RTCU Gateway 2 Plug-in

Developers kit

Version 1.00

Table of Content

Introduction

This document discusses the RTCU Gateway Plug-in Developers kit, which enables you to develop plug-in's to the RTCU Gateway 2 (gateway).
A plug-in is a third-party software components that extends the gateway with new functionality.
It makes it possible to control which nodes are allowed to establish connection with the gateway.
Connect and disconnect events can be received. The name of the nodes can be set, providing a better overview of the nodes.

An example of this is the blacklist standard plug-in included with the gateway, which lets you control which clients are allowed to log-on to the gateway.

Contents of package

The package this document is part of, contains the following:

• RTCU Gateway 2 Plug-in Developers Kit.pdf	This document
• \Plug-ins\	Example plug-ins.

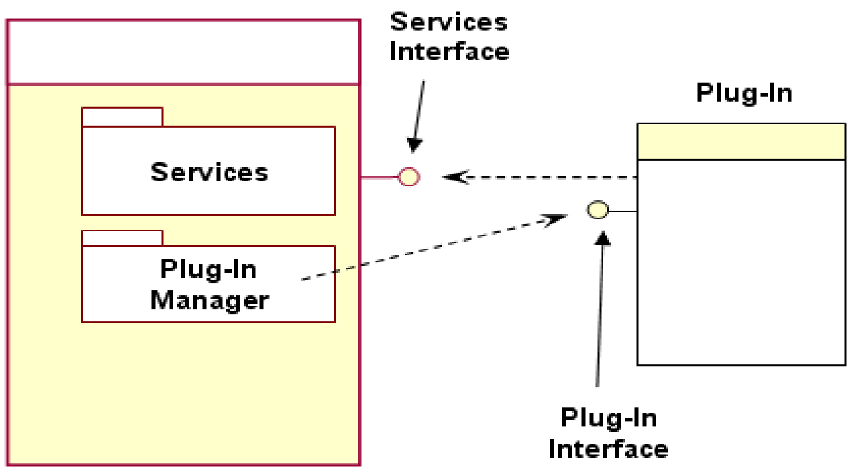
In order to develop plug-ins, you will need to use Microsoft Visual Studio C++ 2005 or newer.
The examples require Microsoft Visual Studio C++ 2010 or newer.

Plug-in interface

The plug-in interface can be described as the gateway providing services which the plug-in can use, including a way for plug-ins to register themselves with the gateway and a protocol for the exchange of data with plug-ins. Plug-ins depend on the services provided by the host application and do not usually work by themselves.

The interface consists of several functions that allows a plug-in to register, de-register and subscribe to several so called services. It also offers so call Invoke service that currently include one function to write to the gateway logging system. A service denotes a specific event that the gateway can notify the plug-in on.
In addition the plug-in may register properties that are user configurable parameters that can be edited in the RTCU Gateway 2 control panel and used by the plug-in.

The architecture of the plug-in interface is shown graphical below:



The function calling convention used is standard **__cdecl**.

Logic IO Aps
Holmboes Allé 14
8700 Horsens
Denmark

Ph: (+45) 7625 0210
Fax: (+45) 7625 0211
Email: info@logicio.com
www.logicio.com

Loading the plug-in

Loading the plug-in is done in two steps,
1. loading the plug-in DLL into memory.
2. initializing the plug-in.

After the gateway has loaded the plug-in DLL, it will search for the entry point, GWPF_InitPlugin. If the entry point is not found, the plug-in DLL will be released, and the gateway will move to the next plug-in.

The gateway will then call the entry point to initialize the plug-in.
Should the plug-in crash while in the entry point, the plug-in DLL will be released and the gateway will move to the next plug-in.

When the entry point returns from the plug-in, the gateway will check if the plug-in is registered. If the plug-in is not registered, the gateway will call the exit point to close the plug-in, the plug-in DLL will be released and the gateway will move to the next plug-in.
This last check is done to ensure that the plug-in and the gateway use a compatible version of the Plug-in Framework API.

DLL entry point GWPF_InitPlugin

Synopsis

```
gwpf_func_exit GWPF_InitPlugin(const gwpf_gateway_services
*Services);
```

Description

The entry point is responsible for initializing the plug-in and acquire the needed resources. (threads, handles, etc.)
The function is called by the gateway when loading the plug-in.
The function must be named ’GWPF_InitPlugin’, and is the only function that is required to be exported by the plug-in DLL.

Input

Services	Services is a pointer to a structure that holds the parameters necessary for the plug-in to work with the gateway.

Returns The function must return a pointer to the exit point as described below.

Configuration parameter structure

Synopsis

```
typedef struct gwpf_gateway_services {
  gwpf_api_version            version;
  gwpf_func_register          registrar;
  gwpf_func_subscribe         subscriber;
  gwpf_func_invoke            invoker;
  const char                  *configuration;
} gwpf_gateway_services;
```

Description Stores the parameters for the plug-in to work with the gateway.

Fields

version	Structure that holds the Plug-in Framework API version the gateway is using.
registrar	Pointer to the function the plug-in must call to register with the gateway.
subscriber	Pointer to the function that the plug-in must use to subscribe to services.
invoker	Pointer to the function the plug-in must call to execute commands in the gateway. Should be stored for later use.
configuration	Pointer to a zero terminated UTF-8 string that holds the configuration for the plug-in.

Plug-in Framework API version structure

Synopsis

```
typedef struct gwpf_api_version {
   int                        major;
   int                        minor;
} gwpf_api_version;
```

Description Stores the gateway plug-in framework API version.

Fields

major	Major version number.
minor	Minor version number.

Exit function

Synopsis `typedef void (*gwpf_func_exit)(void);`

Description Used by the gateway to close the plug-in.
Must release all acquired resources.

Returns None.

Registrar function

Synopsis

```
typedef int (*gwpf_func_register)(const char       *PluginName,
                                  gwpf_api_version *Version);
```

Description

The registrar is a function that the gateway pass to the plug-in when initializing it. The function verifies that the plug-in uses a compatible version of the Plug-in Framework API and then registers it.
The registrar must be called before the plug-in uses any other services from the gateway.
If the registration fails, the gateway will perform a graceful close of the plug-in with the gwpf_func_exit function.

Input

PluginName	Zero terminated UTF8 string with the name of the plug-in that is registering.
Version	Pointer to structure containing the version of the API used by the plug-in.

Returns

0	Plug-in is registered successfully.
1	Plug-in is not found.
2	Plug-in is incompatible with the gateway.

Subscribe function

Synopsis

```
typedef int (*gwpf_func_subscribe)(const char *PluginName,
                                   const int   ServiceID,
                                   void       *ServiceParams);
```

Description

The subscriber is a function that the gateway pass to the plug-in when initializing it. The function is used by the plug-in to subscribe to notifications from the gateway. The events that the plug-in can subscribe to are listed in Control Panel Services (page 10) and Gateway Services (page 13).

Input

PluginName	PluginName is a zero terminated UTF-8 string that contains the name of the plug-in, must be the same used to register the plug-in (see Registrar).
ServiceID	ServiceID is the ID number of the service to subscribe to.
ServiceParams	ServiceParams is a pointer to the parameters required to subscribe to the service.

Returns

0	Success
1	Plug-in is not found.
2	Plug-in is has not been registered.
3	Unknown service
4	Invalid parameters.
5	This plug-in have already subscribed to the service or another plug-in have

subscribed to this service and the service can only be used by a single plug-in.

Invoker

Synopsis

```
typedef int (*gwpf_func_invoke)(const char *PluginName,
                                const int  ServiceID,
                                void       *ServiceParams);
```

Description

The invoker is a function that the gateway pass to the plug-in when initializing it. This function is used by the plug-in to access the services the gateway offers.
The services offered by the gateway is listed in Available Services.

Input

PluginName	Zero terminated UTF8 string with the name of the plug-in.
ServiceID	The ID number of the service to invoke.
ServiceParams	Pointer to the parameters required by the service.

Returns

0	Success.
1	Plug-in is not found.
2	Plug-in is disabled
3	Unknown service
4	Invalid parameters.
	Other error codes as per individual service.

Configuration string

The configuration string is a UTF-8 zero terminated string containing XML syntax.
The configuration string will have the following format:

```
<configuration>
  <property name=”Enable”>1</property>
  <property name=”Folder”>C:\Logs\Gateway\</property>
  ...
  <property name=”...”>...</property>
</configuration>
```

Each property from the configuration template (see below) will be in the configuration string.

It is recommended that an XML parser library is used to parse the incoming configuration string.

Logic IO Aps
Holmboes Allé 14
8700 Horsens
Denmark

Ph: (+45) 7625 0210
Fax: (+45) 7625 0211
Email: info@logicio.com
www.logicio.com

Control Panel Services

The Control Panel supports the following service, and will return 0 for all other services, without validating them.

Plug-in read information subscription

Service ID	GWPF_SERVICE_READTEXT = 1

When adding the plug-in to the gateway, the Control Panel reads some information from the plug-in. This being, the plug-in name, the plug-in description, the plug-in version, and a template for configuration of the plug-in.
To read this information the Control Panel loads the plug-in (using an empty configuration string) and then uses Read text notifications. Once all the information is read, the plug-in is un-loaded again.

Therefore this function must at least be registered when the plug-in is initialized, otherwise the plug-in can not be added to the list of plug-ins.

gwpf_func_readtext

Synopsis

```
typedef char* (*gwpf_func_readtext)(const int Text);
```

Description The plug-in will get a series of requests for different static text information.
This is a service that all plug-ins must implement, as it is used to add the plug-in to the gateway.

Input

TextID	Id of the text information requested, it can be one of the following:	
	1	Request for the plug-in name.
	2	Request for the plug-in description.
	3	Request for the plug-in version.
	4	Request for the plug-in configuration template.

Returns The text returned is expected to be a static zero terminated UTF-8 string or an empty string if the information is not available

Registration parameter structure

Synopsis

```
typedef struct gwpf_readtext_params {
   gwpf_func_readtext func;
} gwpf_readtext_params;
```

Description	This structure holds the parameters for registering the Plug-in text reader function with the gateway.

Input

func	A pointer to the callback function that is called to read the text information.

Configuration template

The configuration template text is a zero terminated UTF-8 XML syntax string.

The XML file must have the following format:

```
<configuration>
   <property>
      <type>Bool</type>
      <name>Message log</name>
      <description>Enables/Disables adding details to the gateway
log.</description>
      <default>false</default>
   </property>
   ...
   <property>
      ...
   </property>
</configuration>
```

A property group is added to the XML string for each property that the end user can change.
The Plug-in Framework supports 5 different property types.

Type	Description
Bool	This type is an ON/OFF switch. It have the following parameters: **name**, **description**, and **default.** **Default** must be a value of 1, 0 (zero), True, or False.
Num	This type is a number. It have the following parameters: **name, description, default, max**, and **min.** **min** is the lowest value the property can have. **max** is the highest value the property can have. **Default** must be between min and max.
Text	This type is a text string. It have the following parameters: **name**, **description**, **default**, and **limit.** **limit** is the maximum length, in characters, the text string can have. If the limit is zero, then no limit will be enforced.
File	This type is a path to a file on the hard disk. It have the following parameters: **name**, **description**, **default**, **ext**, and **filter.**

	ext is the file extension, for example '*.xml'. **filter** is the filter used by the find file dialog, for example 'All files (*.*)|*.*||'.
Folder	This type is a path to a directory on the hard disk. It have the following parameters: **name**, **description** and **default.**

The name parameter for the properties is both shown in the plug-in configuration page in the Control Panel, and used in the configuration string included in the plug-in initialization.

The description parameter is used to tell the user what the property is used for.

Please note that the Control Panel adds a property called 'Enable' to the template when adding the plug-in. This property is used to Enable/Disable the plug-in for each installed gateway.

Because of this, it is not possible to use a property with this name.

Only the listed parameters used by each type will be read, other unused tags will be ignored.

Gateway Services

Log on

Service ID GWPF_SERVICE_LOGON = 3

gwpf_func_onlogon

Synopsis

```
typedef int (*gwpf_func_onlogon)(const long           NodeID,
                                 const char           *addr,
                                 const unsigned short port)
```

Description The plug-in will be notified by the gateway when a client tries to log on to the gateway, before the gateway checks key and max client limit.
The plug-in can then decide to Accept or Reject the client.

Input

NodeID	The node id the client is requesting. 0 is dynamic.
addr	A zero terminated string containing the IP address the client connects from, for example “192.168.0.1”.
port	IP port the client connects from.

Returns Returning zero from the callback function will reject the client log on, any other value will accept the client log on.

Registration parameter structure

Synopsis

```
typedef struct gwpf_service_onlogon_params {
   gwpf_func_onlogon   func;
} gwpf_service_onlogon_params;
```

Description This structure holds the parameters for subscribing to the client log on attempt.

Input

func	A pointer to the callback function.

Connect

Service ID GWPF_SERVICE_CONNECT = 4

Logic IO Aps
Holmboes Allé 14
8700 Horsens
Denmark

Ph: (+45) 7625 0210
Fax: (+45) 7625 0211
Email: info@logicio.com
www.logicio.com

gwpf_func_onconnect

Synopsis `typedef void (*gwpf_func_onconnect)(const long NodeID)`

Description The plug-in will be notified by the gateway when a client is connected to the gateway. (after successful logon)

Input

NodeID	The node id of the connected client.

Returns None.

Registration parameter structure

Synopsis

```
typedef struct gwpf_service_onconnect_params {
   gwpf_func_onconnect func;
} gwpf_service_onconnect_params;
```

Description This structure holds the parameters for subscribing to the client connected event.

Input

func	A pointer to the callback function.

Close

Service ID GWPF_SERVICE_CLOSED = 5

gwpf_func_onclose

Synopsis `typedef void (*gwpf_func_onclose)(const long NodeID)`

Description The plug-in will be notified by the gateway when a client is disconnected from the gateway.

Input

NodeID	The node id of the client that is disconnected

Returns None.

Registration parameter structure

Synopsis

```
typedef struct gwpf_service_onclose_params {
   gwpf_func_onclose func;
} gwpf_service_onclose_params;
```

Description This structure holds the parameters for subscribing to the client closed event.

Input

func	A pointer to the callback function.

Look up names

Service ID GWPF_SERVICE_NAMELIST = 6

***Note**: This service can only be subscribed to by a single plug-in.*

gwpf_func_nameitem

Synopsis

```
typedef int (*gwpf_func_nameitem)(char *name,
                                  int  *name_len,
                                  long *node);
```

Description The plug-in will get a series of request from the gateway while it iterates through the list of NodeID and Name pairs. The iteration is achieved by sending a request for the first pair and then sending a request for sub sequent pairs, until the plug-in returns No more names (3).

Input

name_len	Pointer to the variable that specifies the size, in bytes, of the buffer pointed to by the variable name. When the function returns success this variable must contain the size of the data copied to name, including the zero terminator. If name is not large enough the function must return length error and store the required size, including the zero terminator, in the variable pointed to by name_len.

Output

name	Pointer to the buffer where the name of the node is stored. The name is expected to be a zero terminated UTF-8 string.
node	Pointer to the variable that the NodeID is stored in.

Returns

0	The name and node id is returned.
1	One or more of the parameters are missing.
2	The name buffer is not large enough.
3	No more names in list.

Registration parameter structure

Synopsis

```
typedef struct gwpf_service_namelist_params {
   gwpf_func_nameitem       first;
   gwpf_func_nameitem       next;
} gwpf_service_namelist_params;
```

Description This structure holds the parameters for subscribing to the name lookup event.

Input

`first`	Callback function that is called to get the first NodeID and Name pair.
`next`	Callback function that is called to get the remaining NodeID and Name pairs.

Available Requests

Similar to subscriptions, except the structures must be given to invoke, to perform an action on the gateway.

Write to log

Service ID	GWPF_SERVICE_LOG = 2

This request makes it possible to write messages to the gateway log.

Command parameter structure

Synopsis

```
typedef struct gwpf_service_log_params {
   int                 type;
   const char          *message;
} gwpf_service_log_params;
```

Description	This structure holds the parameters for inserting a message into the gateway log.

Input

`type`	Type of log message to add.	
	1	Error
	2	Event
	3	Detailed event
`message`	Pointer to zero terminated UTF-8 text.	

Logic IO Aps
Holmboes Allé 14
8700 Horsens
Denmark

Ph: (+45) 7625 0210
Fax: (+45) 7625 0211
Email: info@logicio.com
www.logicio.com