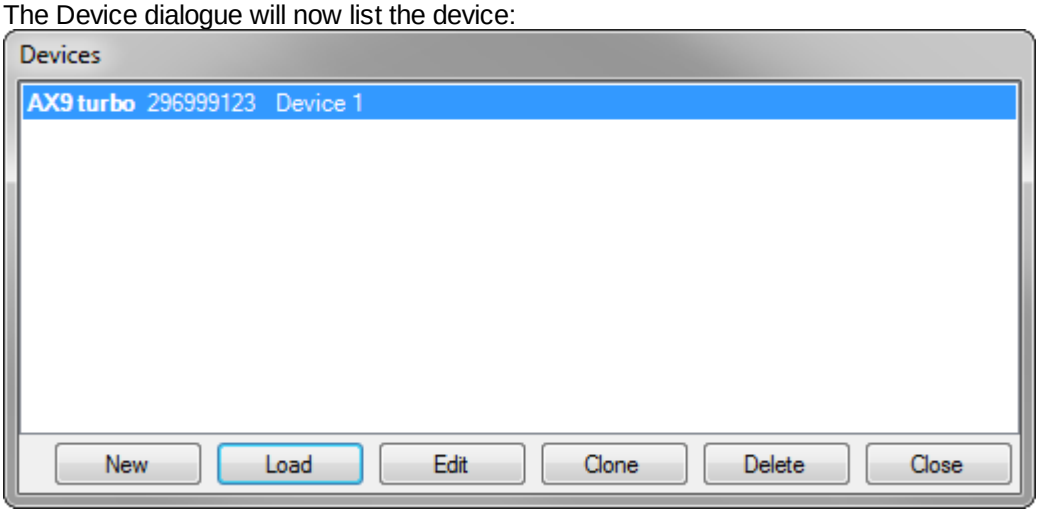
The first time the dialog is opened, there are no devices defined and the list will be empty.

So click on the 'New' button to create a device:



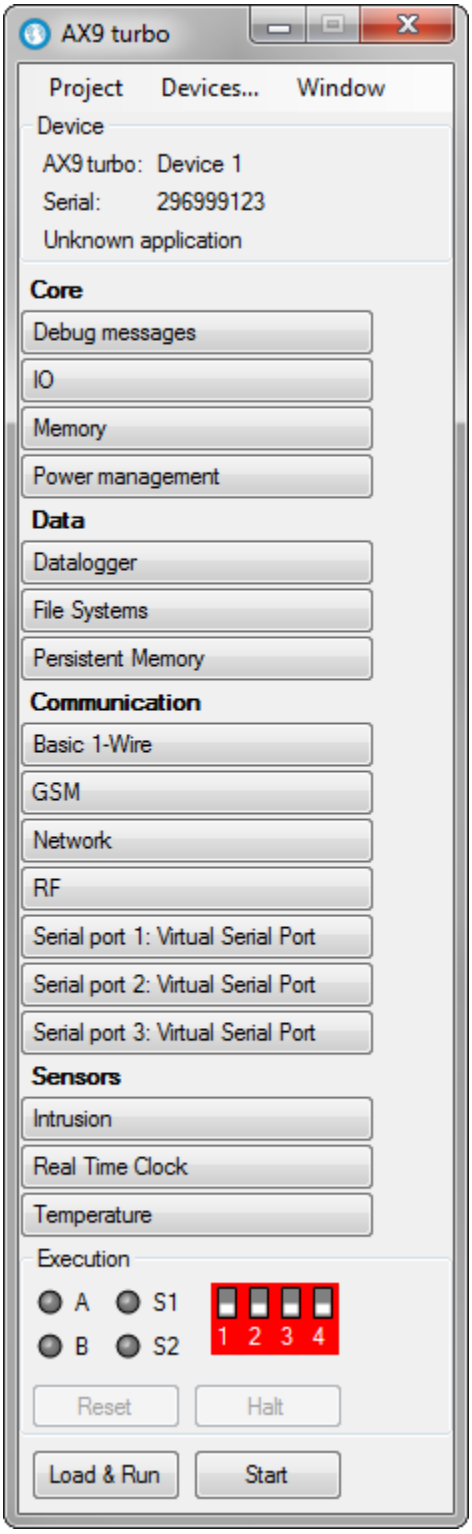
Click on the Family drop down and select the 'AX9' option, then click on the 'OK' button.

The Device dialogue will now list the device:



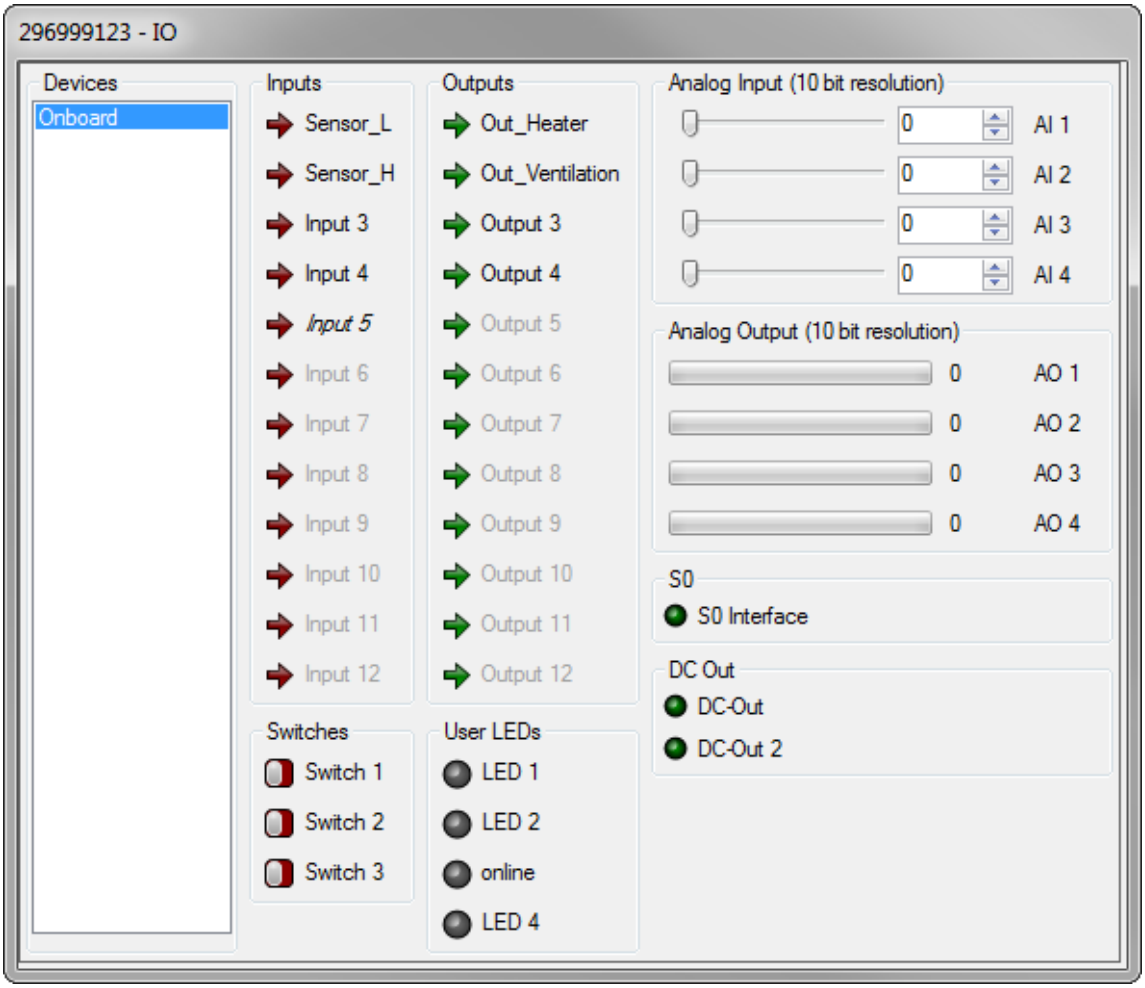
Select the newly created device and click on the 'Load' button.

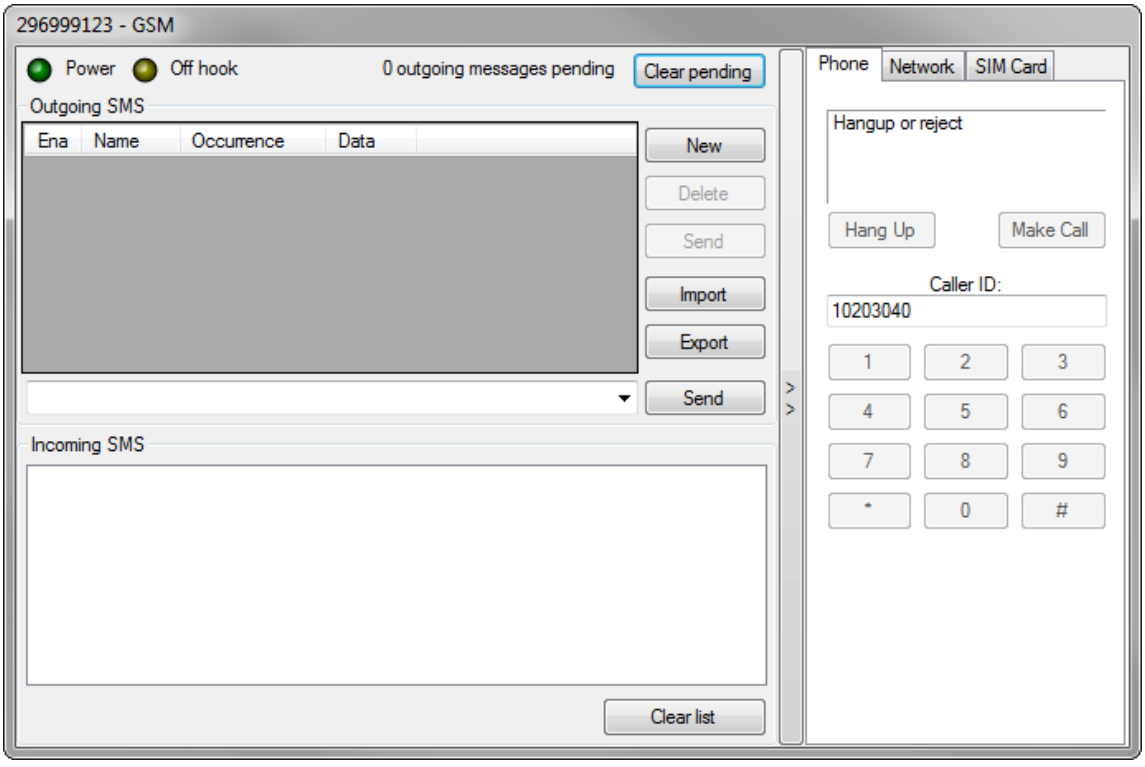
The main window of the RTCU Emulator will now look like this:



With the main window, you can now enable the different control windows you want to use for testing purposes. Click on the "IO" to enable the I/O control window, and then click on the "GSM" to enable the GSM control window.

You now see these two windows:



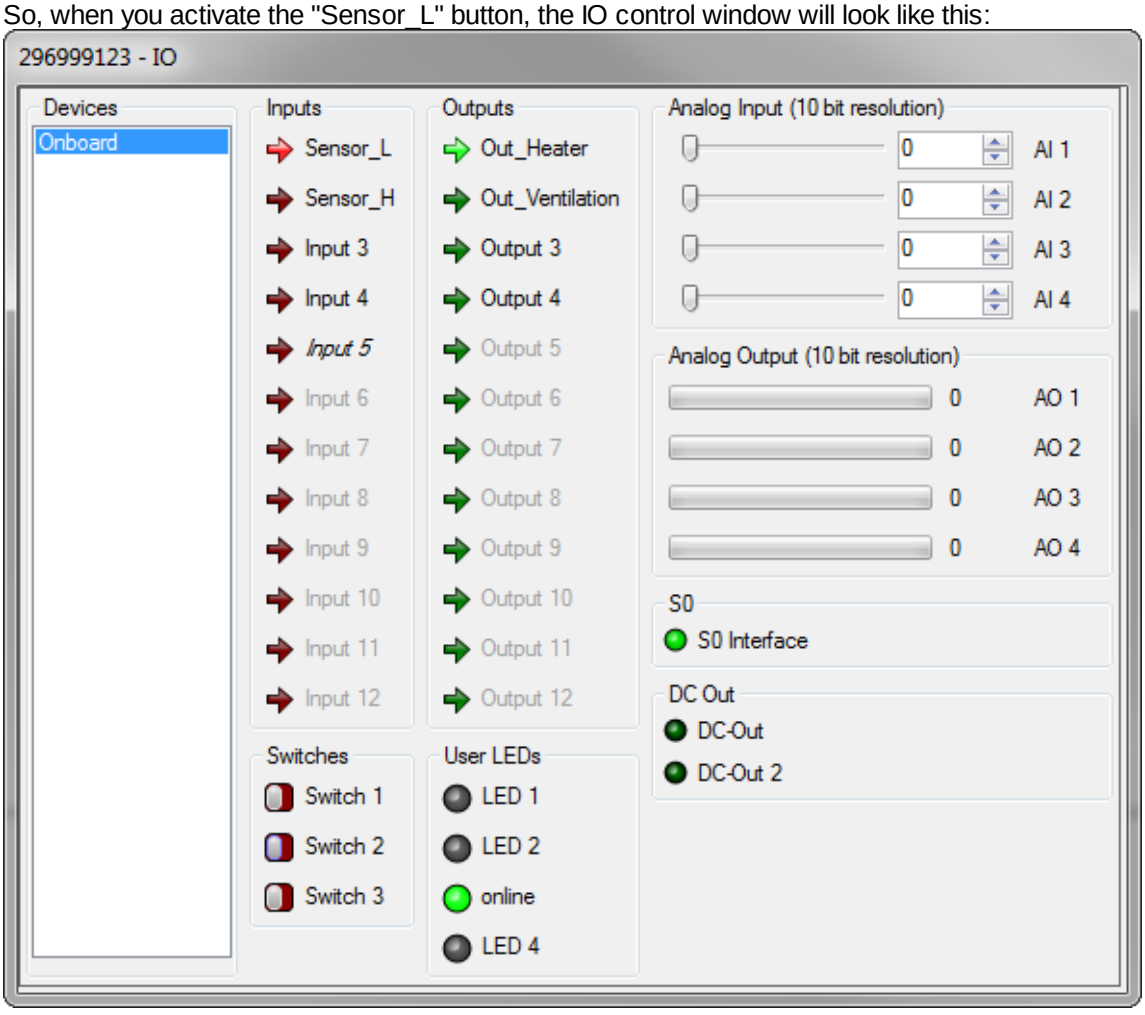


Please note that the names of the four Digital I/O signals we have defined are shown in the IO control window. This makes it easier to identify which I/O signal is assigned to which variable in the program.

Now we can load and execute our program by pressing the "Load & Run" button on the main window.
Now your program is actually running!

If you click on the arrow button at the "Sensor_L" digital input, you will see that the digital output "Out_Heater" will turn on. And when you turn off the "Sensor_L" digital input, the "Out_Heater" will also be deactivated. The same applies to input "Sensor_H" and output "Out_Ventilation".

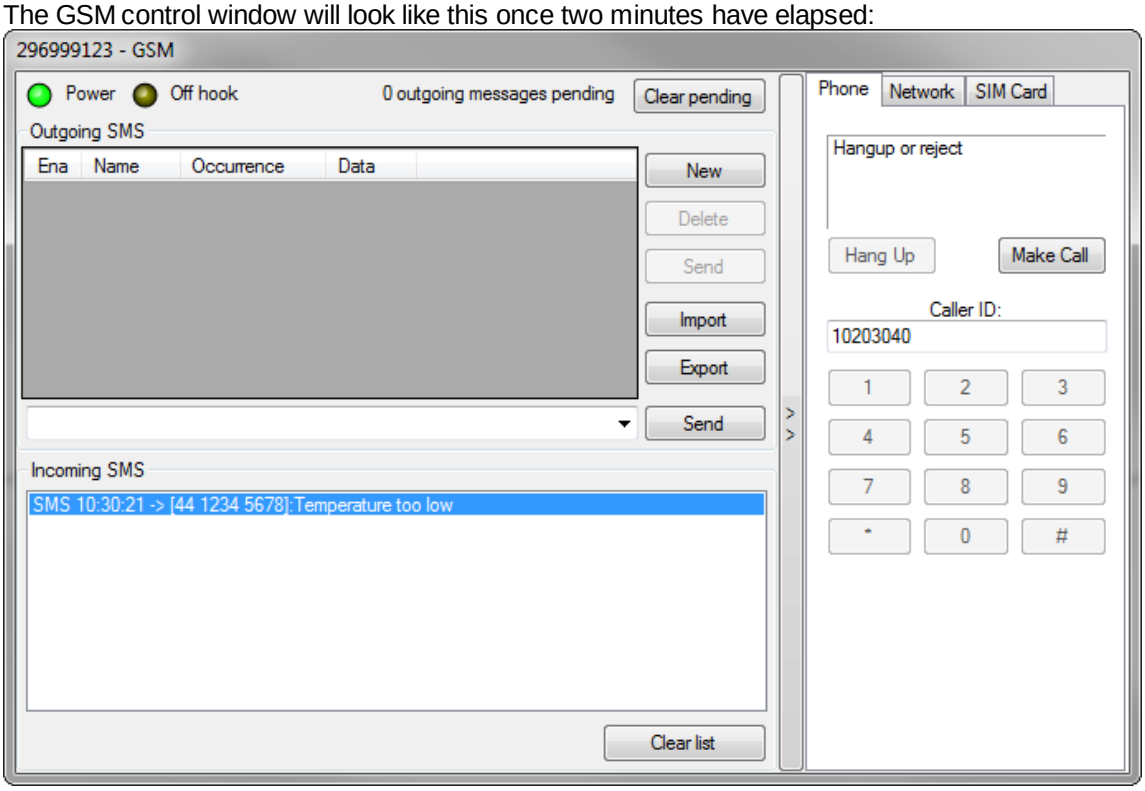
So, when you activate the "Sensor_L" button, the IO control window will look like this:



Try to leave the "Sensor_L" input on for more than two minutes and see if you can guess what will happen.

If you guessed the following, you are right!

The GSM control window will look like this once two minutes have elapsed:



You received an SMS message, because the "Sensor_L" was active for more than two minutes!

It is now up to you to figure out what will happen if the "Sensor_H" input is active for more than 2 minutes...

This was a simple session about using the RTCU Emulator. There are many more aspects to cover, but these are covered in the online help system.

This was merely emulation. However, if you have an RTCU device, you can now proceed to:

Chapter 8 Transfer the project to a RTCU device

This is a good place to take a break from the tutorial and come back later.

Should you encounter any difficulties during this tutorial, please contact support.

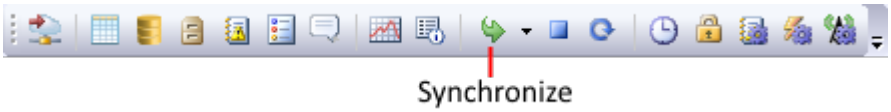
7.9 Chapter 8, Transfer the Project to a RTCU Device

At this point, you have tested your application in the RTCU Emulator. If you also want to transfer your project to a physical RTCU device, this chapter is the right one for you.

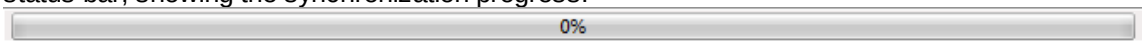
To transfer a project to an RTCU device, you need a programming cable. This cable will connect your PC to the RTCU device. The exact position of the programming connector depends on the type of RTCU device that you possess. Please consult the technical documentation for your particular type of RTCU in order to locate the connector.

The connection status in the status bar will change to  when the connection to the RTCU device has been established, and you are ready to proceed.

Now click on the "Synchronize" button on the project toolbar:



This will begin synchronizing the project with the RTCU Device, and show a progress bar on the status bar, showing the synchronization progress.



When it reaches 100%, and writes "Transfer done." or "Intellisync completed." in the Build window, everything went well.
The project has now actually been transferred to the RTCU device. A few moments later (~10 seconds), the program should be up and running. From here, it will take a few seconds more for the GSM module to actually log onto a GSM base station. If you activate one of the two sensor inputs, you will notice that one of the digital outputs will activate, and if you leave one of the inputs on for more than 2 minutes, you will receive an SMS message on your mobile phone just like in the RTCU Emulator. This presumes, of course, that the RTCU has a SIM card installed, and is connected to a suitable GSM antenna. Please take a look at the technical documentation for the RTCU device if you are unsure about this being the case.
The procedure above will also work when connected to a remote RTCU device using either a data call (CSD) or through an connection.

To use other methods for transferring a project to an RTCU Device, please see the documentation for the Project: Transfer menu.

That is it. Now go on with:

Chapter 9 Where to go from here?

7.10 Chapter 9, Where to go from here ?

So, this completes the tutorial. For further information, you should consult the section RTCU IDE Development environment in the online help manual. You can also have a look at some of the examples in the online help. These will be of great help in understanding the VPL programming environment and the many features of the RTCU devices.

If you stayed with us this far, you truly deserve a break!

Troubleshooting

8 Troubleshooting

8.1 Troubleshooting

These are some of the common pitfalls encountered when using the RTCU IDE program and the VPL programming language.

RTCU IDE Integrated Development Environment:

1) I get errors when I try to upload my project to my RTCU device.

There can be several reasons why this happens. Usually there is a problem with the serial port (COM). Either it is not the COM port that is specified in the Setting menu, or if you are using a laptop PC the serial port may sometimes disable itself to save power. Alternatively it could be because the port is configured for IRDA (infrared) communication.
Also check that your programming cable is intact and firmly secured both to the PC and the RTCU device. Please also make sure that the RTCU is powered and is running.

2) Why can I not send SMS messages and make voice calls by using the RTCU device?

You must make sure that the RTCU device you are using has the GSM module installed. You must also make sure that you have a valid SIM card installed in the SIM card reader of the RTCU device. The RTCU device needs an external antenna, and this must be an antenna that matches the requirements listed in the technical documentation for your actual RTCU device. If this does not help, try the SIM card in a mobile phone and make sure that the mobile phone can connect to a base station and that you can actually make/receive calls by using the SIM card.

A common error is to forget to turn on power to the built-in GSM module by using the gsmPower function. Please also note that if a PIN code is required for your SIM card, this code must be set on the Device-> Configuration -> GSM Parameters -> PIN code page. The GSM module will always be switched on when the program starts. If needed, it can then be switched off and on via the program and gsmPower functions.

You must also make sure that the RTCU device is connected successfully to a GSM base station. A common error is to switch on the power to the GSM module (by using gsmPower) and then immediately try to send an SMS or make a voice call. After the gsmPower function, a number of seconds must elapse before the GSM module is actually connected with a GSM base station. This amount of time depends on the type of GSM network as well as the general traffic on the GSM net. A safe method to check for this is to monitor the gsmConnected function. When the gsmConnected function returns true, the GSM module is connected successfully to a GSM base station.

Another possible error, although seldom seen, is if you use a 5 Volts SIM card. The RTCU devices will only work with 3 Volts SIM cards. Some older SIM cards exist that are only able to work at 5 Volts so please make sure that your card can work at 3 Volts.This is also a problem that ordinary mobile phones today face and is therefore not specific to the RTCU devices.

3) I get an error when I try to register the RTCU IDE program by using "Send via Email".

This is usually because you do not have a "MAPI"-compatible mail program installed, or it may be that it is not configured properly for the RTCU IDE to use it.

4) I get an error when I try to send my project by using the "Send Project via Email".

This is usually because you do not have a "MAPI"-compatible mail program installed, or it may be that it is not configured properly for the RTCU IDE to use it.

5) I got an email saying that a new firmware version is available. What do I do?

It all depends on the message in the email. When we make new features available, we will normally make a new version available of the RTCU firmware for you to download. The message in the email will tell you in detail what the new features are, and under what circumstances it would be beneficial for you to upgrade your RTCU devices with the new firmware version. The email will also tell you step-by-step how to upload the new firmware to your RTCU devices. Please note, however, that it is very rarely that you will need to upgrade the firmware. Generally most of the new features will be available in the RTCU IDE, which is available for download from www.logicio.com

6) I have uploaded my project to the RTCU device. While doing so, I lost communication with the device.

In this case, it is possible that the RTCU device will be left in a state where it is not able to execute the RTCU program. Normally you just upload the project again and everything will then be fine. If, for some reason, you are unable upload your project to the RTCU device after such an error, you can always activate the recovery switch present on all RTCU devices. This will prevent any execution of the RTCU program, and then you can upload your project again.

7) When I send an SMS message by using gsmSendSMS, it will not always be delivered.

This error can have several causes. This can happen when the GSM network is busy - for example not enough signal strength. Another potential problem can be if the GSM provider delivers an acknowledgment every time someone sends an SMS message by using their net. Some providers allow you to suppress this acknowledgment by using some specific prefix before the SMS message text. This extra acknowledgment message can sometimes interfere with the transmission of SMS messages from the RTCU device. To overcome this, always check the return code from the gsmSendSMS() function call. This will tell you whether the transmission was all right or not. If you got an error, you can send the message again. Typically, these errors will resolve themselves very quickly and will only last a few seconds.

8) I am not able to receive SMS messages on the RTCU from certain mobile phones/numbers.

The SIM card used in the RTCU device must have its phone book cleared. If the sender of an SMS message exists in the phone book of the SIM card in the RTCU, all SMS messages from this number are discarded. A simple solution is to make sure that the SIM card used on the RTCU device is empty.

9) When I try to use gsmIncomingPDU, it always returns 3 in the status field.

When using the gsmIncomingPDU(), you must set the "message" field to the address of the buffer you want the message to arrive in before the instance of the gsmIncomingPDU() is called (this will be called either explicit by writing "name()" where "name" is of type gsmIncomingPDU() or when the BEGIN statement is executed.)

10) Transferring a large amount of data from the RTCU takes a very long time.

If the program in the device is sending a massive number of debug messages, this will make the communication very slow.
Another reason can be that the program in the device is overloading the processor with too many threads that never sleep or block.

8.2 Error Messages

The most likely error messages from the VPL compiler:

"keyword/name/symbol" is expected!

This is a general syntax error.

Identifier "name" is declared more than once!

The identifier has been declared more than once within the same scope.

This is an error:

```
VAR_INPUT
   temp : INT;
END_VAR;

VAR
   temp : INT;
END_VAR;

PROGRAM testprog;
BEGIN
   ...
END;
END_PROGRAM;
```

This is not an error:

```
VAR
   temp : INT;
END_VAR;

FUNCTION_BLOCK test;
VAR
   temp : INT;
END_VAR
   // This "temp" is the one we declared in this function block
   IF temp THEN
      ...
   END_IF;
   ...
END_FUNCTION_BLOCK;

PROGRAM testprog;
BEGIN
   // this "temp" is the globally defined, before the function block was
declared.
   IF temp = 17 THEN
      ...
      ...
   END_IF;
END;
END_PROGRAM;
```

Unknown identifier: "name"

The identifier has not been declared.

"name" is *not* accessible!

This error can occur if you try to assign a value in your program to a variable that has been defined in the VAR_INPUT section.

Global variables are already declared!

It is only allowed to have one section of VAR_INPUT, VAR_OUTPUT, and VAR in the global scope.

ARRAY has illegal sub-range!

The array has not been specified properly. Remember that the lower bound must be smaller than the upper bound.

Maximum number of elements in the array has been exceeded!

You have tried to create an array with too many elements - thereby exhausting the memory available. Decrease the number of elements.

Illegal type specified. This is *not* a function block.

When you declare variables, only the simple data types and function blocks can be used as the type.

Elementary type expected (SINT/INT/DINT/BOOL/STRING/PTR/CHANNEL/SEMAPHORE/MUTEX/FILE)

It is only allowed to specify a simple data type as the type.

The variable is not compatible with a string assignment!

The variable is not declared as a string.

Illegal initializer constant!

Please examine the section Constants to see the proper way of declaring numerical constants.

Edge-detection attribute is *not* allowed!

The R_EDGE or F_EDGE are not allowed at this place.

"name" is *not* a variable!

The specified name is not declared as a variable.

IF / CASE statement expected

An IF or CASE statement is expected.

Iteration-statement expected.

A FOR/REPEAT/WHILE statement is expected.

Only a simple variable can be used as a control variable.

It is only allowed to specify a simple data type as the control value in a FOR statement.

Only a local variable can be used as a control variable.

It is only allowed to specify a local variable as the control value in a FOR statement. It is not allowed to use a global variable.

"EXIT" is only allowed in a iteration statement!

Exit is only allowed in a FOR/REPEAT/WHILE statement.

"BEGIN/END" is *only* allowed in the main program.

Use only BEGIN and END in the main program.

Incompatible types encountered!

Two variables of different types can only be assigned to each other when the destination is "larger" that the source. By larger is meant that it can hold larger numbers. Additionally, integer and floating-point numbers can not be mixed.

"name" is *not* a function!

The specified name has not been declared as a function.

String conversion not allowed!

It is not allowed to typecast a string to any other type.

Conversion to string is not allowed!

It is not allowed to typecast from any other type to a string type.

Constant, identifier, function call, or expression expected.

This is a general syntax error. This will typically be in assignments or statements where an expression is expected - such as in IF statements etc.

Arrays can *not* be assigned in a function call!

It is not legal to assign values to an array in a function call. Arrays are not supported as parameters to function calls.

Error opening file: "filename".

There was a general error with opening the file.

Error opening file: "filename" (check VINCLUDE.)

The file that is to be included by using the INCLUDE statement has not been found in the current project directory, and it is also not found in the path specified in the VINCLUDE environment variable. The VINCLUDE environment variable must point to the include directory that was created as a sub-directory with respect to the directory where the RTCU IDE program was installed. Please make sure that the file exists in the current project directory, and that the VINCLUDE environment variable has not been corrupted. If the environment variable is corrupted, please reinstall the RTCU IDE program.

Error opening file: "filename" (file locked ?)

The file name is currently open in another program and is not currently accessible. Please close the file in the other program.

Base specified is illegal (must be: 2,8 or 16).

Please examine the section Constants to see the proper way of declaring numerical constants.

Constant is illegal!

Please examine the section Constants to see the proper way of declaring numerical constants.

Unterminated string encountered.

A string has not been terminated with the " symbol. Please look at the string section for proper string usage.

Size of local variables exceeds the target system's run-time stack size!

The number of local variables in a function exceeds the maximum space available.
The stack size is 256 bytes for X32 and 512 bytes for the NX32 architecture.

8.3 Fault codes

A fault occurs when the RTCU device is not able to continue because of a severe error either in the VPL program or in some other condition detected by the firmware.
An example is if the VPL application tries to make a divide by zero operation. That will be caught by the firmware, and the device will enter a fault state where the LEDs of the device flash rapidly. The fault code will also be saved in the fault log of the device with timestamp. The fault log can be retrieved by using the Device: Data: Fault Log dialog.

It is possible to instruct the firmware to automatically reset the device after a certain time when a fault has occurred by using the boardSetFaultReset() function.

The table below describes the possible fault codes.

Fault code#	Description
-1	**Low-level TCP/IP failure.** This fault code is followed by another internal failure code. Please report this fault to Logic IO.
1	**Unspecified error**. This should not occur.
2	**Not a valid VSX image**. The program image in the device is invalid - probably because of an interruption in the programming of the device.
3	**Out of memory.** The program in the device requires more memory than available to execute.
4	**Index error.** The program in the device has made an illegal index into an array.
5	**Illegal op-code.** Most likely an attempt to use a feature that is not available in the firmware.
6	**Illegal string-ID referenced.** Can be caused by misuse of string assignments from multiple threads. See the section on STRING.
7	**Dynamic string reference illegal (not owner of string).**
8	**Dynamic string-allocation failed due to illegal size request.** Most likely the program has attempted to create a STRING larger than 254 characters.
9	**Dynamic string allocation failed.** Maximum number/size of the dynamic strings has been exceeded. See also STRING.
10	**Reference-decrement on no-reference dynamic string detected.**
11	**Divide by zero.** A division by zero operation has occurred.
12	**Stack overflow.**
13	**String memory arena has been destroyed.**
14	**X32 enhanced memory not available.**
17	**Timers used has exceeded the maximally allowed.**
18	**The callback function stack was too small.**
19	**Call is not allowed from a high priority callback function.**
20	**I/O update is not allowed from a callback function.**
128	**Flash write error.** The write to Flash has failed - probably because of too many write cycles to either the Flash persistent memory or the data loggger.
129	**GSM: Invalid PIN code.** Pin code set from the RTCU IDE or gsmSetPIN() is not correct.
130	**GSM: SIM PUK required.**

Fault code#	Description
	Too many attempts with the wrong PIN code. The PUK code must be entered by using a normal mobile phone.
131	**GSM: SIM PIN2 required.** The PUK2 code must be entered by using a normal mobile phone.
132	**GSM: SIM PUK2 required.** The PUK2 code must be entered by using a normal mobile phone.
135	**Memory allocation error.** The allocated dynamic strings in the program use too much memory, or the allocated CHANNEL data in the program use too much memory.
136	**GSM: Missing PIN code.** The SIM card is expecting a PIN code, but no PIN code has been set. A PIN code can be set either from the RTCU IDE or by using the gsmSetPIN() function.
137	**GSM: Module power-off failed.**
138	**GSM: Low-level recovery reset.**
139	**GPRS: Recovery boot #1. Permanent problems establishing the cellular connection.**
140	**GPRS: Recovery boot #2. Permanent problems with establishing the RTCU Communication Hub session.**
141	**GPRS: Recovery boot #3. Prolonged time to establish connection to the RTCU Communication Hub.**
142	**Battery charger failed. Charger permanently stopped.**
143	**User Watchdog timeout. Device resetting.**
144	**GSM: SIM-card has been reinserted.**
145	**Drive A: has been repaired due to an unexpected shutdown.**
146	**Drive B: has been repaired due to an unexpected shutdown.**
147	**Unsupported SDHC SD-CARD inserted.**
148	**Generic 1-wire lock timeout.**
149	**Temperature too high.**
151	**GPS/GNSS initialization failure.**
152	**factory.vsx is not found on the project drive.**
154	**Failed to connect to the RTCU Communication Hub. Switching to fallback configuration.**
169	**Incompatible firmware.** The firmware programmed into the device is not compatible. Please upgrade to a newer version.
199	**Unexpected reset by firmware watchdog.** The device has been reset by the firmware watchdog during upgrade.
200	**Unexpected runtime reset.**
201	**Hardware watchdog timeout. Device resetting.**

* * * * THIS PAGE IS INTENTIONALLY LEFT BLANK * * *

Quick start guide

9 Quick start guide

9.1 Quick start guide

This is a short guide that will help you get the RTCU device wired and the RTCU IDE software installed so that you can start developing RTCU applications. This is only a short guide - for complete documentation please refer to the online documentation.

Depending on the type of RTCU device there are some differences in the connection of power and I/O to the device. Please refer to the technical manual of the specific device in question.

How to get started:

To install and use an RTCU device, you will need the following:

1) A power supply
Please look at the technical description of the RTCU device you are using, but for all models a power supply that is able to deliver 12 to 24 VDC will be sufficient. Current capabilities for the power supply should a minimum of 1.5 Amp @ 12 V.

2) Antennas
A suitable GSM and GPS antenna. The connectors to use depend on the type of RTCU device you are using, but generally SMA is used for the GSM antenna, and SMB/SMA is used for the GPS antenna.

3) A SIM card
This must be a SIM card that is able to function in the specific RTCU device.

4) A suitable programming/service cable
Most RTCU devices connect to the PC using a suitable USB cable that depends on the device in use.
Please download the USB Programming Port driver package from www.logicio.com. Look under the "download" section and follow the instructions.

5) A PC:
Your PC must fulfill the following (minimum) requirements:

- Pentium-class 2 GHz or better.
- Minimum 2 GByte RAM .
- Minimum 50 MByte available space on hard disk.
- Windows Vista/Windows 7/Windows 8/Windows 10.
- 1 available USB port (or RS232 serial port for certain devices).
- Optional: modem and internet connection.

Download the most current version of the RTCU IDE for free from www.logicio.com. Look under the "download" section and follow the instructions.
To install the RTCU IDE, Integrated Development Environment, find the program named "RTCUIDE.msi" (the file name also includes the current version) and double-click on it to start the installation process. Then follow the instructions on the screen.
At the end of the installation, you can start the RTCU IDE program from the "Start" menu. The RTCU IDE program will be named "RTCU IDE".

The first time you start the RTCU IDE program, or when you upgrade the RTCU IDE to a newer version, you will be asked to register the program by filling in some information. This is to enable us to supply you with information about new versions of the program, new products, etc.

After the program starts, you can access the online help from the "Help" menu item. A good place to start is at the "Tutorial" section. This will illustrate the complete development process of a RTCU project and will guide you through the complete process - from developing the program,

building it, testing it, and finally to transferring the project to your RTCU device. This section is also included in this manual.

If you have any problems with your RTCU products, please let us know, and we will do our very best to help you.

Please see the contact information page.

* * * * THIS PAGE IS INTENTIONALLY LEFT BLANK * * *

Contact information

10 Contact information

You can contact us in one of the following ways:

Logic IO ApS
Holmboes Allé 14
DK-8700 Horsens
Denmark

Tel: **(+45) 7625 0210**
Fax: **(+45) 7625 0211**
web: **www.logicio.com**
Email: **info@logicio.com**
Support: **support@logicio.com**

- # -

- _ -

- A -

- B -

- C -

- D -

- E -

- F -

- G -

- H -

- I -

- J -

- L -

- M -

- N -

- O -

- P -

- Q -

- R -

- S -

- T -

- U -

- V -

- W -

- X -

- Z -