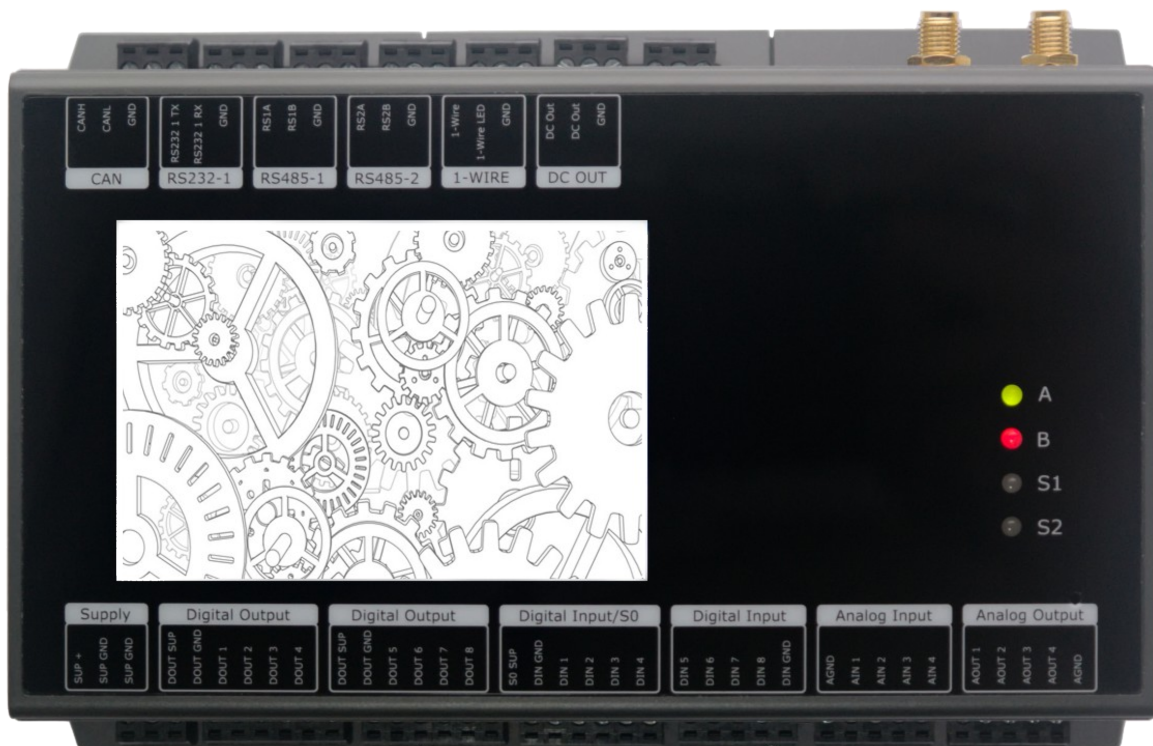
logic IO
The M2M Enabler

# RTCU Platform SDK

# V2.00




Reference Manual

# 1 Table of contents

1 Table of contents....2
2 Introduction....7
3 Getting started with the build environment....8
3.1 Installation....8
3.2 Starting the environment....8
3.3 Makefiles....8
3.4 Tools....9
3.4.1 rtcuexttag.exe....9
3.4.2 licensekeygen.exe....10
3.4.3 pem2h.exe....10
4 Program extensions....11
4.1 Execution environment for program extensions....11
4.2 Examples....12
4.2.1 EchoServer....12
4.2.2 HttpServer....12
4.2.3 WebServer....13
5 Extension modules....14
5.1 Execution environment for module extensions....14
5.2 Module licensing....14
5.2.1 The license id....15
5.2.2 Private and public key generation....15
5.3 VPL Callbacks....16
5.3.1 Synchronous and asynchronous callbacks....16
5.3.2 Callback priority....16
5.3.2.1 High priority callback limitations....17
5.4 Examples....18
5.4.1 Empty....18
5.4.2 HelloWorld....18
5.4.3 dt_mod....18
5.4.4 Example....18
5.4.5 NTP....18
5.4.6 callback....19
5.4.7 ext_lib....19
5.4.8 Wireless M-Bus....19
5.4.9 Audio....20
5.4.10 SSL....20
5.4.10.1 Server....20
5.4.10.2 Client....21
5.4.11 C++ Class....21
5.4.12 License....21
5.4.12.1 Private and public key generation....22
5.4.12.2 Generating licenses....22

5.5 Creating modules....23
5.5.1 moduleInit....23
5.5.2 moduleNotify....23
5.6 Creating Functions....24
5.6.1 VPL function....24
5.6.2 C Function....24
5.6.3 C++ Function....25
5.6.4 Variable handling....25
5.6.4.1 Variable types....26
Input variables....26
Output variables....26
Return values....27
ACCESS Variables....27
5.6.5 Finishing the function....27
5.7 Creating callback functions....28
5.7.1 VPL callback function....28
5.7.2 VPL Callback registration function....28
5.7.3 Struct for variable handling....29
5.7.4 Calling the callback....30
5.8 Troubleshooting extension modules....32
5.8.1 extModuleLoad returns -4:....32
5.8.1.1 The module is missing the functions moduleNotify and moduleInit....32
5.8.1.2 The module fails to load....32
5.8.1.3 The module has not been built correctly....32
5.8.2 extModuleLoad returns -5:....33
5.8.3 Fault 19: Call is not allowed from a high priority callback function....33
6 VPL interface....34
6.1 MODCALL....34
6.2 extModuleLoad....34
6.3 extProgramStart....35
6.4 extProgramSignal....35
6.5 extProgramStatus....35
6.6 extTagEnumerate and extTagRead....35
7 Module API Functions....36
7.1 Structures....37
7.1.1.1 vplFunctionEntry....37
7.2 Definitions and Macros....38
7.2.1 Enumerations....38
7.2.1.1 rtcu_io_signals....38
7.2.1.2 rtcu_fw_types....39
7.2.2 Declarations....39
7.2.2.1 MODDECL....39
7.2.3 Data types....39
7.2.3.1 int8....39
7.2.3.2 int16....39
7.2.3.3 int32....39
7.2.3.4 uint8....40

7.2.3.5 uint16 ... 40
7.2.3.6 uint32 ... 40
7.2.3.7 HANDLE ... 40
7.2.4 SOS Flags ... 40
7.2.4.1 SOS_FLAG_NONE ... 40
7.2.4.2 SOS_FLAG_RO ... 40
7.2.4.3 SOS_FLAG_CRYPT ... 41
7.2.5 Notification events ... 41
7.2.5.1 EVENT_HALT ... 41
7.2.5.2 EVENT_RESET ... 41
7.2.5.3 EVENT_SHUTDOWN ... 41
7.2.5.4 EVENT_POWERFAIL ... 41
7.2.5.5 EVENT_POWERSAVE ... 41
7.2.6 Alignment macros ... 42
7.2.6.1 ALIGN_ATTR ... 42
7.2.6.2 PACKED_ATTR ... 42
7.2.6.3 read16b ... 42
7.2.6.4 read24b ... 42
7.2.6.5 read32b ... 43
7.2.6.6 write16b ... 43
7.2.6.7 write24b ... 43
7.2.6.8 write32b ... 44
7.3 Type definitions ... 45
7.3.1.1 vpl_function_call ... 45
7.3.1.2 capture_handle ... 45
7.3.1.3 playback_handle ... 46
7.4 Functions ... 47
7.4.1 Basic functions ... 47
7.4.1.1 vplInstallFunctions ... 47
7.4.1.2 rtcuDebug ... 48
7.4.1.3 rtcuDebugF ... 48
7.4.2 String functions ... 49
7.4.2.1 vplStringGet ... 49
7.4.2.2 vplStringMake ... 49
7.4.2.3 vplStringUpdate ... 50
7.4.2.4 vplStringRefInc ... 50
7.4.2.5 vplStringRefDec ... 51
7.4.3 Time functions ... 51
7.4.3.1 rtcuClockGet ... 51
7.4.3.2 rtcuClockTimeToLinsec ... 52
7.4.3.3 rtcuClockLinsecToTime ... 52
7.4.3.4 rtcuClockTMToLinsec ... 53
7.4.3.5 rtcuClockLinsecToTM ... 53
7.4.4 SOS functions ... 54
7.4.4.1 rtcuSosDataCreate ... 54
7.4.4.2 rtcuSosDataUpdate ... 55
7.4.4.3 rtcuSosDataGet ... 56

7.4.4.4 rtcuSosStringCreate....57
7.4.4.5 rtcuSosStringUpdate....57
7.4.4.6 rtcuSosStringGet....58
7.4.4.7 rtcuSosFloatCreate....59
7.4.4.8 rtcuSosFloatUpdate....60
7.4.4.9 rtcuSosFloatGet....61
7.4.4.10 rtcuSosIntCreate....62
7.4.4.11 rtcuSosIntUpdate....63
7.4.4.12 rtcuSosIntGet....64
7.4.4.13 rtcuSosBoolCreate....65
7.4.4.14 rtcuSosBoolUpdate....65
7.4.4.15 rtcuSosBoolGet....66
7.4.4.16 rtcuSosDelete....66
7.4.5 Audio functions....67
7.4.5.1 rtcuAudioStreamRegister....67
7.4.6 File functions....68
7.4.6.1 rtcuFsMapFile....68
7.4.7 Certificate functions....69
7.4.7.1 rtcuCertEnumerate....69
7.4.7.2 rtcuCertGetFilename....70
7.4.7.3 rtcuCertGetCApath....71
7.4.8 License functions....72
7.4.8.1 rtcuLicenseCheck....72
7.4.9 IO functions....73
7.4.9.1 rtcuSetIOSignal....73
7.4.10 Network functions....74
7.4.10.1 rtcuNetPresent....74
7.4.10.2 rtcuNetConnected....74
7.4.10.3 rtcuNetAddressFromInterface....75
7.4.11 Device information functions....75
7.4.11.1 rtcuDeviceVersion....75
7.4.11.2 rtcuDeviceType....76
7.4.11.3 rtcuDeviceTypeName....76
7.4.11.4 rtcuDeviceSerialNumber....76
7.4.11.5 rtcuDeviceGetApplication....77
7.4.12 Callback functions....78
7.4.12.1 vplCallbackCall....78
8 Deprecated functions....79
8.1 Renamed functions....79
9 Included libraries....80
9.1 D-Bus....80
9.2 Paho MQTT client library....80
9.3 Mosquitto MQTT client library....80
9.4 json-c JSON library....80
9.5 Jansson JSON library....80
9.6 Libcurl....80
9.7 OpenSSL....81

9.8 LibGcrypt....81
9.9 libuuid....81
9.10 libiconv....81
9.11 zlib....81
9.12 libbzip2....81
9.13 PCRE1 - Perl Compatible Regular Expressions....81
9.14 Mini-XML....82
9.15 SQLite....82

# 2 Introduction

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 in C or C++ 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 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.
They also have access to the resources and functionality provided by the Linux system, but in addition also have access to the functions in the module API.
The module API makes it possible to interact with the VPL program directly, as well as to use some platform functionality directly, such as accessing the network status and the SOS.

The purpose of this document is to supply all the necessary information and tools allowing development of Platform Extensions using the **RTCU M2M Platform SDK**.

# 3 Getting started with the build environment

## 3.1 Installation

To set up the system to be able to compile new extension programs and modules, please install a 64 bit Cygwin, using setup-x86_64.exe from https://cygwin.com/. In addition to the default package selection, the `make` package must be selected to be able to build the examples.

Extract the SDK archive to a suitable location with no spaces in the path and change the `cygwin` variable in the file `use.bat` in the SDK to point at the correct path to the Cygwin root folder.

**Note:** The installation requires a 64-bit Windows system to work.

**Note**: If a path with spaces is used, building the extensions will fail with error messages such as `\library\build\module.mk: No such file or directory`.

## 3.2 Starting the environment

To start the build environment, double-click `start_sdk.bat`, which opens a command prompt with the correct paths set.

To build a module or a program, navigate to its folder and run `make`, which will build it unless it is already built and no changes to the source code have been made.

To force a rebuild, either run `make clean` followed by `make` or run `make -B`.

## 3.3 Makefiles

The build environment uses GNU Make to build the modules and programs.
The build rules and default parameters for the toolchain are located in the shared makefiles located in the `library/build` folder.
The project-specific settings are configured in the makefile in each individual project folder.
The makefiles for modules and programs are very similar and starts by setting up the following project specific parameters:

| Variable | Description |
|---|---|
| `SOURCE` | List of .o files to include in the binary. The .o files will be generated from the source .c or .cpp files with the same name.<br>Example:<br>`ntp_mod.o ntp.o` |
| `NAME` | The name of the generated binary, without the extension.<br>Example:<br>`hello` |
| `TAGS` | List of tags to provide to rtcuexttag.exe<br>Example:<br>`version 1.0 description "A simple Hello World function"` |

For more complex projects, the following variables can be used to configure how they should be built:

| Variable | Description |
|---|---|
| `OPTIMIZE` | Parameters to the compiler, e.g. optimization parameters.<br>Example:<br>`-g0 -O0` |
| `STATIC_LIBS` | List of static libraries to include.<br>Example:<br>`./mxml/libmxml.a` |
| `DYNAMIC_LIBS` | List of dynamic libraries to link against.<br>See section *9 Included libraries* for an overview of the available libraries.<br>Example:<br>`-lm -ljson-c -lcurl` |
| `INC_PATHS` | Additional folders to search for include files.<br>Example:<br>`-I"${SDK}\sysroot\usr\include\json-c" -I"${SDK}\sysroot\usr\include\curl"` |

The project-specific makefile must end by including the shared makefile to add the rules needed to build the project.

## 3.4 Tools

The following tools are available in the `library/bin` folder.

### 3.4.1 rtcuexttag.exe

The tool rtcuexttag.exe is used to add tags to the programs and modules, to mark them as valid extension programs and modules.

In addition to the built-in tags, it is possible to define additional tags that will be shown in the IDE when installing the extension and which can be read from the VPL application.

```
rtcuexttag <filename> [options] [metadata]

  options:
    -v                  - Show verbose output
    -h                  - Show how to use the tool
    -out <filename>     - Store the result in <filename>
```

The metadata is a list of space separated key-value pairs, each defining a single tag. It is possible to use quotes to use values containing spaces.

The examples in the SDK uses the makefiles to define the custom tags, such as the version and description, resulting in a command line that looks like this:

- `rtcuexttag.exe empty.so -out empty.rmx version 1.0 description "An empty module"`

This results in a module, `empty.rmx`, with a version tag of 1.0 and a description tag of "An empty module".

Every extension module and program will contain a special built-in tag, `checksum`, which contains a checksum of the entire extension. By reading the checksum from the extension with `extTagRead` and comparing it with the known checksum from the build, it is possible to validate that the extension has not been modified since it was built.

### 3.4.2 licensekeygen.exe

The tool licensekeygen.exe is used to generate device specific extension module license keys.

```
LicenseKeyGen generate <license id> <private key> <serial>

where:
 <license id>     - The license id string
 <private key>    - The PEM file with the private key
 <serial>         - The serial number of the RTCU device
```

The private key used by the tool is the private key of an asymmetric encryption key pair.

The tool can use private key both with and without password protection.

### 3.4.3 pem2h.exe

The tool pem2h.exe is used to convert a PEM file containing a public key, into a C header file.

```
pem2h <public key PEM file> <header file>
```

By including the header file in the extension module, the two variables, `public_key_size` and `public_key`, can be used when checking if a license is present.

# 4 Program extensions

The programs extensions makes it possible to run custom binary code in the RTCU device, by building a stand-alone program and then transferring it to the RTCU device and running it.

The program extensions are stand alone programs, that are built normally and then run through rtcuexttag.exe, which adds a number of tags to the module, marking it as a valid program extension.

To communicate with the VPL application, it is possible to use sockets, files and other similar methods for communicating between processes.

The program extension runs in its own separate process so if it crashes it will not take the entire system down with it.

The program extensions are especially suitable when a program for solving the task already exists or for long running tasks, e.g. for providing services for the VPL application or via external interfaces.

## 4.1 Execution environment for program extensions

The program starts with the current folder set to the location of the program, typically the `SYSTEM\EXT\` folder on the internal drive.

To navigate to the SD Card or an USB drive, use relative paths, using the drive letter(in lower case) to change the drive, e.g. `..\..\..\a\` to access the SD Card. Note that the drives must be opened from VPL first.

Absolute paths should be avoided as the paths may be changed without notice.

Note that on the internal drive, all file and folder names are case sensitive and all names on it are with uppercase letters.

## 4.2 Examples

The SDK contains the following examples extension programs:

### 4.2.1 EchoServer

This program is used to demonstrate a way for a VPL application to communicate with the extension program.

EchoServer sets up a socket and begins listening on it. When the VPL application connects to the socket, it sends "Hello, World!" to the application, waits 5 seconds and closes the socket.

This could be expanded to e.g. send a command to the program which could then respond with the result of the command.

### 4.2.2 HttpServer

This program demonstrates how an extension program can be used to make data from the VPL application available as a web-page, without having to create the entire server or the HTML page from VPL.

HttpServer is a very simple HTTP server, based on the Simple web-server[1]. When started, it starts serving the files in the `B:\WEB` folder.

The VPL application creates the `B:\WEB` folder and populates it with an `index.htm` file.

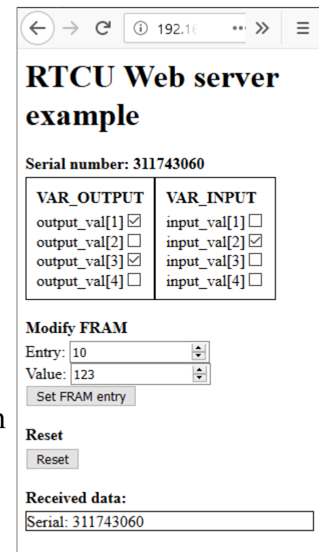
The application then starts the server and begins updating the file `B:\WEB\STATUS.TXT` with the current supply voltage as well as the serial number of the device.

`Index.htm` then uses JavaScript to fetch the `STATUS.TXT` file at a fixed interval and updates the page with the current values.

This could e.g. be expanded to VPL creating one or more simple files (perhaps in the JSON format) that are updated when a basic status changes, which is then made available via the web-page.

1 https://github.com/labcoder/simple-webserver/blob/master/server.c

### 4.2.3 WebServer




This program shows how it is possible to control the device from a web page hosted on a simple web server inside the device.

By default it uses the LAN interface, but it can be changed to e.g. WiFi by changing the `iface` variable to 4.

The web server can be accessed from a browser by connecting to the URL shown in the device output, once it has connected to the network.

To avoid having to create entire HTML pages from VPL, all the dynamic content is created by JavaScript that will run in the web browser when it loads the `INDEX.HTM` page, which is just a static file on the internal drive.

The web page updates the input status every 10 seconds and by clicking on the check boxes and the buttons, it is possible to set the outputs, write to FRAM or to reset the device.

The I/O mapping is done in the Job Variable Configuration dialog in the RTCU IDE. By default it is configured as follows:

| input_val | |
|---|---|
| 1 | DIP Switch 1 |
| 2 | DIP Switch 2 |
| 3 | Digital Input 1 |
| 4 | Digital Input 2 |

| output_val | |
|---|---|
| 1 | LED 1 |
| 2 | LED 2 |
| 3 | LED 3 |
| 4 | LED 4 |

Every request is sent by the JavaScript in the web browser to the URL “`/data?`”on the web server.

The web server is a modified version of the HttpServer example. When an URL that starts with "`/data?`" is accessed, the web server creates a socket connection to the VPL application, which is listening on port 3490, and then sends all the text after the "`?`" to the VPL application.

The VPL application parses the request and handles it, e.g. by setting an output or writing to FRAM before it returns a response which the web server then returns on to the web browser.

The JavaScript in the web browser then parses the response and updates the web page.

This allows for using VPL to handle the low level I/O functionality while at the same time having access to all the high level UI functionality provided by HTML, CSS and JavaScript.

# 5 Extension modules

The extension modules makes it possible to run custom C/C++ functions directly from the VPL runtime, similar to how the ordinary platform support functions work.

This makes it possible to add powerful new functions to the VPL applications to e.g. manipulate strings, perform complex calculations or to use custom protocols to communicate over external interfaces.

This is a much more advanced form of extending the runtime than the extension programs and is very likely to cause the system to crash in the case of an error.

## 5.1 Execution environment for module extensions

The current folder for the extension modules starts out as the location of the internal drive, but as it is shared for the entire process, it can be changed by other extension modules.

The recommended way for finding the path to a file is to use `rtcuFsMapFile`. It can map both VPL files and devices such as serial ports to the system paths, which can then be used with the normal file operations.

Note that on the internal drive, all file and folder names are case sensitive and all names on it must use uppercase letters.

## 5.2 Module licensing

The license feature is a way for modules to check if a valid license for the module is present.

For a license to be valid, it must be present and match both the id of the module and the serial number of the RTCU device.

To ensure that only the creator of the module can issue valid licenses,  the license key is signed with a private key, using an asymmetric encryption mechanism. The extension module then uses the public key to validate the license key.

It is recommended that a new public/private key pair is generated for each different extension module.

*Important: remember to make a secure backup of the key pair. If the private key is lost, no new licenses can be made.*

To make a license for an extension module, the tool licensekeygen.exe is used.

It will generate the license and a license key which is used to apply the license in an RTCU device.

A different license key must be generated for each RTCU device, as the license is bound to each individual device. It is not possible to create a license key which is valid for multiple RTCU devices.

To make is easier to include the public key in the module code, the pem2h.exe tool can be used to convert a PEM file containing the public key to a header file which can be included in the module.

### 5.2.1 The license id

The license id is a text string which uniquely identifies the module that a license applies to.

It is part of the license key and in addition to validating the license, it is also used when managing the license key.

To prevent multiple modules from using the same license id, it must follow the following conventions:

The license id should always start with the name of the company which makes the extension module, followed by the name of the extension module, with an optional feature name.

The license id can only contain the lower case letters a..z and the numbers 0..9, with the '.' character separating the names.

The license id can maximum have a length of 50 characters.

Example of a valid license id:

- `example.license.feature`

### 5.2.2 Private and public key generation

The license is encrypted using asymmetric encryption, which means that a public/private key pair is needed.

The private key is used to encrypt the license when creating it, and the public key is used to decrypt the license when the RTCU device verifies the license.

The private and public keys can be generated with different tools, including OpenSSH (typically preinstalled in Windows 10) and OpenSSL.

To create the key pair using OpenSSH, the following commands can be used:

```
ssh-keygen -m pkcs8

ssh-keygen -f <key.pub> -e -m pkcs8 > public_key.pem
```

The first command creates the keys, and the second converts the public key to the PEM format.

(replace the <key.pub> with the name of the public key made in step one)

To create a key pair using OpenSSL, the following commands can be used:

```
openssl genpkey -algorithm RSA -out private_key.pem -pkeyopt
    rsa_keygen_bits:2048

openssl rsa -pubout -in private_key.pem -out public_key.pem
```

To password protect the private key when using OpenSSL, the following command can be used:

```
openssl genpkey -aes-256-cbc -algorithm RSA -out private_key.pem
    -pkeyopt rsa_keygen_bits:2048

openssl rsa -pubout -in private_key.pem -out public_key.pem
```

## 5.3 VPL Callbacks

The VPL Callback feature makes it possible for C functions to call VPL functions directly.

This can be useful for event handling and for letting VPL make decisions for the module.

Typical usage examples include generating data when requested by a remote party, fast handling of events, error handling and logging. It also provides a simple method for having optional steps depending on whether the callback is registered or not, e.g. for debugging.

### 5.3.1 Synchronous and asynchronous callbacks

VPL Callbacks can be both called synchronous and asynchronous, depending of the value of the return value pointer.

If the return value pointer is set to `NULL`, the callback will be run asynchronously and the return value from the callback function will be ignored, allowing vplCallbackCall to return immediately.

If the pointer is not `NULL`, the return value from a successful callback function will be stored in the `int32` variable provided by the pointer.

### 5.3.2 Callback priority

The callbacks can be executed at normal or high priority and determines when the callback function is called.

**Normal priority callbacks**

When using normal priority, the callback function is scheduled to be executed once all the previous

callbacks are done. The callback function can do anything a normal thread can do but should try to return quickly to let other callbacks execute.

**High priority callbacks.**

High priority callbacks will be executed immediately but will not be able to call anything but the most basic functions, see High priority callback limitations. This type of callback is suited for interrupt-like callback functions that e.g. sets a flag or increments a counter but does not need to do anything else.

#### 5.3.2.1 High priority callback limitations

The high priority callback functions are only allowed to call the Standard functions and function blocks listed below. Calling any other function from a high priority callback will cause the device to fault with fault *19: Call is not allowed from a high priority callback function*.

```
abs
acos
asin
atan
atan2
bcd_to_dint
bcd_to_int
bcd_to_sint
ceil
chDestroy
chInit
chPeek
chRead*
chStatus
chWrite*
clockDayTimer
clockGet
clockLinsecToTime
clockNow
clockSet
clockTimeToLinsec
cos
crcCalculate
CTD
CTU
CTUD
DEBOUNCE
DebugFmt*
DebugMsg*
degToPosition
degToRad
dint_to_bcd
dintToHex
dintToStr
doubleToStr
exp
exp2
F_TRIG
fabs
floatToStr
floor
frexp
hexToDint
hexToInt
hexToSint
int_to_bcd
intToHex
intToStr
ISINF
ISNAN
ldexp
log
log10
log2
MAX
memcpy
memioRead
memioReadX
memioWrite
memioWriteX
MIN
modf
mxDestroy
mxInit
mxLock
mxStatus
mxUnlock
PCT
positionToDeg
R_TRIG
radToDeg
RF_TRIG
RS
semDestroy
semInit
semSignal
semValue
semWait*
shl16
shl32
shl8
shr16
shr32
shr8
sin
sint_to_bcd
sintToHex
sintToStr
sqrt
SR
strCompare
strConcat
strDecode64
strEncode64
strFind
strFormat
strFromMemory
strGetStrings
strGetValues
strLeft
strLen
strLookup
strMemoryUsage
strMid
strRemoveSpaces
strRight
strToDint
strToDouble
strToFloat
strToInt
strToken
strToMemory
strToSint
strUnlock
tan
thGetID
TOF
TON
TP
VolumeHorizontalTank
VolumeVerticalTank
```

Note that the `DebugMsg` and `DebugFmt` functions will only send the message over a cable connection to the IDE if the message is less than 241 characters. They will not send it to remote clients or add it to the log file.

The `semWait`, `chRead` and `chWrite` functions can only be called with a timeout of 0, any other value will cause the device to fault.

## 5.4 Examples

The following example modules are available in the SDK, located in the `modules` folder:

### 5.4.1 Empty

The `Empty` module is a module with no functions. It is meant as a template for creating new modules.

### 5.4.2 HelloWorld

The `HelloWorld` module contains one function, `HelloWorld`, which returns the string "Hello World".

### 5.4.3 dt_mod

The `dt_mod` module shows how different data types can be moved between VPL and C.

It contains a number of functions with different parameters.

The functions `stringToUpper` and `numberToHex` are helper functions. `stringToUpper` sets the `ACCESS` variable `out` to the uppercase version of the string `str`. `numberToHex` returns a string containing the value of the given DINT as a hexadecimal value with optional padding to 8 characters.

The functions `sintTest`, `intTest`, `dintTest`, `floatTest` and `boolTest` all takes a value and uses it to create two values, one stored in the access value and the other returned from the function.

`sbTest` populates the given `sb_data` STRUCT_BLOCK variable with 7 elements.

`get_buf` and `free_buf` allocates and frees dynamic memory, to show one possible way to use pointers. Please note that allocating large amounts of memory may cause the system to run out of memory and crash, so care should be taken.

`fbTest` is an example of a FUNCTION_BLOCK with some variables.

### 5.4.4 Example

The `Example` module contains a more complete `numberToHex` function that the dt_mod module.

It takes a DINT value and the number of characters to pad the number to and returns a string with the padded hexadecimal value.

It is used for the examples in section *5.6 Creating Functions*.

### 5.4.5 NTP

The `NTP` module uses NTP to provide the current time for the wanted timezone.

The module consists of two C-files.

`ntp.c` contains the NTP client implementation, based on a standalone ntp client[2], changed to be a single function, ntp_get_time.

`ntp_mod.c` contains the code for the module. It has the function get_time which takes the host name from VPL and passes it on to the `ntp_get_time` function. If a timezone is specified, it then uses `localtime` to convert the time into the local time before converting the time to a linsec and returning it.

### 5.4.6 callback

The `callback` module shows how VPL callback functions can be called from a module.

The example uses the `ConfigModule` function to register three callbacks:

- `OnEvent` which is high priority, asynchronous callback that is called every time the `calculate` or the `UpdateText` functions are called. It increments a counter which is printed out in the end.
- `OnCalc` which is called by the `calculate` function. It performs a small calculation which can be read from the `GetValue` function.
- `OnText` which is called by the `UpdateText` function. It updates the string accessible from `GetText`.

The example shows how callbacks can be called and how to pass values to and from the callbacks.

### 5.4.7 ext_lib

The `ext_lib` module is an example of how to use external, static libraries.

The `mxml` folder contains the mini-XML library[3]. It can be built by calling `sh build.sh` from the `mxml` folder, which generates the library `libmxml.a`.

The `ext_lib` makefile has the variable `STATIC_LIBS` set to point at `libmxml.a`, which causes it to be linked with the rest of the module.

The module itself has a single function that parses an XML file and prints an attribute from it to the device output.

The XML file is generated by the VPL application before it calls the function to parse it.

### 5.4.8 Wireless M-Bus

The Wireless M-Bus example module `mbus` is a complex example that also provides the starting point for an API for controlling the Radiocrafts RC1180-MBUS Wireless M-Bus Module that is available on some versions of the NX-400.

For a detailed description of this example, please see the documentation in `modules/mbus/doc/index.html`.

2https://github.com/lettier/ntpclient/blob/master/source/c/main.c
3https://github.com/michaelrsweet/mxml

## 5.4.9 Audio

The Audio example module demonstrates the usage of the `rtcuAudioStreamRegister` function and how modules can be used to handle streaming audio.

The application waits for an incoming phone call and then it plays the file `B:\SND.WAV` to the caller while recording any incoming audio.

When the call is terminated, it stops the recording and is ready for the next call.

To make sure that there is always sound to play, the file is repeated indefinitely.

The module has two callback functions:

- test_stream_capt reads data from the file `B:\SND.WAV` and passes it on to the stream, which then sends it to the caller.
- test_stream_playb receives the caller audio from the stream and records it to the file `B:\REC_10.WAV` on the internal drive.

To reduce the latency, performance critical applications should not generate the filenames and open the files in the callbacks, but e.g. do this from a separate function.

## 5.4.10 SSL

The SSL example module demonstrates how to use the OpenSSL library to handle TLS sockets, and the usage of the `rtcuCertGetCApath` and `rtcuCertGetFilename` functions for accessing the certificate store.

The example contains an example server certificate and the CA certificate to verify it; both the certificates will be installed on the device if not present.

The application provides a very primitive HTTPS server and an equally primitive HTTPS client that connects to the server to read a simple message.

### *5.4.10.1 Server*

The server is started when the application starts, by calling the ssl_server function, makes it listen for incoming TLS connections on port 5001.
When a connection is established, a simple HTTP response is sent to the client and then the connection is closed.
After the connection is closed it continues to listen for connections.

It is possible to connect to the server from a normal browser,  the address to connect to is shown in the device output when the network connection has been established.

Note that the browser will not accept the default certificate, as it does not know the CA and it does not match the IP address of the device, but it can be ignored, typically by clicking advanced and then clicking on the button to continue to the site.

#### 5.4.10.2 Client

To start a client, set DIP switch 1 ON.

This will call the ssl_client function, which opens a TLS connection to a server.
When a connection is established, a simple HTTP request is sent, and the function waits for a reply. The reply will be sent to the device output line for line, once the reply is received the connection is closed.

By default the client will connect to the server via localhost, but it can be changed to e.g. connect to a server running on a different device by changing the address in the call to ssl_client.

It is also possible to make it connect to a different server on the internet to show that it work with other certificates:

```
ssl_client(ip := sockIPFromName(str := "example.com"), port := 443);
```

Note that the CA certificates for validating the server certificate must be installed for the connection to work.

### 5.4.11 C++ Class

The C++ class example module in the `cpp_class` folder demonstrates how to make modules in C++.

The module contains a single function, `vplOutput`, which creates an instance of the class `output`, calls the `print()` function on it to get a string and returns it to VPL.

### 5.4.12 License

The license example module shows an example of how a module can use the license feature to check if there is a valid license for the module installed.

This can be used to create modules that are licensed to third parties.

The module checks for a license in the start of `moduleInit` and if no license is present, it sets the trial flag. If a license is present but it is not valid, it returns an error instead, stopping the loading of the module. In both cases, a messages is sent to the debug output to inform the user that a valid license is needed.

If the license is valid or the trial flag is set, the loading continues, creating the `HelloWorld` function. When called, the `HelloWorld` function checks the trial flag which determines the string to return.

The VPL project will show if a license is installed and call the `HelloWorld` function.

By sending an SMS from the IDE with the content `add`, the license configured in the code will be installed while sending `remove` will remove the license key. As the license is only checked when the module is loaded, the application must be restarted to see the change.

The license key included by default in the code is generated for serial number 123456789 and will

not be valid on other devices, so a valid license must be generated for the specific device, following the description in section *5.4.12.2 Generating licenses*.

### 5.4.12.1 Private and public key generation

The private and public keys can be generated using OpenSSH (typically preinstalled in Windows 10) and OpenSSL. The makefile in the example has been extended with special targets to demonstrate how these keys can be generated.

Note that when generating a new key pair, it will overwrite the old pair which will prevent the generation of further licenses for modules using the old key pair. It is therefore very important to back up the keys used for deployed modules.

The following commands can be used to generate the keys:

- `make generate_openssh`

This command will generate a new private/public key pair using ssh-keyen from OpenSSH.

- `make generate_openssl`

This command will generate a new private/public key pair using OpenSSL.

- `make generate_openssl_pass`

This command will generate a private/public key pair using OpenSSL with a password protected private key. It is necessary to enter the wanted password three times; twice for validating the password when generating the private key and once to unlock the private key when generating the public key.

When the public key has changed, the header file must be updated using the following command:

- `make key`

This command will use pem2h to convert the public key to a header file, which will update the key used by the module.

### 5.4.12.2 Generating licenses

The following command can be used to generate licenses for a specific device:

- `make license SERIAL=<serial>`

It uses LicenseKeyGen to generate a new license key for the device with the serial number provided in the variable SERIAL.

## 5.5 Creating modules

The module extensions are shared libraries, that are built normally and then run through `rtcuexttag.exe`, which adds a number of tags to the module, marking it as a valid module extension.

The adjustable tags, such as version and the description are defined in the makefile.

To be able to load the module, it must export the functions moduleInit and moduleNotify.

### 5.5.1 moduleInit

This function is called by the system when `extModuleLoad` is called.

Any initialization the module might require must be performed here, including installing functions and function blocks.

See `vplInstallFunctions` for information on how to install modules.

**Returns:**

- 0 Success
- 1 Failed to initialize module.

**Declaration**

```
int MODDECL moduleInit(void)
```

### 5.5.2 moduleNotify

The function `moduleNotify` is called whenever an event happens that the module should handle. This could e.g. be power events such as reset, where the module may need to release some resources.

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | event | - | The event ID |
| In | parm | - | The event parameter |

**Returns:**

None.

**Declaration**

```
void MODDECL moduleNotify(int event, int parm)
```

## 5.6 Creating Functions

This section describes how to create a new function, using the `Example` module as the basis.

### 5.6.1 VPL function

To create a new custom function, start by creating the VPL function that will call the C function, and have it call the `MODCALL` function, with the expected name of the module and function. Make sure the function uses the `ALIGN` keyword, to make it align the variables correctly on the stack.

```
FUNCTION ALIGN numberToHex:STRING;
VAR_INPUT
   v:DINT;
   padding:SINT := 0;
END_VAR;
VAR
   err:INT;
END_VAR;
   numberToHex := STRING(INT(MODCALL("mod_example",
     "numberToHex", err)));
   IF err <> 0 THEN
      numberToHex := "Error";
   END_IF;
END_FUNCTION;
```

*From modules/example/example_vpl/mod_example.vpl.*

Build the VPL project to generate, among other files, the `.LST` file, which is needed to define the C function.

### 5.6.2 C Function

If creating a new module, create a new C file by copying the example file `modules\empty\empty.c`, otherwise the function should just be added to the existing file.

Create a new C function with the wanted name and a signature matching the signature of `vpl_function_call`.

```
static int32 MODDECL numberToHex(HANDLE* pCPU, void* pbase)
{
     ...
}
```

*From modules/example/mod_example.c.*

Add the name of the function(used in `MODCALL`) and the function to the function table, ftable. This

will make the function be called when the VPL function calls MODCALL.

```
static vplFunctionEntry ftable[] =
{
      { "numberToHex", numberToHex },
      { NULL, NULL }
};
```

*From modules/example/mod_example.c.*

### 5.6.3 C++ Function

The only change needed to use C++ to implement the functions instead of C is that the file extension must be '.cpp'.

Just like the C functions, the C++ functions must have a signature matching the signature of `vpl_function_call` (non-static member functions of a class can not be used), and must be added to the function table.
See section *5.4.11 C++ Class* for an example of how to make a module in C++.

### 5.6.4 Variable handling

Open the `.LST` file from the VPL project and find the newly created VPL function:

```
;*** FUNCTION numberToHex ***
;*** var
;numberToHex          off=0000 key=791722 sz=02 ty=std ival=0
;err                  off=0002 key=791745 sz=02 ty=std ival=0
;*** var_input
;v                    off=0004 key=749631 sz=04 ty=std ival=0
;padding              off=0008 key=791735 sz=01 ty=std ival=0
;*** var_input (access)
```

*From modules/example/example_vpl/mod_example.lst.*

Below the line with the function name, is the variables of the functions.

Each line contains one variable, ordered by what kind of variable it is, i.e. input, input access, output or local variables.

The first variable has the name of the function and will be used from VPL for the return value from the function.

The second variable is the local variable `err`, which is used for the call to `MODCALL`. It should not be modified, as it is already used by `MODCALL`.

After the name of each variable is a number of features about the variable:

- `off` is the offset of the variable from the start. The first variable, that has the same name as the function, is the return code and has the offset of 0.
- `sz` is the size of the variable. SINTs and BOOLs are 1 byte, INTs and STRINGs are two bytes and DINTs and FLOATs are 4 bytes.

Function blocks are similar to functions, except that they do not have a return code and that they supports output variables.

Based on this information, create a struct that contains the same variables in the same order, using data types of the specified size. The struct must use the `ALIGN_ATTR` attribute.

```
typedef struct {
   /*** var */
   int16       retval;
   int16       error;
   /*** var_input */
   int32       v;
   int8        padding;

} ALIGN_ATTR tdef_numbertohex;
```

*From modules/example/mod_example.c.*

In the C function, create a struct pointer to your newly created struct, cast the `pbase` pointer to be a pointer to the struct and assign it to the struct pointer.

```
tdef_numbertohex *data =(tdef_numbertohex*)pbase;
```

*From modules/example/mod_example.c.*

For very simple functions, the pointer can instead be cast in-place to avoid having to create the struct pointer.

### 5.6.4.1 Variable types

Depending on the kind and type of the variables, they must be handled differently.

See the `dt_mod` example for examples for many of the different combinations.

#### Input variables

The number types and BOOLs can be used directly from the struct.

To get a string from an input variable, use the `vplStringGet` function, which takes the string id from VPL and returns a pointer to the string.

#### Output variables

Output variables using the number types and BOOLs can be set just by assigning the value to the variable.

To set the string in an output variable, use the `vplStringMake` function to store the string.

#### Return values

The return code is set to the return code from the function.

To return a string, create a new string id by calling the `vplStringMake` function.

The number types and BOOL can just return the value directly.

#### ACCESS Variables

`ACCESS` variables are a bit more complex.

`ACCESS` number and BOOL variables are pointers to the actual VPL variable, and as the variables are not necessarily aligned, the data must be copied to and from the variables using the `write16b`/`write32b` and `read16b`/`read32b` functions.

`ACCESS` STRING variables are still string IDs.

To retrieve the string from an `ACCESS` STRING variable, use the `vplStringGet` function.

To change the string in an `ACCESS` STRING variable, use the `vplStringUpdate` function.

### STRUCT_BLOCKs

Struct blocks ACCESS variables can be represented as struct pointers within the variable struct. The structure of the STRUCT_BLOCK can also be read from the `.LST` file. See the `dt_mod` example for an example of how ACCESS Struct blocks can be handled.

## 5.6.5 Finishing the function

Once the variable handling is in place, all that remains is to implement the custom functionality.

Then it is just a matter of rebuilding the module by running `make`, installing it using the IDE and to test if it works by loading and calling it from the VPL application.

If it does not work as expected, the `rtcuDebug` and `rtcuDebugF` functions can be used to send debug messages to the IDE from strategic places in the code, to help with the debugging.

If the system crashes before any debug message can be printed from the function, it is likely a problem with how the variables are handled.

## 5.7 Creating callback functions

This section describes how to create a new callback function, using the `callback` module as the basis.

### 5.7.1 VPL callback function

To create a new custom callback function, start by creating the VPL function that will be called. Make sure the function uses the `CALLBACK` keyword, so it can be used as a callback function.

```
FUNCTION CALLBACK  OnCalc : DINT;
VAR_INPUT
   val1  : DINT;
   val2  : DINT;
END_VAR;
   cb_calc := val1 + val2;
END_FUNCTION
```

*From modules/callback/callback_vpl/test.vpl.*

Build the VPL project to generate, among other files, the `.LST` file, which is needed to define the C function.

### 5.7.2 VPL Callback registration function

To be able to call the callback, the module must know the address of the callback function to call.

The address of the callback function must be passed to the module using a registration function that use the `CALLBACK` data type for the callback address parameter.

```
FUNCTION ALIGN ConfigModule : INT
VAR_INPUT
   event : CALLBACK;
   calc  : CALLBACK;
   text  : CALLBACK;
END_VAR;
VAR
   rc    : INT;
END_VAR;
   ConfigModule := INT(MODCALL("callback", "config", rc));
   IF rc <> 0 THEN
      ConfigModule := -1;
   END_IF;
END_FUNCTION;
```

*From modules/callback/callback_vpl/callback.inc.*

When registering the callback function, use the @ operator to get the address of the function.

```
ConfigModule(event := @OnEvent,
             calc  := @OnCalc,
             text  := @OnText);
```

The registration function must store the callback addresses in the module to make them available for the calling functions.

### 5.7.3 Struct for variable handling

A callback function must have a corresponding struct that matches the variables in the `.LST` file.

The struct can be created based on the procedure described in *5.6.4 Variable handling*.

```
typedef struct {
   /*** var_rc */
   int32       retval;
   /*** var_input */
   int32       val1;
   int32       val2;
   /*** var_input */
   /*** var */
} ALIGN_ATTR frame_calc;
```

*From modules/callback/callback.c.*

In the module, this struct is used as the basis for the stack frame, to provide the parameters to the callback function.

The struct must always be created and handed to the callback, even if the callback function does not have any parameters, as the return code will always be present.

### 5.7.4 Calling the callback

To execute the callback, the vplCallbackCall function is used. As some setup and variable handling is required, it is recommended to wrap it in a function matching the VPL callback, see eg. the `call_OnCalc` function in the Callback example:

```
static int32 call_OnCalc(HANDLE* pCPU, int32 val1, int32 val2)
{
   // Var
   frame_calc frame;
   int32      retVal;
   int        rc;

   // Build frame
   frame.retval = 0;
   frame.val1   = val1;
   frame.val2   = val2;

   // Callback
   rc = vplCallbackCall(pCPU, cb_calc, (uint8*)&frame, sizeof(frame), 0, &retVal);
   if (rc) return 0;

   // Completed
   return retVal;
}
```

*From modules/callback/callback.c.*

The return value from the callback function will always be provided as an `int32`, so in some cases it will be necessary to convert it to get the correct value. Because of this. it is not possible to return 64-bit values such as `DOUBLE`.

## 5.8 Troubleshooting extension modules

There can be many reasons for an extension module to not work as expected:

### 5.8.1 extModuleLoad returns -4:

The following are typical causes for return code -4:

#### *5.8.1.1 The module is missing the functions moduleNotify and moduleInit.*

Please make sure that these functions have been implemented.

#### *5.8.1.2 The module fails to load.*

It may be that the module is depending on some libraries that are not available.

The tool `objdump` can be used to get a list of the dependencies:

```
arm-cortexa5-linux-uclibcgnueabihf-objdump -p hello.rmx | grep NEEDED
```

The list of libraries can then be checked against the default libraries available in the SDK library folders:

- `sysroot/lib`
- `sysroot/usr/lib`
- `library/lib`

If it needs a library that is not available in one of these folders by default, it is necessary to link to the library statically.

#### *5.8.1.3 The module has not been built correctly.*

It is possible that a wrong compiler has been used for compiling the module. To check if this is the case, compare it against a known good module, e.g. one of the examples.
This can be done with multiple tools:

- `file`:
  - Call `file hello.rmx`
  - Should show "`ELF 32-bit LSB shared object, ARM, EABI5 version 1 (SYSV), dynamically linked, not stripped, with debug_info`" or something similar.
- `readelf`
  - Call `arm-cortexa5-linux-uclibcgnueabihf-readelf -h hello.rmx`
  - `Machine` must be `ARM`.
  - `Flags` must be `0x5000400, Version5 EABI, hard-float ABI`.

If what it shows does not match, it suggests that the wrong compiler has been used.

Please verify that the build environment is set correctly up.

### 5.8.2 extModuleLoad returns -5:

The `moduleInit` function returned a value different from 0.

Please check the source for the module to determine the reason for this.

### 5.8.3 Fault 19: Call is not allowed from a high priority callback function.

A high priority callback function has tried to call a function that it may not call. See *High priority callback limitations* to determine which functions are allowed to be called.

# 6 VPL interface

The program and module extensions are controlled by the VPL application, using the functions below. For more information on how to use the functions from VPL, see the RTCU IDE Online Help.

## 6.1 MODCALL

`MODCALL` is used from VPL to call C functions in the module extensions.

For examples of how this can be used, please see the examples in the `modules` folder.

The `dt_mod` example is especially interesting as it shows how to handle many different types of variables.

**Parameters:**

| Direction | | Description |
|---|---|---|
| In | - | The name of the module to call |
| In | - | The name of the function in the module. |
| Out | | The result code from calling the function. A variable, e.g. called `err`, must be used for this which may not be touched from within the function call, as it is automatically set from the system.<br>The possible values for this parameter are:<br>• 0 = Function was successfully invoked.<br>• 1 = The symbolic function name was not found.<br>• 2 = Invalid module and/or function name.<br>• 3 = Not supported. |

**Returns:**
The return value from the C function, if the out parameter is set to 0.
Depending on the wanted return type, it is often necessary to cast it before it can be returned from the VPL function.
STRINGs require an extra cast to first convert the return value to an INT, i.e. `STRING(INT(MODCALL(…)))`.

## 6.2 extModuleLoad

`extModuleLoad` is used to load module extensions. It is called with the path to the extension, and will call the `moduleInit` function in the module.
Please see section *5.8 Troubleshooting extension modules* for information on common solutions when this function fails.

## 6.3 extProgramStart

`extProgramStart` starts the program extension, which will then run in a separate process.

## 6.4 extProgramSignal

`extProgramSignal` sends a signal to a running program extension, either to stop it or just to notify the program that it must do something.

## 6.5 extProgramStatus

`extProgramStatus` checks the status of the program, to determine if it is still running and to release the program handle.

## 6.6 extTagEnumerate and extTagRead

`extTagEnumerate` and `extTagRead` are used to read the tags written by `rtcuexttag.exe`, to make the application able to choose which module to load or to validate that it is the expected module.

# 7 Module API Functions

This section describes all the functions etc. that can be called from extension modules. To use the functions, the module must link against the libmodule.so library.

These functions can NOT be called from extension programs.

## 7.1 Structures

### 7.1.1.1 vplFunctionEntry

Function table entry.

**Parameters:**

| Name | | Description |
| --- | --- | --- |
| fname | - | The name of the function. Used in VPL to identify the function |
| func | - | Pointer to the function in module. |

**Declaration:**

```
struct vplFunctionEntry {
      const char* fname
      vpl_function_call func
}
```

# 7.2 Definitions and Macros

## 7.2.1 Enumerations

Enumeration data types.

### *7.2.1.1 rtcu_io_signals*

IO Signals that can be controlled using rtcuSetIOSignal()
Note that not all signals are available on all targets.

| Name | | Description |
|---|---|---|
| RF_OFF | - | Control power to the optional RF module. Set to 0 to turn the power on. |
| RF_CFG | - | Control the CFG pin on the RF module. |
| SER0_EN | - | Enable serial port 0 if available. |
| SER1_EN | - | Enable serial port 1 if available. |
| SER2_EN | - | Enable serial port 2 if available. |
| SER3_EN | - | Enable serial port 3 if available. |
| SER4_EN | - | Enable serial port 4 if available (reserved) |
| SER5_EN | - | Enable serial port 5 if available (reserved) |
| SER6_EN | - | Enable serial port 6 if available (reserved) |
| SER7_EN | - | Enable serial port 7 if available (reserved) |
| SER8_EN | - | Enable serial port 8 if available (reserved) |
| CAN0_EN | - | Enable CAN interface 0 if available.<br>*Available since: 1.20* |
| CAN1_EN | - | Enable CAN interface 1 if available.<br>*Available since: 1.20* |

#### 7.2.1.2 rtcu_fw_types

Firmware types.

**Available since:** 2.00

| Name | | Description |
|---|---|---|
| FW_TYPE_SYSTEM | - | System firmware. |
| FW_TYPE_RUNTIME | - | Runtime firmware. |
| FW_TYPE_MONITOR | - | Monitor firmware. |

### 7.2.2 Declarations

#### 7.2.2.1 MODDECL

Define to include in the declaration of functions, in case the calling conventions change.

**Declaration:**
*Empty*

### 7.2.3 Data types

These defines provide platform independent data types.

#### 7.2.3.1 int8

Signed 8 bit integer, corresponding to SINT in VPL.

**Declaration:**
`signed char`

#### 7.2.3.2 int16

Signed 16 bit integer, corresponding to INT in VPL.

**Declaration:**
`signed short`

#### 7.2.3.3 int32

Signed 32 bit integer, corresponding to DINT in VPL.

**Declaration:**
`signed long`

#### 7.2.3.4 uint8

Unsigned 8 bit integer.

**Declaration:**
`unsigned char`

#### 7.2.3.5 uint16

Unsigned 16 bit integer.

**Declaration:**
`unsigned short`

#### 7.2.3.6 uint32

Unsigned 32 bit integer.

**Declaration:**
`unsigned long`

#### 7.2.3.7 HANDLE

Handle to internal system data.

**Declaration:**
`void`

### 7.2.4 SOS Flags

Flags for the SOS objects.

These flags can be combined bitwise.

#### 7.2.4.1 SOS_FLAG_NONE

No special flags.

**Declaration:**
`0x00`

#### 7.2.4.2 SOS_FLAG_RO

Read-only object. The object can not be written by external systems, such as the RTCU IDE.

**Declaration:**
`0x01`

#### 7.2.4.3 SOS_FLAG_CRYPT

The object is encrypted. Number and boolean objects can not be encrypted.

**Declaration:**
```
0x02
```

### 7.2.5 Notification events

Event IDs for notifications sent to moduleNotify

#### 7.2.5.1 EVENT_HALT

The system is halting.

**Declaration:**
```
1
```

#### 7.2.5.2 EVENT_RESET

The system is resetting.

**Declaration:**
```
2
```

#### 7.2.5.3 EVENT_SHUTDOWN

The system is shutting down.

**Declaration:**
```
3
```

#### 7.2.5.4 EVENT_POWERFAIL

External power has disappeared/has appeared again. Parm is 1 when the power disappears and 0 when it appears again.

**Declaration:**
```
4
```

#### 7.2.5.5 EVENT_POWERSAVE

The system is entering/leaving pmWaitEvent. Parm is 1 when entering pmWaitEvent and 0 when leaving.

**Declaration:**
```
5
```

## 7.2.6 Alignment macros

Macros for working with unaligned data

### 7.2.6.1 ALIGN_ATTR

Attribute to set alignment.

**Declaration:**
```
__attribute__ ((packed, aligned(2)))
```

### 7.2.6.2 PACKED_ATTR

**Declaration:**
```
__attribute__ ((packed))
```

### 7.2.6.3 read16b

Macro for unaligned read of a 16 bit value.

**Parameters:**

| Name | | Description |
|---|---|---|
| addr | - | The address to read from |

**Returns:**
The value read from the address

**Declaration:**
```
((uint16)(((((uint8*)(addr))[1]<<8) | ((uint8*)(addr))[0])))
```

### 7.2.6.4 read24b

Macro for unaligned read of a 24 bit value.

**Parameters:**

| Name | | Description |
|---|---|---|
| addr | - | The address to read from |

**Returns:**
The value read from the address

**Declaration:**
```
((uint32)((((uint8*)(addr))[2]<<16) | (((uint8*)(addr))[1]<<8) | (((uint8*)
(addr))[0])))
```

### 7.2.6.5 read32b

Macro for unaligned read of a 32 bit value.

**Parameters:**

| Name | | Description |
|---|---|---|
| addr | - | The address to read from |

**Returns:**
The value read from the address

**Declaration:**
```
((uint32)((((uint8*)(addr))[3]<<24) | (((uint8*)(addr))[2]<<16) | (((uint8*)
(addr))[1]<<8) | (((uint8*)(addr))[0])))
```

### 7.2.6.6 write16b

Macro for unaligned write of a 16 bit value.

**Parameters:**

| Name | | Description |
|---|---|---|
| addr | - | The address to write to |
| val | - | The value to write |

**Declaration:**
```
{ ((uint8*)(addr))[0] = ((val)&0xff); ((uint8*)(addr))[1] = ((val)>>8)&0xff; }
```

### 7.2.6.7 write24b

Macro for unaligned write of a 24 bit value.

**Parameters:**

| Name | | Description |
|---|---|---|
| addr | - | The address to write to |
| val | - | The value to write |

**Declaration:**
```
{ ((uint8*)(addr))[0] = ((val)&0xff); ((uint8*)(addr))[1] = ((val)>>8)&0xff;
((uint8*)(addr))[2] = ((val)>>16)&0xff; }
```

### 7.2.6.8 write32b

Macro for unaligned write of a 32 bit value.

**Parameters:**

| Name | | Description |
|---|---|---|
| addr | - | The address to write to |
| val | - | The value to write |

**Declaration:**

```
{ ((uint8*)(addr))[0] = ((val)&0xff); ((uint8*)(addr))[1] = ((val)>>8)&0xff;
((uint8*)(addr))[2] = ((val)>>16)&0xff; ((uint8*)(addr))[3] =
((val)>>24)&0xff; }
```

## 7.3 Type definitions

### 7.3.1.1 vpl_function_call

Function footprint definition.

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | pCPU | - | The system handle. |
| In | pBase | - | Pointer to structure with function parameters. |

**Returns:**
Return value to pass on to VPL.

**Declaration:**
```
typedef int32(MODDECL * vpl_function_call) (HANDLE *pCPU, void *pbase)
```

### 7.3.1.2 capture_handle

Audio capture callback, for use with vplAudioStreamRegister().
When this callback is called, the module must fill the buffer with the data to play and store the data size in size.

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | id | - | The ID of the stream to provide data for. |
| In | buffer | - | The buffer to place the data in. |
| In | buffer_size | - | The maximum size of data to write to the buffer. |
| Out | size | - | The actual amount of data written to the buffer. |

**Returns:**

| Value | | Description |
|---|---|---|
| 0 | - | Success. |
| <0 | - | Error, currently ignored. |

**Declaration:**
```
typedef int(* capture_handle) (int16 id, char *buffer, uint32 buffer_size,
uint32 *size)
```

#### 7.3.1.3 playback_handle

Audio playback callback, for use with vplAudioStreamRegister().
When this callback is called, the module must use the data provided in the buffer and return.

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | id | - | The ID of the stream to provide data for. |
| In | buffer | - | The buffer containing the audio data. |
| In | size | - | The amount of data in the buffer. |

**Returns:**

| Value | | Description |
|---|---|---|
| 0 | - | Success. |
| <0 | - | Error, currently ignored. |

**Declaration:**

```
typedef int(* playback_handle) (int16 id, char *data, uint32 size)
```

# 7.4 Functions

The following functions can be called from the Extension Module to interact with the system.

## 7.4.1 Basic functions

### *7.4.1.1 vplInstallFunctions*

This function must be called from moduleInit to register the functions in the system, making it possible to call the functions from VPL.
It is possible to call this function multiple times, to install functions under multiple module names or to dynamically choose which functions to install.

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | modname | - | The name of the module to install the function for. The first character must be a letter or underscore, the remaining characters may also contain digits. Maximum length is 20 characters. |
| In | ftable | - | Pointer to the Function table of the module. The table must be terminated with an empty entry, where both fname and func are NULL. |

**Returns:**

| Value | | Description |
|---|---|---|
| 0 | - | Success. |
| 1 | - | A function with the same name already exists. |
| 2 | - | Invalid module name. |
| 3 | - | Modules are not supported. |

**Declaration:**

```
uint16 MODDECL vplInstallFunctions(const char modname[16], const
vplFunctionEntry *ftable)
```

### 7.4.1.2 rtcuDebug

Send a debug message to Device output in a connected IDE.
This function can be used for debugging the module, by printing out strategic locations and variables.

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | message | - | The text to send. |

**Returns:**
None.

**Declaration:**
```
void MODDECL rtcuDebug(const char *message)
```

### 7.4.1.3 rtcuDebugF

Send a formatted debug message to Device output in a connected IDE.
The formatting is identical to printf. This function can be used for debugging the module, by printing out strategic locations and variables.

**Available since:** 1.20

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | message | - | The text to send. |
| In | ... | - | Formatting arguments for message. |

**Returns:**
None.

**Declaration:**
```
void MODDECL rtcuDebugF(const char *message,...) __attribute__((format(printf
```

## 7.4.2 String functions

The following functions are used to move string variables between VPL and C.

### 7.4.2.1 vplStringGet

Reads a string from VPL and returns a pointer to it.
A pointer to an empty string is returned if the string handle is not valid.

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | cpu | - | The handle to the system. |
| In | stringid | - | The handle to the string. |

**Returns:**
A pointer to the string.

**Declaration:**
```
const char* MODDECL vplStringGet(HANDLE *cpu, uint16 stringid)
```

### 7.4.2.2 vplStringMake

Create a new VPL string from the provided C string.

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | cpu | - | The handle to the system. |
| In | text | - | The text to store in the new VPL string. |

**Returns:**
Handle to the new string.

**Declaration:**
```
uint16 MODDECL vplStringMake(HANDLE *cpu, const char *text)
```

#### 7.4.2.3 vplStringUpdate

Used to update existing VPL strings by releasing the old string and returning a handle to the new string that should be used instead.
Mainly for use with ACCESS variables.

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | cpu | - | The handle to the system. |
| In | strid | - | The handle to the old VPL string. It is invalid after this call. |
| In | text | - | The text to store in the new VPL string. |

**Returns:**
Handle to the new VPL string.

**Declaration:**
```
uint16 MODDECL vplStringUpdate(HANDLE *cpu, uint16 strid, const char *text)
```

#### 7.4.2.4 vplStringRefInc

Increment the reference count for a string.
Should only be used when it is necessary to keep a string around after the function returns. Use vplStringRefDec() to decrement the string reference when it is no longer needed, to avoid leaking strings.

**Available since:** 2.00

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | cpu | - | The handle to the system. |
| In | strid | - | The handle to the VPL string to increment the reference to. |

**Returns:**

| Value | | Description |
|---|---|---|
| 0 | - | Success |

**Declaration:**
```
int MODDECL vplStringRefInc(HANDLE *cpu, uint16 strid)
```

### 7.4.2.5 vplStringRefDec

Decrement the reference count for a string.
Must be used on strings returned from a callback and on strings where the reference was manually incremented using vplStringRefInc().

**Available since:** 2.00

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | cpu | - | The handle to the system. |
| In | strid | - | The handle to the VPL string to decrement the reference to. It should be considered invalid after this call. |

**Returns:**

| Value | | Description |
|---|---|---|
| 0 | - | Success |

**Declaration:**
```
int MODDECL vplStringRefDec(HANDLE *cpu, uint16 strid)
```

## 7.4.3 Time functions

The following functions are used to work with the time.

### 7.4.3.1 rtcuClockGet

Retrieves the current system time.

**Returns:**
The current time as linsec.

**Declaration:**
```
int32 MODDECL rtcuClockGet(void)
```

### 7.4.3.2 rtcuClockTimeToLinsec

Converts a time provided as the individual elements to a linsec.

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | year | - | The year. (1980..2048) |
| In | month | - | The month. (1..12) |
| In | day | - | The day. (1..31) |
| In | hour | - | The hour. (0..23) |
| In | minute | - | The minute. (0..59) |
| In | second | - | The second. (0..59) |

**Returns:**
The time as linsec.

**Declaration:**
```
int32 MODDECL rtcuClockTimeToLinsec(int16 year, int8 month, int8 day, int8 hour,
int8 minute, int8 second)
```

### 7.4.3.3 rtcuClockLinsecToTime

Convert linsec to the individual time elements.

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | linsec | - | The linsec time |
| Out | year | - | The year. |
| Out | month | - | The month. |
| Out | day | - | The day. |
| Out | hour | - | The hour. |
| Out | minute | - | The minute. |
| Out | second | - | The second. |

**Returns:**
None.

**Declaration:**
```
void MODDECL rtcuClockLinsecToTime(int32 linsec, int16 *year, int8 *month, int8
*day, int8 *hour, int8 *minute, int8 *second)
```

### 7.4.3.4 rtcuClockTMToLinsec

Convert a struct tm to linsec.

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | time | - | Pointer to a struct tm with the time. |

**Returns:**
The time as linsec.

**Declaration:**
```
int32 MODDECL rtcuClockTMToLinsec(struct tm *time)
```

### 7.4.3.5 rtcuClockLinsecToTM

Convert linsec to a struct tm.

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | linsec | - | The linsec time |
| Out | time | - | Pointer to a struct tm with the time. |

**Returns:**
None.

**Declaration:**
```
void MODDECL rtcuClockLinsecToTM(int32 linsec, struct tm *time)
```

## 7.4.4 SOS functions

The following functions are used to work with the System Object Storage, which can be used for storing settings for the module.

The functions provide the same functionality as the VPL functions, but with the added support for using encrypted objects and objects that can not be changed from the IDE.

### *7.4.4.1 rtcuSosDataCreate*

Create a data object in the SOS table.

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | name | - | The name of the object. |
| In | data | - | The data of the object. |
| In | len | - | The number of bytes in the data. |
| In | flags | - | The flags for the object. |
| In | desc | - | A short description of the object. (max 80 characters + null terminator) |
| In | limit | - | The maximum number of bytes allowed in the data. |

**Returns:**

| Value | | Description |
|---|---|---|
| 0 | - | Success. |
| -1 | - | Error. |
| -2 | - | Illegal parameter. |

**Declaration:**

```
int MODDECL rtcuSosDataCreate(const char *name, char *data, int len, int flags,
char *desc, int limit)
```

#### 7.4.4.2 rtcuSosDataUpdate

Update a data object in the SOS table.

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | name | - | The name of the object. |
| In | data | - | The data of the object. |
| In | len | - | The number of bytes in the data. |

**Returns:**

| Value | | Description |
|---|---|---|
| 0 | - | Success. |
| -1 | - | Error. |
| -2 | - | Illegal parameter. |
| -10 | - | Object is not found. |
| -11 | - | Object is not a data value. |

**Declaration:**

```
int MODDECL rtcuSosDataUpdate(const char *name, char *data, int len)
```

#### 7.4.4.3 rtcuSosDataGet

Read a data object in the SOS table.

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | name | - | The name of the object. |
| In | max_len | - | The maximum number of bytes that can be read. |
| Out | data | - | The data of the object. |
| Out | len | - | The number of bytes in the data. |
| Out | flags | - | The flags for the object. |
| Out | desc | - | A short description of the object. (max 80 characters + null terminator) |
| Out | limit | - | The maximum number of bytes allowed in the data. |

**Returns:**

| Value | | Description |
|---|---|---|
| 0 | - | Success. |
| -1 | - | Error. |
| -2 | - | Illegal parameter. |
| -10 | - | Object is not found. |
| -11 | - | Object is not a data value. |
| -12 | - | Data buffer is too small. |

**Declaration:**

```
int MODDECL rtcuSosDataGet(const char *name, char *data, int max_len, int *len,
int *flags, char *desc, int *limit)
```

### 7.4.4.4 rtcuSosStringCreate

Create a string object in the SOS table.

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | name | - | The name of the object. |
| In | data | - | The string data of the object. |
| In | flags | - | The flags for the object. |
| In | desc | - | A short description of the object. (max 80 characters + null terminator) |
| In | limit | - | The maximum string length allowed. |

**Returns:**

| Value | | Description |
|---|---|---|
| 0 | - | Success. |
| -1 | - | Error. |
| -2 | - | Illegal parameter. |

**Declaration:**

```
int MODDECL rtcuSosStringCreate(const char *name, const char *data, int flags,
char *desc, int limit)
```

### 7.4.4.5 rtcuSosStringUpdate

Update a string object in the SOS table.

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | name | - | The name of the object. |
| In | data | - | The string data of the object. |

**Returns:**

| Value | | Description |
|---|---|---|
| 0 | - | Success. |
| -1 | - | Error. |
| -2 | - | Illegal parameter. |
| -10 | - | Object is not found. |
| -11 | - | Object is not a string value. |

**Declaration:**

```
int MODDECL rtcuSosStringUpdate(const char *name, const char *data)
```

### 7.4.4.6 rtcuSosStringGet

Read a string object in the SOS table.

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | name | - | The name of the object. |
| In | max_len | - | The maximum number of bytes that can be read. |
| Out | data | - | The data of the object. |
| Out | flags | - | The flags for the object. |
| Out | desc | - | A short description of the object. (max 80 characters + null terminator) |
| Out | limit | - | The maximum number of bytes allowed in the string. |

**Returns:**

| Value | | Description |
|---|---|---|
| 0 | - | Success. |
| -1 | - | Error. |
| -2 | - | Illegal parameter. |
| -10 | - | Object is not found. |
| -11 | - | Object is not a string value. |
| -12 | - | Data buffer is too small. |

**Declaration:**

```
int MODDECL rtcuSosStringGet(const char *name, char *data, int max_len, int
*flags, char *desc, int *limit)
```

#### 7.4.4.7 rtcuSosFloatCreate

Create a float object in the SOS table.

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | name | - | The name of the object. |
| In | data | - | The float data of the object. |
| In | flags | - | The flags for the object. |
| In | desc | - | A short description of the object. (max 80 characters + null terminator) |
| In | min_data | - | The manimum value allowed. |
| In | max_data | - | The maximum value allowed. |

**Returns:**

| Value | | Description |
|---|---|---|
| 0 | - | Success. |
| -1 | - | Error. |
| -2 | - | Illegal parameter. |

**Declaration:**

```
int MODDECL rtcuSosFloatCreate(const char *name, float data, int flags, char
*desc, float min_data, float max_data)
```

### 7.4.4.8 rtcuSosFloatUpdate

Update a float object in the SOS table.

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | name | - | The name of the object. |
| In | data | - | The value to store. |

**Returns:**

| Value | | Description |
|---|---|---|
| 0 | - | Success. |
| -1 | - | Error. |
| -2 | - | Illegal parameter. |
| -10 | - | Object is not found. |
| -11 | - | Object is not a float value. |

**Declaration:**

```
int MODDECL rtcuSosFloatUpdate(const char *name, float data)
```

### 7.4.4.9 rtcuSosFloatGet

Read a float object in the SOS table.

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | name | - | The name of the object. |
| Out | data | - | The value of the object. |
| Out | flags | - | The flags for the object. |
| Out | desc | - | A short description of the object. (max 80 characters + null terminator) |
| Out | min_data | - | The manimum value allowed. |
| Out | max_data | - | The maximum value allowed. |

**Returns:**

| Value | | Description |
|---|---|---|
| 0 | - | Success. |
| -1 | - | Error. |
| -2 | - | Illegal parameter. |
| -10 | - | Object is not found. |
| -11 | - | Object is not a float value. |
| -12 | - | Data buffer is too small. |

**Declaration:**

```
int MODDECL rtcuSosFloatGet(const char *name, float *data, int *flags, char
*desc, float *min_data, float *max_data)
```

### 7.4.4.10 rtcuSosIntCreate

Create an integer object in the SOS table.

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | name | - | The name of the object. |
| In | data | - | The value to store. |
| In | flags | - | The flags for the object. |
| In | desc | - | A short description of the object. (max 80 characters + null terminator) |
| In | min_data | - | The manimum value allowed. |
| In | max_data | - | The maximum value allowed. |

**Returns:**

| Value | | Description |
|---|---|---|
| 0 | - | Success. |
| -1 | - | Error. |
| -2 | - | Illegal parameter. |

**Declaration:**

```
int MODDECL rtcuSosIntCreate(const char *name, int data, int flags, char *desc,
int min_data, int max_data)
```

### *7.4.4.11 rtcuSosIntUpdate*

Update an integer object in the SOS table.

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | name | - | The name of the object. |
| In | data | - | The value to store. |

**Returns:**

| Value | | Description |
|---|---|---|
| 0 | - | Success. |
| -1 | - | Error. |
| -2 | - | Illegal parameter. |
| -10 | - | Object is not found. |
| -11 | - | Object is not a integer value. |

**Declaration:**

```
int MODDECL rtcuSosIntUpdate(const char *name, int data)
```

### 7.4.4.12 rtcuSosIntGet

Read an integer object in the SOS table.

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | name | - | The name of the object. |
| Out | data | - | The value of the object. |
| Out | flags | - | The flags for the object. |
| Out | desc | - | A short description of the object. (max 80 characters + null terminator) |
| Out | min_data | - | The manimum value allowed. |
| Out | max_data | - | The maximum value allowed. |

**Returns:**

| Value | | Description |
|---|---|---|
| 0 | - | Success. |
| -1 | - | Error. |
| -2 | - | Illegal parameter. |
| -10 | - | Object is not found. |
| -11 | - | Object is not a integer value. |

**Declaration:**

```
int MODDECL rtcuSosIntGet(const char *name, int *data, int *flags, char *desc,
int *min_data, int *max_data)
```

### 7.4.4.13 rtcuSosBoolCreate

Create a boolean object in the SOS table.

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | name | - | The name of the object. |
| In | data | - | The value to store. |
| In | flags | - | The flags for the object. |
| In | desc | - | A short description of the object. (max 80 characters + null terminator) |

**Returns:**

| Value | | Description |
|---|---|---|
| 0 | - | Success. |
| -1 | - | Error. |
| -2 | - | Illegal parameter. |

**Declaration:**

```
int MODDECL rtcuSosBoolCreate(const char *name, int data, int flags, char *desc)
```

### 7.4.4.14 rtcuSosBoolUpdate

Update a boolean object in the SOS table.

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | name | - | The name of the object. |
| In | data | - | The value to store. |

**Returns:**

| Value | | Description |
|---|---|---|
| 0 | - | Success. |
| -1 | - | Error. |
| -2 | - | Illegal parameter. |
| -10 | - | Object is not found. |
| -11 | - | Object is not a boolean value. |

**Declaration:**

```
int MODDECL rtcuSosBoolUpdate(const char *name, int data)
```

### 7.4.4.15 rtcuSosBoolGet

Read a boolean object in the SOS table.

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | name | - | The name of the object. |
| Out | data | - | The value of the object. |
| Out | flags | - | The flags for the object. |
| Out | desc | - | A short description of the object. (max 80 characters + null terminator) |

**Returns:**

| Value | | Description |
|---|---|---|
| 0 | - | Success. |
| -1 | - | Error. |
| -2 | - | Illegal parameter. |
| -10 | - | Object is not found. |
| -11 | - | Object is not a boolean value. |

**Declaration:**

```
int MODDECL rtcuSosBoolGet(const char *name, int *data, int *flags, char *desc)
```

### 7.4.4.16 rtcuSosDelete

Delete an object in the SOS table.

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | name | - | The name of the object. |

**Returns:**

| Value | | Description |
|---|---|---|
| 0 | - | Success. |
| -1 | - | Error. |

**Declaration:**

```
int MODDECL rtcuSosDelete(const char *name)
```

## 7.4.5 Audio functions

The following functions are used to work with audio.

The audio format is in stereo, 16bit signed, little endian, at 16kHz.
It is stored interleaved, with first the left channel followed by the right channel, i.e. the byte stream looks like this: "LLRRLLRRLLRR"

### *7.4.5.1 rtcuAudioStreamRegister*

Register an audio stream handler.
Once a stream handler has been registered, it can be accessed from VPL using the audioRoute function.

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | id | - | The ID of the stream to modify. |
| In | capt | - | The capture callback. It is called to read the next audio data. |
| In | play | - | The playback callback. It is called with the latest audio data, which must be handled. |

**Returns:**

| Value | | Description |
|---|---|---|
| 0 | - | Success. |
| -1 | - | Invalid ID. |
| -2 | - | The stream is in use. |

**Declaration:**

```
int MODDECL rtcuAudioStreamRegister(int16 id, capture_handle capt,
playback_handle play)
```

### 7.4.6 File functions

The following functions are used to work with files.

#### 7.4.6.1 rtcuFsMapFile

Map a VPL or device file name to the real file name.
Absolute and relative paths are supported. Device paths start with "dev:".
Currently supported devices:
Serial ports: "dev:ser<n>" where n = 0..3

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | dest | - | The buffer to store the file name in. |
| In | dest_size | - | The size of the buffer. |
| In | file_format | - | The name of the file to convert. Supports using sprintf formatting such as "%d". |
| In | ... | - | Formatting arguments for file_format. |

**Returns:**

| Value | | Description |
|---|---|---|
| 0 | - | Success. |
| -1 | - | Invalid format. |
| -2 | - | Dest too small. |
| -3 | - | Unknown device |
| -4 | - | Invalid filename |
| -5 | - | Invalid path |
| -6 | - | Drive not available |
| -7 | - | Busy |
| -10 | - | Out of memory |

**Declaration:**

```
int MODDECL rtcuFsMapFile(char *dest, size_t dest_size, const char
*file_format,...)
```

## 7.4.7 Certificate functions

The following functions are used to work with X509 certificates.

### 7.4.7.1 rtcuCertEnumerate

This will enumerate the certificates in the security store.

**Available since:** 1.20

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | index | - | The index. |
| In | size | - | The maximum number of bytes that can be stored in name. |
| Out | name | - | The name of the certificate. |

**Returns:**

| Value | | Description |
|---|---|---|
| >1 | - | The number of bytes required in the buffer for the name to be returned. |
| 0 | - | Success. |
| -1 | - | Certificate is not found. |
| -2 | - | Illegal parameter. |

**Declaration:**

```
int MODDECL rtcuCertEnumerate(const int index, char *name, const int size)
```

### *7.4.7.2 rtcuCertGetFilename*

This will return the full path to a certificate.
The SSL_CTX_use_certificate_chain_file function uses the filename returned in buffer

**Available since:** 1.20

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | name | - | The name of the certificate. |
| In | size | - | The maximum number of bytes that can be stored in buffer. |
| Out | buffer | - | The full path to the certificate. |

**Returns:**

| Value | | Description |
|---|---|---|
| 0 | - | Success. |
| -1 | - | Certificate is not found. |
| -2 | - | The buffer is too small. |
| -3 | - | Illegal parameter. |

**Declaration:**

```
int MODDECL rtcuCertGetFilename(const char *name, char *buffer, const int size)
```

### 7.4.7.3 rtcuCertGetCApath

This will return the path to the folder containing the CA certificates.
The path can be used by the SSL_CTX_load_verify_locations function
to specify the location used for verification.

**Available since:** 1.20

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | size | - | The maximum number of bytes that can be stored in buffer. |
| Out | buffer | - | The full path to the CA certificate folder. |

**Returns:**

| Value | | Description |
|---|---|---|
| 0 | - | Success. |
| -2 | - | The buffer is too small. |
| -3 | - | Illegal parameter. |

**Declaration:**

```
int MODDECL rtcuCertGetCApath(char *buffer, const int size)
```

### 7.4.8 License functions

The following functions are used for licensing of extension modules.

#### 7.4.8.1 rtcuLicenseCheck

This will check if a valid license is present in the RTCU device.

**Available since:** 2.00

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | id | - | The license id to check. |
| In | key | - | The public key to use when checking license. |
| In | size | - | The size of the public key in bytes. |

**Returns:**

| Value | | Description |
|---|---|---|
| 1 | - | A valid license is present. |
| -1 | - | A license is not present.. |
| -2 | - | An invalid license is present.. |

**Declaration:**

```
int MODDECL rtcuLicenseCheck(const char *id, const unsigned char *key, const int
size)
```

### 7.4.9 IO functions

The following functions are used to work with input and output signals.

#### 7.4.9.1 rtcuSetIOSignal

Use this function to set the value of raw IO signals.
See the rtcu_io_signals enumeration for a list of the available signals.

Note that not all signals are available on all targets.
Some signals, such as the serial port enable signals, may return success even when they are not present on the specific target.
This is for consistency with other targets that have such a signal.

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | io | - | The ID of the signal to set |
| In | val | - | The value to set the pin to. Set it to 1 to assert the signal and to 0 to deassert it. |

**Returns:**

| Value | | Description |
|---|---|---|
| 0 | - | Success. |
| -1 | - | Invalid signal. |

**Declaration:**

```
int MODDECL rtcuSetIOSignal(rtcu_io_signals io, int val)
```

## 7.4.10 Network functions

The following functions are used to work with network interfaces.

### 7.4.10.1 rtcuNetPresent

Use this function to determine if the given network interface is present on the device.

**Available since:** 2.00

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | iface | - | The network interface to detect. 1 = mobile, 2 = LAN, 3 = LAN2, 4 = WLAN, 5 = AP |

**Returns:**

| Value | | Description |
|---|---|---|
| 1 | - | interface is present. |
| 0 | - | interface is not present. |

**Declaration:**

```
int MODDECL rtcuNetPresent(uint8 iface)
```

### 7.4.10.2 rtcuNetConnected

Use this function to determine if the given network interface is connected.

**Available since:** 2.00

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | iface | - | The network interface to check. 0 = any interface, 1 = mobile, 2 = LAN, 3 = LAN2, 4 = WLAN, 5 = AP |

**Returns:**

| Value | | Description |
|---|---|---|
| 1 | - | interface is connected. |
| 0 | - | interface is not connected. |

**Declaration:**

```
int MODDECL rtcuNetConnected(uint8 iface)
```

### 7.4.10.3 rtcuNetAddressFromInterface

Use this function to determine the ip address of the specified network interface.
The result is ready for use with e.g. inet_ntop.

**Available since:** 2.00

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | iface | - | The network interface to get the address of. 0 = any interface, 1 = mobile, 2 = LAN, 3 = LAN2, 4 = WLAN, 5 = AP |

**Returns:**

| Value | | Description |
|---|---|---|
| The | - | address of the interface or 0 on error. |

**Declaration:**

```
struct in_addr MODDECL rtcuNetAddressFromInterface(uint8 iface)
```

## 7.4.11 Device information functions

The following functions are used to get information about the devices

### 7.4.11.1 rtcuDeviceVersion

Use this function to get the version numbers of the firmware in the device.

**Available since:** 2.00

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | type | - | The firmware type to get the version of. |
| Out | major | - | The major part of the firmware version. |
| Out | minor | - | The minor part of the firmware version. |
| Out | build | - | The build part of the firmware version. |

**Returns:**

| Value | | Description |
|---|---|---|
| 0 | - | Success. |
| -1 | - | Unknown type |

**Declaration:**

```
int MODDECL rtcuDeviceVersion(rtcu_fw_types type, int16 *major, int16 *minor,
int16 *build)
```

### 7.4.11.2 rtcuDeviceType

Use this function to get the type of the device.

**Available since:** 2.00

**Returns:**

| Value | | Description |
|---|---|---|
| >0 | - | Device type, see table in RTCU IDE help. |
| 0 | - | Error. |

**Declaration:**
```
int MODDECL rtcuDeviceType(void)
```

### 7.4.11.3 rtcuDeviceTypeName

Use this function to get the type name of the device.

**Available since:** 2.00

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| Out | title | - | The type name of the device |

**Returns:**
None.

**Declaration:**
```
void MODDECL rtcuDeviceTypeName(char title[31])
```

### 7.4.11.4 rtcuDeviceSerialNumber

Use this function to get the serial number of the device .

**Available since:** 2.00

**Returns:**
Serial number.

**Declaration:**
```
uint32 MODDECL rtcuDeviceSerialNumber(void)
```

### *7.4.11.5 rtcuDeviceGetApplication*

Use this function to get the version number and name of the running application.

**Available since:** 2.00

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | name | - | Buffer to store the application name in. Must have space for 15 characters and the terminator. |
| Out | version | - | The version number of the application. |

**Returns:**

| Value | | Description |
|---|---|---|
| 0 | - | Success. |

**Declaration:**

```
int MODDECL rtcuDeviceGetApplication(char name[16], int *version)
```

## 7.4.12 Callback functions

The following functions are used to call VPL callbacks

### 7.4.12.1 vplCallbackCall

Use this function to call a callback function.
If the callback function returns a string, it is important to call vplStringRefDec() on it to avoid leaking the string.
High priority callbacks are called directly, but can only perform very basic functionality, see the description of VPL callbacks for further details.
The retVal parameter can be used to make the callback run asynchronously when the return value is not needed.

**Available since:** 2.00

**Parameters:**

| | Name | | Description |
|---|---|---|---|
| In | pCPU | - | The system handle. |
| In | func | - | Address of the VPL function to call. |
| In | frame | - | The stack frame to call with all parameters set. |
| In | frame_size | - | The size of the frame, sizeof(frame) |
| In | highprio | - | The callback priority. Set to 1 for high priority. |
| Out | retVal | - | Return value from the function. Set to NULL to run async. |

**Returns:**

| Value | | Description |
|---|---|---|
| 0 | - | Success. |
| -1 | - | Failed to prepare call. Can be caused by an invalid parameter. |
| -2 | - | Call failed |

**Declaration:**

```
int MODDECL vplCallbackCall(HANDLE *pCPU, uint32 func, uint8 *frame, uint32
frame_size, uint8 highprio, int32 *retVal)
```

# 8 Deprecated functions

## 8.1 Renamed functions

The following functions have been renamed to make the naming more consistent. The old names are still supported for backwards compatibility, but should not be used.

| Old function | New function |
|---|---|
| vplAudioStreamRegister | rtcuAudioStreamRegister |
| vplCertEnumerate | rtcuCertEnumerate |
| vplCertGetCApath | rtcuCertGetCApath |
| vplCertGetFilename | rtcuCertGetFilename |
| vplClockGet | rtcuClockGet |
| vplClockLinsecToTime | rtcuClockLinsecToTime |
| vplClockLinsecToTM | rtcuClockLinsecToTM |
| vplClockTimeToLinsec | rtcuClockTimeToLinsec |
| vplClockTMToLinsec | rtcuClockTMToLinsec |
| vplDebug | rtcuDebug |
| vplDebugF | rtcuDebugF |
| vplFsMapFile | rtcuFsMapFile |
| vplSetIOSignal | rtcuSetIOSignal |
| vplSosBoolCreate | rtcuSosBoolCreate |
| vplSosBoolGet | rtcuSosBoolGet |
| vplSosBoolUpdate | rtcuSosBoolUpdate |
| vplSosDataCreate | rtcuSosDataCreate |
| vplSosDataGet | rtcuSosDataGet |
| vplSosDataUpdate | rtcuSosDataUpdate |
| vplSosDelete | rtcuSosDelete |
| vplSosFloatCreate | rtcuSosFloatCreate |
| vplSosFloatGet | rtcuSosFloatGet |
| vplSosFloatUpdate | rtcuSosFloatUpdate |
| vplSosIntCreate | rtcuSosIntCreate |
| vplSosIntGet | rtcuSosIntGet |
| vplSosIntUpdate | rtcuSosIntUpdate |
| vplSosStringCreate | rtcuSosStringCreate |
| vplSosStringGet | rtcuSosStringGet |
| vplSosStringUpdate | rtcuSosStringUpdate |

# 9 Included libraries

In addition to the common libraries such as uClibc and pthread, the SDK also includes the following libraries to use for accessing different protocols and algorithms.
To link against any of the libraries, please add it to the `DYNAMIC_LIBS` variable in the makefile.

## 9.1 D-Bus

The libdbus library is used to access the D-Bus message bus system. This can e.g. be used for IPC between an extension module and an extension program.
See https://dbus.freedesktop.org/ for further documentation.

## 9.2 Paho MQTT client library

The libpaho-mqtt3 MQTT client libraries can be used to communicate with MQTT servers.
See https://eclipse.org/paho/clients/c/ for further documentation.

## 9.3 Mosquitto MQTT client library

The libmosquitto MQTT client library can be used to communicate with MQTT servers.
See https://mosquitto.org/man/libmosquitto-3.html for further documentation.

## 9.4 json-c JSON library

The json-c library can be used to handle JSON objects.

See https://github.com/json-c/json-c/ for further documentation.

## 9.5 Jansson JSON library

The jansson library can be used to handle JSON objects.

See http://www.digip.org/jansson/ for further documentation.

## 9.6 Libcurl

libcurl is a multiprotocol file transfer library that supports a large number of transfer protocols including FTP, HTTP(S), SMTP and SMB.

See https://curl.haxx.se/libcurl/ for further documentation.

## 9.7 OpenSSL

The OpenSSL library general purpose cryptographic library with support for the TLS and SSL protocols.

See https://www.openssl.org/ for further documentation.

## 9.8 LibGcrypt

The libgrypt library is a general purpose cryptographic library.

See https://gnupg.org/software/libgcrypt/index.html for further documentation.

## 9.9 libuuid

The libuuid library can be used to generate unique identifiers.

See https://linux.die.net/man/3/libuuid for further documentation.

## 9.10 libiconv

The libiconv library is used to convert text between different character encodings including Unicode.
See https://www.gnu.org/software/libiconv/ for further documentation.

## 9.11 zlib

The zlib library has support for data compression.

See https://zlib.net/ for further documentation.

## 9.12 libbzip2

The libbz2 library has support for compressing and decompressing data in the bzip2 format.

See https://www.sourceware.org/bzip2/ for further documentation.

## 9.13 PCRE1 - Perl Compatible Regular Expressions

The libpcre library adds support for regular expression pattern matching using the same syntax and semantics as Perl 5.
See https://www.pcre.org/original/doc/html/ for further documentation.

## 9.14 Mini-XML

The libmxml library is a tiny XML library that can be used to read and write XML files.

See https://www.msweet.org/mxml/ for further documentation.

## 9.15 SQLite

The libsqlite3 library implements a self-contained SQL database engine.
See https://www.sqlite.org/ for further documentation.