

RTCU Communication Deployment Package

Version 7.00

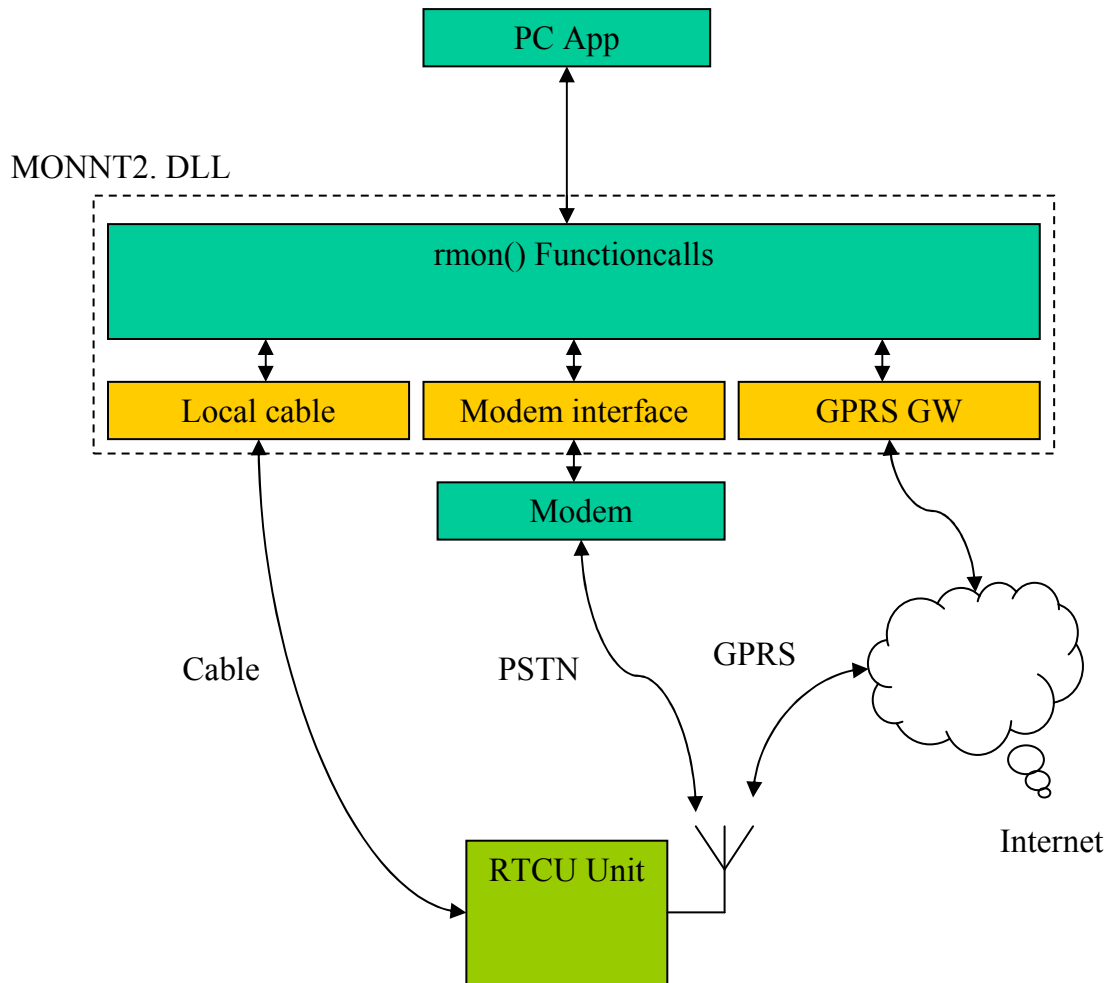




Table of Contents

| | |
|--|-----------|
| Introduction | 6 |
| Graphic illustration of the library | 7 |
| Contents of package | 8 |
| Interface and state diagram | 9 |
| Initializing the communication. | 10 |
| Entering the Idle State. | 10 |
| The Local/Remote Connected State. | 10 |
| Closing or changing the connection. | 10 |
| Functions in the MONNT2.DLL library | 11 |
| Return codes | 11 |
| Initialization/Configuration | 12 |
| rmonOpen() | 12 |
| rmonClose() | 12 |
| rmonSetModemInit() | 12 |
| rmonSetRemoteBaudrate() | 13 |
| rmonSetGWParameters() | 14 |
| rmonSetComport() | 15 |
| rmonConnect() | 15 |
| rmonDisconnect() | 16 |
| rmonAuthenticate() | 16 |
| rmonConnected() | 17 |
| Program/Firmware upload | 18 |
| rmonFirmwareUpload() | 18 |
| rmonFirmwareStartUpload() | 19 |
| rmonFirmwareResumeUpload() | 20 |
| rmonApplicationUpload() | 21 |
| rmonApplicationStartUpload() | 22 |
| rmonApplicationResumeUpload() | 23 |
| rmonVoiceUpload() | 24 |
| rmonNumOfVoiceMessages() | 24 |
| Manipulation of Persistent memory | 25 |
| rmonReadPersistentFRAM() | 25 |
| rmonWritePersistentFRAM() | 26 |



| | |
|---|-----------|
| rmonReadPersistentFLASH() | 27 |
| rmonWritePersistentFLASH() | 28 |
| rmonGetXFLASHSize() | 28 |
| rmonReadPersistentXFLASH() | 29 |
| rmonWritePersistentXFLASH() | 30 |
| Datalogger | 31 |
| rmonLogFirst() | 31 |
| rmonLogLast() | 31 |
| rmonLogReadExt() | 32 |
| rmonLogGetValuesPerRecord() | 33 |
| rmonLogClear() | 33 |
| rmonLogGotoLinsec() | 34 |
| rmonLogReadByTag() | 35 |
| rmonLogSeek() | 36 |
| I/O system functions | 37 |
| rmonReadIOMemory() | 37 |
| rmonWriteIOMemory() | 38 |
| rmonGetIOState() | 39 |
| rmonSetIOState() | 40 |
| Real time clock | 41 |
| rmonGetRTC() | 41 |
| rmonSetRTC() | 42 |
| GSM/SMS functions | 43 |
| rmonGetIMEI(), rmonGetIMSI(), rmonGetICCID() | 43 |
| rmonSendSMS() | 44 |
| rmonReceiveSMS() | 45 |
| rmonReceiveSMSEnable() | 45 |
| rmonGetGSMConnected() | 46 |
| rmonSetAllowedCallerList() | 46 |
| rmonGetAllowedCallerList() | 47 |
| rmonSetGSMPIN() | 48 |
| rmonGetGSMPIN() | 49 |
| Filesystem functions | 50 |
| rmonMediaPresent() | 51 |
| rmonMediaWriteprotected() | 51 |
| rmonMediaOpen() | 52 |
| rmonMediaClose() | 52 |
| rmonMediaQuickformat () | 53 |
| rmonMediaEject() | 53 |



| | |
|------------------------------------|-----------|
| rmonFSStatusLED() | 54 |
| rmonDirCreate() | 54 |
| rmonDirChange() | 55 |
| rmonDirCurrent() | 55 |
| rmonDirCatalog() | 56 |
| rmonDirDelete() | 56 |
| rmonFileCreate() | 57 |
| rmonFileOpen() | 57 |
| rmonFileExists() | 58 |
| rmonFileRename() | 58 |
| rmonFileDelete() | 59 |
| rmonFileStatus() | 59 |
| rmonFileGetInfo() | 60 |
| rmonFileSeek() | 60 |
| rmonFilePosition() | 61 |
| rmonFileRead() | 61 |
| rmonFileReadString() | 62 |
| rmonFileWrite() | 62 |
| rmonFileWriteString() | 63 |
| rmonFileWriteStringNL() | 63 |
| rmonFileClose() | 64 |
| rmonFileFlush() | 64 |
| Misc. functions | 65 |
| rmonReset() | 65 |
| rmonHalt() | 65 |
| rmonGetSerialNumber() | 66 |
| rmonIsUnitProgrammable() | 66 |
| rmonVer() | 66 |
| rmonGetTargetInfo() | 67 |
| rmonSetPassword() | 68 |
| rmonReceiveDebugMsg() | 68 |
| rmonGetDebugEnabled() | 69 |
| rmonSetDebugEnabled() | 69 |
| rmonVoiceMessagesAbove64K() | 70 |
| rmonGetAppInfo() | 70 |
| rmonGetGPRSSettings() | 71 |
| rmonSetGPRSSettings() | 72 |
| rmonGetGatewaySettings() | 73 |
| rmonSetGatewaySettings() | 74 |
| rmonFaultLogClear() | 74 |



| | |
|--|-----------|
| rmonFaultLogRead() | 75 |
| rmonSoftwareUpgrade() | 76 |
| <i>Appendix A, simple application</i> | 77 |
| <i>Appendix B, RTCUPROG 5.03 application</i> | 80 |



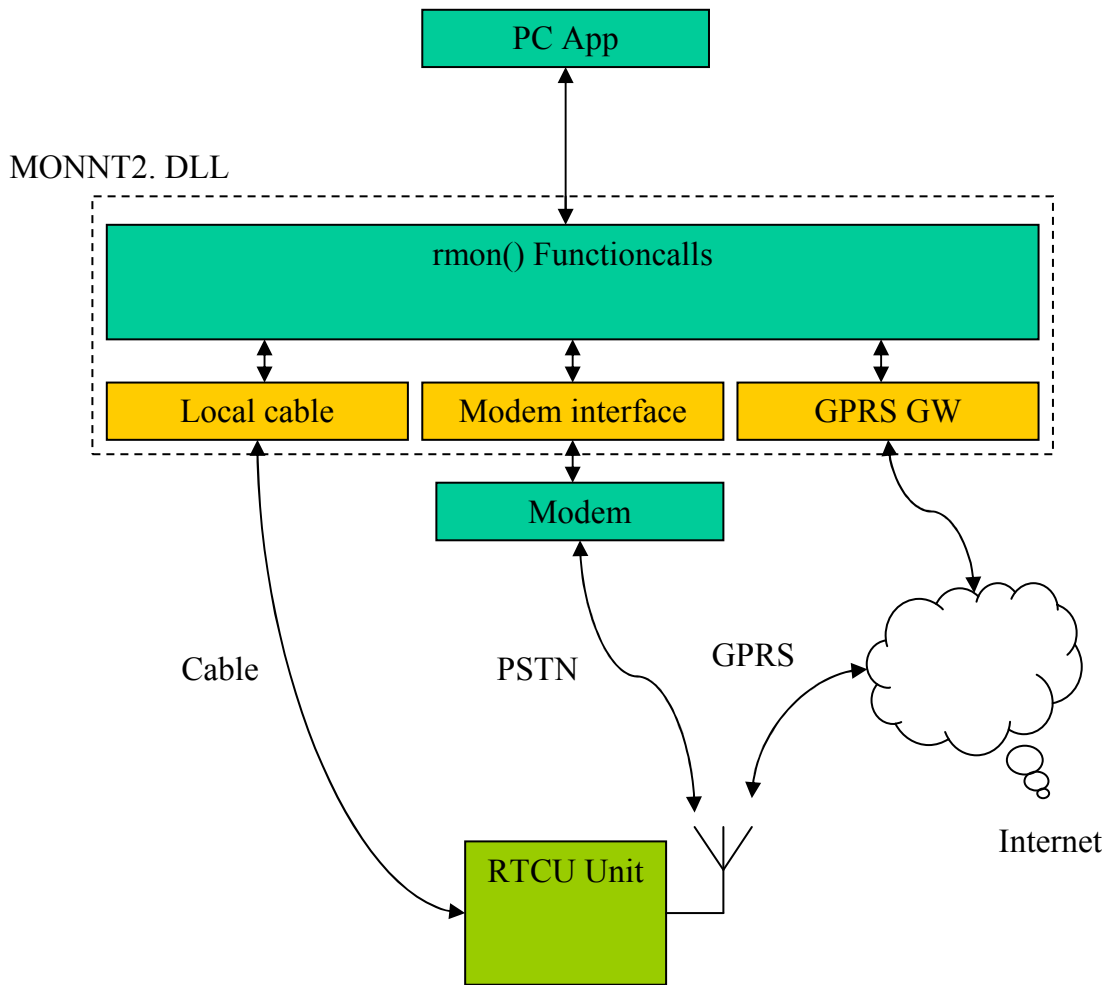
Introduction

This document discusses the library MONNT2.DLL, which enables you to establish communication with the Logic IO RTCU products. The library is a collection of functions, which will enable your own application to “talk with” the RTCU units, transfer new applications to them, upload new firmware, manipulate persistent memory, and in general perform a lot of operations, which are also accessible from within the RTCU-IDE Integrated Development Environment. The library allows you to connect to an RTCU unit thru 3 different means, the simplest being a direct cable connection. When remote connection to an RTCU unit is needed, there are two possible solutions, either thru a normal telephone modem (data call), or using TCP/IP connection via the RTCU GPRS Gateway to the RTCU unit.

Please note that if you are interested in more information about the GPRS Gateway, you should download the “GPRS Gateway Deployment Package” which is also available at the Logic IO website.

At the end of this document, you will find appendix A and appendix B. Appendix A will show you how to make a very simple basic application, mainly it shows you how to connect to a RTCU unit, and perform some simple operations. Appendix B is a complete application; this is the RTCU-PROG application. The RTCU-PROG application allows you to upload new RTCU projects and new firmware files to a RTCU unit, either thru a cable connection, thru a modem, or thru the GPRS Gateway. The application shows all the different aspects of establishing and maintaining an application with an RTCU unit, authentication and so on. Both applications will give a good hands-on experience to the library, and can also function as a good starting point for you own applications.

Graphic illustration of the library





Contents of package

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

“\RTCU Communication Deployment Package. pdf”
“\RTCUPROG 4.50\”
“\Library\”

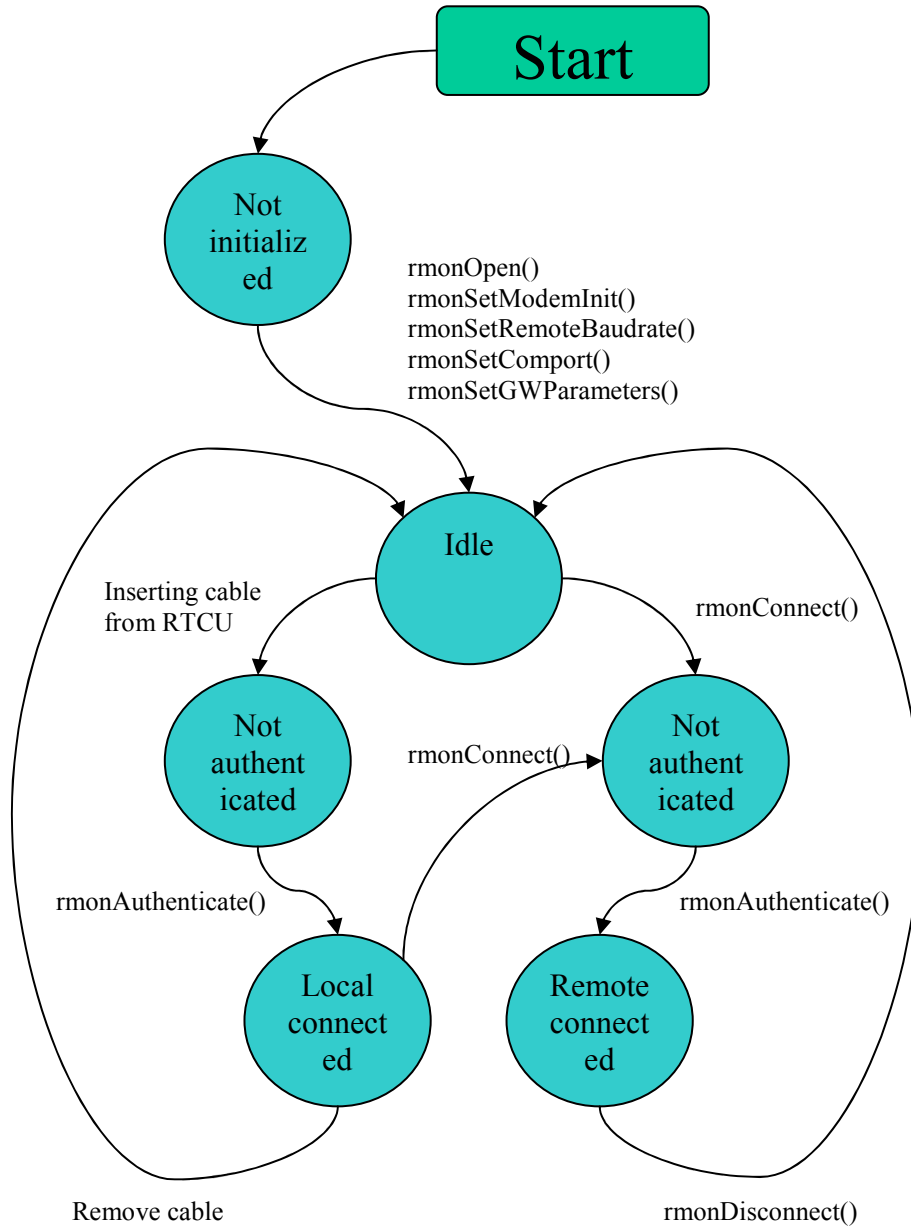
This document
The RTCUPROG application described in appendix B
The .H, .LIB and .DLL files needed for the MONNT
library. (Please note that the MONNT2.DLL also uses
other DLL's, these are included in the “Library” folder
also)

In order to use this library, you will need to use Microsoft Visual Studio C++ 6.00. For an example of which Settings etc are needed, please have a look at the RTCUPROG application included in this package.

Note: Due to a major change in the architecture of this library a re-linking with the new LIB file included is necessary. Also there has been a renaming of some of the DLL's in this library.

Interface and state diagram

To be able to go into details about using the interface it is necessary to know the different states in which the communication protocol can be:





Initializing the communication.

To begin with you are in the **Not Initialized State**. In order to proceed from there you will have to carry out the following two steps:

- **Step A.**
You will have to **prepare and open the communication system** with `rmonSetComport()`, `rmonOpen()` etc.
- **Step B.**
Then determine **what kind of connection** you want to be started.
 - Should it be a local cable connection?
 - Should it be a data call (CSD) modem connection?
 - Should it be a GPRS Gateway connection?

Entering the Idle State.

After the communication system is initialized, the **Idle State** is entered. From there it is possible to connect to a RTCU unit either thru

- A local cable connection or
- A remote connection – (modem (CSD) or GPRS Gateway connection)

Either inserting a cable between the PC and RTCU or `rmonConnect()` will enter the **Not Authenticated State**. From the **Not Authenticated State** carry out the `rmonAuthenticate()` which will put the library into either the **Locally Connected State** or the **Remotely Connected State**.

The Local/Remote Connected State.

The communication is now up running and your PC application can now communicate with the RTCU unit using the functions described.

Closing or changing the connection.

A. The Locally Connected State:

As you can see at the state diagram above you have **two possibilities** being in a **Locally Connected State**. When you want to leave this state you can either

- **Close the communication** with `rmonClose()` – you will then enter the **Not Initialised State**, or you can
- Use `rmonConnect()` followed by `rmonAuthenticate()` **which will bring you into the Remotely Connected State**.

B. The Remotely Connected State:

From a **Remotely Connected State** you can either close the connection (using `rmonClose()`), thereby bringing you to the **Not Initialised State**, or you can call `rmonDisconnect()` which will bring you into the **Idle State**



Functions in the MONNT2.DLL library

In the following description of the different function calls, we will differentiate between large and small RTCU units, as some of the functions are only supported on the large models. The large models currently includes the M11 Series, M10 Series and A9i, and small models are D4, A5, A6, DIN, SA and M7. Please consult Logic IO's website for up-to-date information on new products and their availability.

Return codes

The return codes from most of the functions will be one of those shown below. These return codes are declared in the header file (rtcumon_public.h) for the library as an "enum struct rmonRet".

| Symbolic name | Value | Description |
|---------------------|-------|--|
| rmonOK | 0 | Operation OK |
| rmonError | 1 | General error |
| rmonComError | 2 | Communication error |
| rmonTargetError | 3 | Other error in unit than rmonComError |
| rmonNoData | 4 | No data |
| rmonNoMoreData | 5 | No more data |
| rmonNotInnit | 6 | Not initialized |
| rmonDenied | 7 | Access to unit denied |
| rmonIllegalFile | 8 | File specified is illegal (corrupt/non existent) |
| rmonOldFormat | 9 | Old firmware file format, not supported |
| rmonIllegalTarget | 10 | Target and type of firmware file does not match |
| rmonNoMonitorMode | 11 | Not able to enter monitor mode |
| rmonErrorReset | 12 | Not able to reset RTCU |
| rmonErrorHalt | 13 | Not able to halt RTCU |
| rmonNotProgrammable | 14 | Attempt to program a Micro unit with a VSX file |
| rmonNoBackground | 15 | Background upload not supported by RTCU unit. |
| rmonInterrupted | 16 | Background upload was interrupted. |
| rmonCancelled | 17 | Data transfer has been cancelled. |
| rmonFileNotFound | 18 | Application or Firmware file not found. |



Initialization/Configuration

Functions that are usually needed before any real communication can be started with the RTCU unit are listed here. For the proper sequence of calling the functions, please refer to the State diagram, and to the demo applications delivered as part of this package (see appendix A and appendix B).

rmonOpen()

RTCU model: Large & Small
Called in: Not Initialised and Idle State

Synopsis int __stdcall rmonOpen(void)

Description Opens and initializes the communication system. The system can be closed again (thereby freeing the serial ports used) by calling rmonClose().

Returns rmonOK or rmonError

rmonClose()

RTCU model: Large & Small
Called in: Locally and Remotely Connected State

Synopsis int __stdcall rmonClose(void)

Description Closes communication system. The communication system must be opened with rmonOpen() in order for it to be used again.

Returns rmonOK or rmonError

rmonSetModemInit()

RTCU model: Large & Small
Called in: Not Initialised and Idle State

Synopsis int __stdcall rmonSetModemInit(const char* atcmd)

Description Specifies initialisation string to modem for usage in remote connection. A list of common initialization string for different types of modems, can be seen in the RTCU-IDE Integrated Development Environment (menu: Settings -> Setup).

Input

| | |
|--------------|---------------------------------|
| Atcmd | Initialisation string for modem |
|--------------|---------------------------------|

Returns rmonOK



rmonSetRemoteBaudrate()

RTCU model: Large & Small
Called in: Not Initialised and Idle State

Synopsis `int __stdcall rmonSetRemoteBaudrate(int baud)`

Description This function sets the baud rate that will be used when communication is established to a remote unit thru a modem. This is the baud rate used between the PC and the Modem, and has nothing to do with the speed being used on the link between the modem and RTCU unit (which can be influenced by setting the appropriate settings in the `rmpnSetModemInit()`).

Input

| | |
|------|---|
| Baud | Baud rate to be used. If baud is set to 0 a default value of 57600 baud is used. If allowed by hardware the baud rate in principle can be set to anything Commonly used baud rates are: 9600, 19200, 38400, 57600 and 115200 Other protocol parameters are: No parity, 8 data bits and 1 stop bit. |
|------|---|

Returns `rmonOK`



rmonSetGWParameters()

RTCU model: Large
Called in: Not Initialised and Idle State

Synopsis

```
Int __stdcall rmonSetGWParameters(const unsigned short Port, const unsigned
long MyNodeID, Const char* IP, const char* Key,
unsigned char gw_max_connection_attempt,
unsigned char gw_max_send_req_attempt, unsigned short gw_response_timeout,
unsigned short gw_alive_freq)
```

Description

If connection to a remote RTCU is to be done using the GPRS Gateway, some parameters has to be set before this is possible. These parameters are set using this function. Please see the online help to the RTCU-IDE Integrated Development Environment for a description of these parameters (Menu: Unit -> Communication -> Connect via GPRS Gateway).

Input

| | |
|---------------------------|--|
| Port | Gateway port (please refer to rmonConnect() for making a connection thru the GPRS Gateway) |
| MyNodeID | Node ID (can be set to 0, this will allow the Gateway to issue a "dynamic" ID to this node). |
| IP | IP address of the GPRS Gateway |
| Key | Key value (must be set to the same value as set in the GPRS Gateway) |
| gw_max_connection_attempt | Maximum connection attempts. Default value: 3. Range: 1-60 |
| gw_max_send_req_attempt | Maximum transmission attempts. Default value: 3. Range: 1-60 |
| gw_response_timeout | Response timeout. Default value: 30. Range: 5-60 |
| gw_alive_freq | Keep alive frequency. Default value: 60. Range: 0-60000 |

Returns

rmonOK



rmonSetComport()

RTCU model: Large & Small
Called in: Not Initialised and Idle State

Synopsis `int __stdcall rmonSetComport(const char* LocalPort, const char* RemotePort)`

Description The library needs to know which serial ports on the PC is going to be used for communication, both for direct cable connection, and for connection thru a modem. These ports are configured using this call. If one of the ports are not to be used, simply set it to "COM0".

Input

| | |
|------------|--|
| LocalPort | COM port to be used for direct cable connection. |
| RemotePort | COM port to be used for remote connection thru a modem |

Returns 1: Local port open failed.
 2: Remote port open failed.
 3: Local and remote port open failed.

rmonConnect()

RTCU model: Large & Small
Called in: Idle and Not Authenticated State

Synopsis `int __stdcall rmonConnect(const char* phonenumber)`

Description This function is used to establish a connection to a remote RTCU. The connection can either be thru a modem, or thru the GPRS Gateway. If the remote RTCU is to be contacted thru a modem, simply use the telephone number of the SIM card in the RTCU, if connection is thru the GPRS Gateway, the units serial number (nodeid) is to be used, prefixed with a "@" character!

Input

| | |
|-------------|---|
| phonenumber | The phone number of the SIM card in the remote RTCU if connection is thru modem, or the serial number of the RTCU (prefixed with "@") if connection is thru GPRS Gateway. |
|-------------|---|

Returns rmonOK – if the connection is established, otherwise rmonError.



rmonDisconnect()

RTCU model: Large & Small
Called in: Remotely Connected State

Synopsis int __stdcall rmonDisconnect(void)

Description Disconnect a connection to a remote RTCU unit.
 Please note that if a local RTCU is physically connected via a cable, when the remote RTCU is being disconnected, the local RTCU will be connected to the library (indicated by rmonConnected() returning RMONCON_LOCAL)

Returns rmonOK if successful disconnect, otherwise rmonError.

rmonAuthenticate()

RTCU model: Large & Small
Called in: Not Authenticated State

Synopsis rmonRet __stdcall rmonAuthenticate (int nodeid, const char password[21])

Description if there is established a (new) connection with a RTCU unit, either local or remote, the first thing to do, is for the application to authenticate itself for the RTCU unit. This is done using this function, and must be done, BEFORE any other communication with the unit can take place.
 If there is no password set in the RTCU unit, this function must still be called, just with a empty string "" as the password.

Input

| | |
|----------|--|
| nodeid | Always 1 |
| password | Password, zero terminated ASCII string |

Returns rmonComError, rmonDenied, rmonOK



rmonConnected()

RTCU model: Large & Small
Called in: All states except Not Initialised State

Synopsis int __stdcall rmonConnected(void)

Description rmonConnected() returns the type of connection that is currently (if any) established with the RTCU unit.

Returns Type of connection, see below

Type of connection:

| Symbolic name | Value | Description |
|----------------|-------|---------------------------------|
| RMONCON_NONE | 0 | Currently not connected |
| RMONCON_LOCAL | 1 | Connected using cable |
| RMONCON_REMOTE | 2 | Connected thru a modem |
| RMONCON_GW | 3 | Connected thru the GPRS Gateway |



Program/Firmware upload

The following functions are used for uploading new applications, voice messages and firmware to a RTCU unit: All functions reports their progress, by calling an optional call-back function, defined as follows:

```
typedef int (__stdcall *rmoncbprogress)(void* uptr,int elemtotal,int elemtogo);
```

Please note that the elemtotal will be a number from 0..100, indicating the percentage of the operation that is completed. Elemtogo is not used.

| | |
|-----------------------------|---|
| rmonFirmwareUpload() | RTCU model: Large & Small Called in: Locally & Remotely Connected State |
|-----------------------------|---|

Synopsis rmonRet __stdcall rmonFirmwareUpload(char *FirmwareFilename, rmoncbprogress cbfunc, void *uptr);

Description Upload firmware to RTCU.
 Function takes a .BIN firmware file, and transfers it to a RTCU unit. The function will halt execution in the unit, upload the new firmware file, and after the transfer, it will reset the unit. Note that to upload a new firmware to a RTCU unit when the connection to it is via the GPRS Gateway, you need to use background update (see rmonFirmwareStartUpload).

Input

| | |
|------------------|---|
| FirmwareFilename | Firmware file |
| cbfunc | Call-back function for progress |
| uptr | User data that will be passed to call-back function when called |

Returns rmonOK, rmonNoMonitorMode, rmonErrorReset, rmonIllegalFile, rmonOldFormat, rmonIllegalTarget, rmonComError, rmonFileNotFound



rmonFirmwareStartUpload()

RTCU model: Large
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonFirmwareStartUpload(struct rmonBGReportFW* Report, rmoncbprogress cbfunc, void* uptr);`

Description Upload firmware to RTCU.
 Function takes a .BIN firmware file, and transfers it to a RTCU unit. Unlike rmonFirmwareUpload the function will not halt execution in the unit, but start to upload the firmware in the background. The upload started with this function supports resume if the upload is interrupted. If the upload is interrupted the Report structure will contain the information needed to resume the upload using rmonFirmwareResumeUpload.
 The newly uploaded firmware is used after the unit has been reset.
 Note that the voice memory in the unit is used and any voice data must be uploaded again. Use the function rmonVoiceMessagesAbove64K to determine if the use of this function will overwrite any voice data.

Input

| | |
|--------|---|
| Report | A structure containing Firmware file and progress status (see definition below) |
| cbfunc | Call-back function for progress |
| uptr | User data that will be passed to call-back function when called |

Returns rmonOK, rmonNoBackground, rmonIllegalFile, rmonOldFormat, rmonInterrupted, rmonIllegalTarget, rmonComError, rmonFileNotFound

```
struct rmonBGReportFW{
    char* FirmwareFilename; // Firmware file
    int CodeSeg; // Code segment written
    int ConstSeg; // Const segment written
    int HeaderSeg; // Header segment written
};
```



rmonFirmwareResumeUpload()

RTCU model: Large
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonFirmwareResumeUpload(struct rmonBGReportFW* Report, rmoncbprogress cbfunc, void* uptr);`

Description Upload firmware to RTCU.
 Function takes a report containing a .BIN firmware file and progress status, and resumes where the upload was interrupted. If the upload is interrupted the Report structure will contain the information needed to resume the upload.
 Note that this function cannot be used to start a new upload, only complete an interrupted upload.
 Note that the voice memory in the unit is used and voice data must be uploaded again.

Input

| | |
|--------|---|
| Report | A structure containing Firmware file and progress status (see definition below) |
| cbfunc | Call-back function for progress |
| uptr | User data that will be passed to call-back function when called |

Returns `rmonOK, rmonNoBackground, rmonIllegalFile, rmonOldFormat, rmonInterrupted, rmonIllegalTarget, rmonComError, rmonFileNotFound`

```
struct rmonBGReportFW{
    char* FirmwareFilename; // Firmware file
    int CodeSeg; // Code segment written
    int ConstSeg; // Const segment written
    int HeaderSeg; // Header segment written
};
```



rmonApplicationUpload()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonApplicationUpload(char *VSXFilename, rmoncbprogress cbfunc, void *uptr);`

Description Uploads application to RTCU
 Function takes a .VSX or a .PSX file, and transfers it to a RTCU.
 Please note that the execution of the VPL program in the RTCU unit **must** be halted with rmonHalt(), **before** calling this function ! The RTCU will have to be reset after the transfer, to start the new application.

Input

| | |
|-------------|---|
| VSXFilename | VSX filename |
| cbfunc | Call back function for progress |
| uptr | User data that will be passed to call back function when called |

Returns rmonOK, rmonComError, rmonErrorHalt, rmonIllegalFile, rmonErrorReset, rmonNotProgrammable, rmonFileNotFound



rmonApplicationStartUpload()

RTCU model: Large
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonApplicationStartUpload(struct rmonBGReportAPP* Report, rmoncbprogress cbfunc, void *uptr);`

Description Uploads application to RTCU
 Function takes a Report containing a .VSX or a .PSX file, and transfers the file to a RTCU. The upload will be performed in the background without interfering with the running application. The upload started with this function supports resume if the upload is interrupted. If the upload is interrupted the Report structure will contain the information needed to resume the upload using `rmonApplicationResumeUpload`.
 The newly uploaded application is used after the unit has been reset.
 Note that the voice memory in the unit is used and any voice data must be uploaded again. Use the function `rmonVoiceMessagesAbove64K` to determine if the use of this function will overwrite any voice data.

Input

| | |
|--------|--|
| Report | A structure containing VSX filename and progress status (see definition below) |
| cbfunc | Call back function for progress |
| uptr | User data that will be passed to call back function when called |

Returns `rmonOK, rmonComError, rmonNoBackground, rmonIllegalFile, rmonInterrupted, rmonNotProgrammable, rmonFileNotFound`

```
struct rmonBGReportAPP{
    char* VSXFilename; // VSX filename
    int CodeSeg; // Code segment written
    int HeaderSeg; // Header segment written
};
```



rmonApplicationResumeUpload()

RTCU model: Large
Called in: Locally & Remotely
 Connected State

Synopsis rmonRet __stdcall rmonApplicationResumeUpload(struct rmonBGReportAPP* Report, rmoncbprogress cbfunc, void *uptr);

Description Uploads application to RTCU
 Function takes a report containing a .VSX or a .PSX file and progress status, and resumes where the upload was interrupted. If the upload is interrupted the Report structure will contain the information needed to resume the upload.
 Note that this function cannot be used to start a new upload, only complete an interrupted upload.
 Note that the voice memory in the unit is used and any voice data must be uploaded again.

Input

| | |
|--------|--|
| Report | A structure containing VSX filename and progress status (see definition below) |
| cbfunc | Call back function for progress |
| uptr | User data that will be passed to call back function when called |

Returns rmonOK, rmonComError, rmonErrorHalt, rmonIllegalFile, rmonErrorReset, rmonNotProgrammable, rmonFileNotFound

```
struct rmonBGReportAPP{
    char* VSXFilename; // VSX filename
    int CodeSeg; // Code segment written
    int HeaderSeg; // Header segment written
};
```



rmonVoiceUpload()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonVoiceUpload(char *ProjectFilename, rmoncbprogress cbfunc, void *uptr);`

Description Upload Voice messages to RTCU. Function transfers all voice messages associated with a RTCU project. It is important that the relative directory structure for the project is maintained as when the project was built in the RTCU-IDE environment, otherwise the function will have trouble locating the voice message files. Please note that the execution of the VPL program in the RTCU unit **must** be halted with `rmonHalt()`, **before** calling this function ! The RTCU will have to be reset after the transfer for the new voice messages to take effect.

Input

| | |
|-----------------|---|
| ProjectFilename | Project filename |
| cbfunc | Call back function for progress |
| uptr | User data that will be passed to call back function when called |

Returns `rmonOK, rmonComError, rmonErrorHalt, rmonErrorReset, rmonIllegalFile`

rmonNumOfVoiceMessages()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonNumOfVoiceMessages(char *ProjectFilename, int *NumFiles);`

Description Determine how many voice messages is included in a project. This is useful for determining if the `rmonVoiceUpload()` function has to be called when uploading a complete project to a RTCU unit.

Input

| | |
|-----------------|----------------------------------|
| ProjectFilename | Project filename |
| NumFile | Number of voice file in PRJ file |

Returns `rmonOK, rmonIllegalFile`



Manipulation of Persistent memory

The Persistent memory of the RTCU unit, can be manipulated with this set of functions. The FLASH based Persistent memory in the RTCU units, is accessible from VPL using the functions SaveData/LoadData, SaveString/LoadString. The FRAM based memory, is accessible with the functions SaveDataF / LoadDataF, SaveStringF / LoadStringF.

rmonReadPersistentFRAM()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonReadPersistentFRAM(int entry, char *data, int *length, int binary);`

Description Read FRAM based persistent entry.
 This function reads a specific entry from FRAM based Persistent memory in the RTCU. When called, you must specify what type of data you are expecting to read, if the specified type of data is not present, the function returns rmonNoData, otherwise the data and length are returned.
 Data read with this function, can be stored from VPL with the functions SaveStringF() and SaveDataF()

Input

| | |
|--------|---|
| entry | Entry number, from 1 to 20 |
| data | Buffer for data (must be large enough!) |
| length | The number of bytes read from the entry |
| binary | Set to 1 if expecting binary data, 0 if string expected |

Returns rmonNoData, rmonComError, rmonOK



rmonWritePersistentFRAM()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis rmonRet __stdcall rmonWritePersistentFRAM(int entry, char *data, int length, int binary);

Description Write to FRAM based persistent entry.
 This function writes either binary data or a string to a specific entry in the FRAM based persistent memory in the RTCU. When called, you must specify what type of data you are storing.
 Data stored with this function, can be read from VPL with the functions LoadStringF() and LoadDataF().

Input

| | |
|--------|--|
| entry | Entry number, from 1 to 20 |
| data | The data to store |
| length | Length of data |
| binary | Set to 1 if storing binary data, 0 if string |

Returns rmonComError, rmonOK



rmonReadPersistentFLASH()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis rmonRet __stdcall rmonReadPersistentFLASH(int entry, char *data, int *length, int binary);

Description Read FLASH based persistent entry
 This function reads a specific entry from FLASH based persistent memory in the RTCU. When called, you must specify what type of data you are expecting to read, if the specified type of data is not present, the function returns rmonNoData, otherwise the data and length are returned.
 Data read with this function can be stored from VPL with the functions SaveString() and SaveData().

Input

| | |
|--------|---|
| entry | Entry number, from 1 to 192 |
| data | Buffer for data (must be large enough!) |
| length | The number of bytes read from the entry |
| binary | Set to 1 if expecting binary data, 0 if string expected |
| data | Pointer to input data |

Returns rmonNoData, rmonComError, rmonOK



rmonWritePersistentFLASH()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonWritePersistentFLASH(int entry, char *data, int length, int binary);`

Description Write to FLASH based persistent entry.
 This function writes either binary data or a string to a specific entry in the FLASH based persistent memory in the RTCU. When called, you must specify what type of data you are storing.
 Data stored with this function, can be read from VPL with the functions LoadString() and LoadData().

Input

| | |
|--------|--|
| entry | Entry number, from 1 to 192 |
| data | The data to store |
| length | Length of data |
| binary | Set to 1 if storing binary data, 0 if string |

Returns `rmonComError, rmonOK`

rmonGetXFLASHSize()

RTCU model: Large
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonGetXFLASHSize(int *size);`

Description Get the number of entries in extended FLASH.
 This function is identical to the VPL function GetFlashXSize().

Output

| | |
|------|--------------------------------------|
| size | Number of entries in extended flash. |
|------|--------------------------------------|

Returns `rmonComError, rmonOK`



rmonReadPersistentXFLASH()

RTCU model: Large
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonReadPersistentXFLASH(int entry, char *data, int *length, int binary);`

Description Read extended FLASH based persistent entry
 This function reads a specific entry from extended FLASH based persistent memory in the RTCU. When called, you must specify what type of data you are expecting to read, if the specified type of data is not present, the function returns `rmonNoData`, otherwise the data and length are returned.
 Data read with this function can be stored from VPL with the functions `SaveStringX()` and `SaveDataX()`.

Input

| | |
|--------|---|
| entry | Entry number, from 1 to Size (Determined with the function <code>rmonGetXFLASHSize</code>) |
| data | Buffer for data (must be large enough!) |
| length | The number of bytes read from the entry |
| binary | Set to 1 if expecting binary data, 0 if string expected |

Returns `rmonNoData`, `rmonComError`, `rmonOK`



rmonWritePersistentXFLASH()

RTCU model: Large
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonWritePersistentXFLASH(int entry, char *data, int length, int binary);`

Description Write to extended FLASH based persistent entry.
 This function writes either binary data or a string to a specific entry in the extended FLASH based persistent memory in the RTCU. When called, you must specify what type of data you are storing.
 Data stored with this function, can be read from VPL with the functions LoadStringX() and LoadDataX().

Input

| | |
|--------|---|
| entry | Entry number, from 1 to Size (Determined with the function rmonGetXFLASHSize) |
| data | The data to store |
| length | Length of data |
| binary | Set to 1 if storing binary data, 0 if string |

Returns `rmonComError, rmonOK`



Datalogger

The built-in datalogger of the RTCU unit can be manipulated with this set of functions. The log can be read, searched and cleared etc. For a more detailed description of the datalogger in the RTCU units, please refer to the online help for the RTCU-IDE Integrated Development Environment.

rmonLogFirst()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis rmonRet __stdcall rmonLogFirst(int nodeid)

Description Moves the current read pointer to the first (oldest) entry in the datalogger.

Input

| | |
|--------|----------|
| nodeid | Always 1 |
|--------|----------|

Returns rmonComError, rmonTargetError, rmonOK

rmonLogLast()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis rmonRet __stdcall rmonLogLast(int nodeid)

Description Moves the current read pointer to the last (newest) entry in the datalogger..

Input

| | |
|--------|----------|
| nodeid | Always 1 |
|--------|----------|

Returns rmonComError, rmonTargetError, rmonOK



rmonLogReadExt()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis

rmonRet __stdcall rmonLogReadExt(int nodeid, int operation, int* values_per_rec, int* entries_in_buffer, char* buffer);

Description

This function reads up to 21 log entries from the datalogger each time it is called. When it is called, the current read pointer is either incremented or decremented, according to the operation parameter.

Input

| | |
|-----------|------------------------------------|
| nodeid | Always 1, |
| operation | RMONLOGGET_NEXT or RMONLOGGET_PREV |

Output

| | |
|-------------------|--|
| values_per_rec | Number of entries in tdefExtLog logvalues (array length), maximum 8 |
| entries_in_buffer | Number of tdefExtLog records in buffer, maximum 21 |
| buffer | Buffer containing entries in buffer records with a tdefExtLog structure, see struct definition below |

Returns

rmonComError, rmonOK, rmonNoData, rmonNoMoreData, rmonError

```
typedef struct {
    signed   char year;
    unsigned char month;
    unsigned char date;
    unsigned char hour;
    unsigned char minute;
    unsigned char second;
    unsigned char tag;
    int        logvalue[8];
} tdefExtLog;
```



rmonLogGetValuesPerRecord()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonLogGetValuesPerRecord(int nodeid, int* numberofvalues)`

Description This function returns information about how many values (up to 8) are stored at each record in the datalogger (is configure via the VPL program in the RTCU unit)

Input

| | |
|--------|----------|
| nodeid | Always 1 |
|--------|----------|

Output

| | |
|----------------|---|
| numberofvalues | The number of values stored in each record in the datalogger. |
|----------------|---|

Returns `rmonComError, rmonTargetError, rmonOK`

rmonLogClear()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonLogClear(int nodeid)`

Description Clears data in the RTCU datalogger. The current datastructure in the RTCU datalogger is maintained.

Input

| | |
|--------|----------|
| nodeid | Always 1 |
|--------|----------|

Returns `rmonComError, rmonNotlnit, rmonTargetError, rmonOK`



rmonLogGotoLinsec()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis rmonRet __stdcall rmonLogGotoLinsec(int nodeid, struct rmonLogGotoLinsecTime timestamp, unsigned char direction)

Description rmonLogGotoLinsec will search for an entry in the datalogger, that matches the specified timestamp, and if no match is found, it will select the nearest record (if any). It is possible to specify the search direction as either forward or backward.

Input

| | |
|-----------|--|
| nodeid | Always 1 |
| timestamp | Complete time of record to search for, see struct definition below |
| direction | False (0) means backwards search, True (different from 0) means forward search |

Returns rmonOK, rmonNoData, rmonComError

```
struct rmonLogGotoLinsecTime {
    short year;                              // 1980..2048
    unsigned char month;                    // 01..12
    unsigned char day;                      // 01..31
    unsigned char hour;                     // 00..23
    unsigned char minute;                  // 00..59
    unsigned char second;                  // 00..59
};
```



rmonLogReadByTag()

RTCU model: Large
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonLogReadByTag(int operation, unsigned char tag, int* values_per_rec, int* entries_in_buffer, char* buffer);`

Description This function reads up to 21 log entries from the datalogger each time it is called. When it is called, the current read pointer is either incremented or decremented, according to the operation parameter.

Input

| | |
|-----------|--|
| operation | RMONLOGGET_NEXT or RMONLOGGET_PREV |
| tag | The tag that is used to filter datalog entries |

Output

| | |
|-------------------|--|
| values_per_rec | Number of entries in tdefExtLog logvalues (array length), maximum 8 |
| entries_in_buffer | Number of tdefExtLog records in buffer, maximum 21 |
| buffer | Buffer containing entries in buffer records with a tdefExtLog structure, see struct definition below |

Returns `rmonComError, rmonOK, rmonNoData, rmonNoMoreData, rmonError`

```
typedef struct {
    signed   char year;
    unsigned char month;
    unsigned char date;
    unsigned char hour;
    unsigned char minute;
    unsigned char second;
    unsigned char tag;
    int        logvalue[8];
} tdefExtLog;
```



rmonLogSeek()

RTCU model: Large
Called in: Locally & Remotely
Connected State

Synopsis rmonRet __stdcall rmonLogSeek(short tag, int n)

Description rmonLogSeek will search for an entry in the datalogger, that matches the specified tag, and move n records from there. The n parameter determines the direction of the search.

Input

| | |
|-----|--|
| tag | The tag to search for. |
| n | The number of records to move. > 0 (zero): Seek forward. = 0 (zero): No effect. < 0 (zero): Seek Backwards. |

Returns rmonOK, rmonNoData, rmonComError



I/O system functions

This group of functions allows access to the physical in- and outputs of the RTCU unit, as well as the memory I/O system. The memory I/O system, which is 16 elements in a small RTCU, and 256 elements in a large RTCU, is accessible thru the VPL program as normal VAR_INPUT/VAR_OUTPUT variables. The variables must be configured in the RTCU-IDE job configuration to either “To memory” or “From Memory”, depending on if they are declared as VAR_INPUT or VAR_OUTPUT variables.

rmonReadIOMemory()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonReadIOMemory(int nodeid, int location, int count, int type, void *data);`

Description This function read the from the memory I/O system in the RTCU. It is possible to indicate what type of data is stored at each location read; this is to help the function minimizing communication traffic. This function was rmonScadaRead()

Input

| | |
|----------|--|
| nodeid | Always 1 |
| location | This is the start location to read from, 0 based. |
| count | Number of memory locations to read |
| type | 1=BOOL, 2=SINT, 3=INT, 4=DINT, this is the type of data to read from each location |

Output

| | |
|------|--|
| data | Data read from the RTCU will be stored in this buffer (must be ‘count’ number long, and of the same type at ‘type’ specifies (BOOL and SINT is 1 byte, INT is 2 bytes and DINT is 4 bytes) |
|------|--|

Returns `rmonComError, rmonError, rmonOK`



rmonWriteIOMemory()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonWriteIOMemory(int nodeid, int location, int count, int type, void *data);`

Description This function writes to the memory I/O system in the RTCU. It is possible to indicate what type of data is to be stored at each location; this is to help the function minimizing communication traffic. Please note that if the location(s) written to, is also used as VAR_OUTPUT variables by the VPL program in the RTCU unit, the RTCU and this function will both write to the same location, in which case the writing done by this function, will be overwritten by the RTCU unit itself (all I/O configured variables will be updated in each scan of the VPL program in the RTCU unit). This function was rmonScadaWrite()

Input

| | |
|----------|---|
| nodeid | Always 1 |
| location | This is the start location to write to, 0 based. |
| count | Number of memory locations to write |
| type | 1=BOOL, 2=SINT, 3=INT, 4=DINT |
| data | Data written to the RTCU is taken from this buffer (must be 'count' number long, and of the same type at 'type' specifies (BOOL and SINT is 1 byte, INT is 2 bytes and DINT is 4 bytes) |

Returns `rmonError, rmonComError, rmonOK`



rmonGetIOState()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonGetIOState(int nodeid, int iotype, int ioindex, int* value)`

Description This function is used to read the state of the physical in- and output signals in the RTCU unit. The 'iotype' indicates which input/output you are reading from, and 'ioindex' indicates which input- or output number you are reading.

Input

| | |
|---------|--|
| nodeid | Always 1. |
| iotype | Select the type of I/O system you want to read from, see below |
| ioindex | Valid index of IO to get value from. Starts with index 0 |

Output

| | |
|-------|-----------------------|
| value | Input or output value |
|-------|-----------------------|

Returns `rmonComError, rmonError, rmonOK`

iotype:

| Symbolic name | Value | Description |
|-------------------|-------|----------------|
| RMON_IOTYPE_DIN | 1 | Digital input |
| RMON_IOTYPE_DOUT | 2 | Digital output |
| RMON_IOTYPE_AIN | 3 | Analog input |
| RMON_IOTYPE_AOUT | 4 | Analog output |
| RMON_IOTYPE_LED | 5 | LED |
| RMON_IOTYPE_DIPSW | 6 | Dip switch |



rmonSetIOState()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis

rmonRet __stdcall rmonSetIOState(int nodeid, int iotype, int ioindex, int value)

Description

This function is used to set the status of the physical output signals in the RTCU unit. The 'iotype' indicated which Output you are writing to, and 'ioindex' indicates which output number you are writing to.
 Please note that if the output written to, is also used as a VAR_OUTPUT variable by the VPL program in the RTCU unit, the RTCU and this function will both write to the same output, in which case the writing done by this function will be overwritten by the RTCU unit itself (all I/O configured variables will be updated in each scan of the VPL program in the RTCU unit).

Input

| | |
|---------|---|
| nodeid | Always 1 |
| iotype | Select the type of output system you want to set, see below |
| ioindex | This is the number of the output, 0 based. |
| value | Value to set |

Returns

rmonComError, rmonError, rmonOK

iotype:

| Symbolic name | Value | Description |
|------------------|-------|----------------|
| RMON_IOTYPE_DOUT | 2 | Digital output |
| RMON_IOTYPE_AOUT | 4 | Analog output |
| RMON_IOTYPE_LED | 5 | LED |



Real time clock

Functions that will read and set the realtime clock in the RTCU unit.

The two functions, rmonGetRTC() and rmonSetRTC, both uses the following structure:

```
struct rmonRTCTime {
    unsigned short year;           // 2000..2048
    unsigned char  month;         // 01..12
    unsigned char  date;          // 01..31
    unsigned char  day;           // 01..07
    unsigned char  hour;          // 00..23
    unsigned char  minute;        // 00..59
    unsigned char  second;        // 00..59
};
```

rmonGetRTC()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis rmonRet __stdcall rmonGetRTC(int nodeid, struct rmonRTCTime *rtc, char *calibration)

Description Reads the real time clock on the RTCU unit. Returns the current time in a rmonRTCTime structure.

Input

| | |
|--------|----------|
| nodeid | Always 1 |
|--------|----------|

Output

| | |
|-------------|---|
| rtc | Please refer to the definition of rmonRTCTime above |
| calibration | Reserved. |

Returns rmonComError, rmonTargetError, rmonOK



rmonSetRTC()

RTCU model: Large & Small
Called in: Locally & Remotely
Connected State

Synopsis rmonRet_stdcall rmonSetRTC(int nodeid, struct rmonRTCTime *rtc, char calibration)

Description Sets the real time clock on the RTCU unit. The time is supplied in a rmonRTCTime structure.

Input

| | |
|-------------|---|
| nodeid | Always 1 |
| rtc | Please refer to the definition of rmonRTCTime above |
| calibration | Reserved, must ALWAYS be set to -1 |

Returns rmonComError, rmonTargetError, rmonOK



GSM/SMS functions

This is a set of functions, which allows you to read and set various parameters used in the RTCU unit's interaction with the GSM module.

Also the functions allow you to send and receive "fake" SMS messages to/from the RTCU unit. If the VPL program in the unit sends an sms message to phone number "9999", using the VPL function gsmSendSMS() / gsmSendPDU(), the message will be received by the library, and the message will then available thru the function rmonReceiveSMS(). The same goes for your application, it can call rmonSendSMS(), and the VPL program in the RTCU can use gsmIncomingSMS() / gsmIncomingPDU() to receive this message sent from your application. This is a very easy to use way of communicating small messages back and forth between your PC application and the VPL application of the RTCU unit.

rmonGetIMEI(),
rmonGetIMSI(),
rmonGetICCID()

RTCU model: Large & Small
Called in: Locally & Remotely
Connected State

Synopsis

```
rmonRet __stdcall rmonGetIMEI(int nodeid, char *IMEInumber, int bufsize)
rmonRet __stdcall rmonGetIMSI(int nodeid, char *IMSInumber, int bufsize)
rmonRet __stdcall rmonGetICCID(int nodeid, char *ICCIDnumber, int bufsize)
```

Description

rmonGetIMEI(), rmonGetIMSI() & rmonGetICCID() is used for fetching the IMEI number of the GSM module, or the IMSI and ICC numbers of the SIM card installed in the RTCU unit.

Input

| | |
|---------|--|
| nodeid | Always 1 |
| bufsize | Number of characters to read. If bufsize exceeds the number of characters in the IMEI, IMSI or ICC only the number of characters present will be put in the output buffer. |

Output

| | |
|---------------------------------------|--|
| IMEInumber, IMSInumber or ICCIDnumber | This is where the information will be stored |
|---------------------------------------|--|

Returns

rmonComError, rmonTargetError, rmonOK



rmonSendSMS()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis

rmonRet __stdcall rmonSendSMS(int nodeid, int smstype,
 int messageLength, const char* message)

Description

The PC application can send “fake” SMS messages to the RTCU unit using this function. The RTCU will receive SMS messages with the gsmIncomingSMS() / gsmIncomingPDU(), and when the message is sent to the RTCU using this function, the .phonenumber variable in the gsmIncomingSMS() / gsmIncomingPDU() will indicate “9999” as the originator of the message.

Input

| | |
|---------------|---|
| nodeid | Always 1 |
| smstype | Type of SMS message received, see below |
| messageLength | Only used when smstype is RMONSMS_BINARY |
| message | Zero terminated AZCII string when smstype is RMONSMS_TEXT. If smstype is RMONSMS_BINARY messageLength specifies length of data in message. |

Returns

rmonComError

smstype:

| Symbolic name | Value | Description |
|----------------|-------|------------------------|
| RMONSMS_TEXT | 0 | Text based SMS message |
| RMONSMS_BINARY | 1 | Binary SMS message |



rmonReceiveSMS()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonReceiveSMS(int nodeid, int* smstype, int* dataLength, char* data)`

Description When the connected RTCU unit sends a SMS message to phonenumber "9999" using the either `gsmSendSMS()` or `gsmSendPDU()`, this function will receive these messages. Please note that this function is blocking, it will first return when a message is received.

Input

| | |
|--------|----------|
| nodeid | Always 1 |
|--------|----------|

Output

| | |
|------------|--|
| smstype | Type of SMS message received, see below |
| dataLength | Only used when smstype = RMONSMS_BINARY |
| data | Zero terminated ASCII string when smstype = RMONSMS_TEXT |

Returns `rmonComError, rmonOK`

smstype:

| Symbolic name | Value | Description |
|----------------|-------|------------------------|
| RMONSMS_TEXT | 0 | Text based SMS message |
| RMONSMS_BINARY | 1 | Binary SMS message |

rmonReceiveSMSEnable()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonReceiveSMSEnable(int nodeid, int enable)`

Description Using this function, it is possible to either enable or disable reception of the "fake" SMS messages from the RTCU unit by the function.
 Note, if disabling, the `rmonReceiveSMS()` will still block, waiting for a message to arrive.

Input

| | |
|--------|---------------------|
| nodeid | Always 1 |
| enable | 0=disable, 1=enable |

Returns `rmonError, rmonOK`



rmonGetGSMConnected()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis: `rmonRet __stdcall rmonGetGSMConnected(int nodeid, int* connected)()`

Description This functions check whether the GSM module in the RTCU unit currently is connected to a GSM base station (logged into the GSM network).
 (This function does the same as the VPL function `gsmConnected()`).

Input

| | |
|--------|----------|
| nodeid | Always 1 |
|--------|----------|

Output

| | |
|-----------|--|
| connected | 1 if connected and 0 if not connected. |
|-----------|--|

Returns `rmonComError, rmonOK`

rmonSetAllowedCallerList()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonSetAllowedCallerList (int nodeid, const char numbers[81])`

Description Sets list of allowed phone numbers that can make incoming data calls to the RTCU. Phone numbers must be separated with the “,” character.
 The list of allowed callers can also be set from the RTCU-IDE (menu: Unit -> Communication -> List of Callers).
 (This function does the same as the VPL function `gsmSetListOfCallers()`).

Input

| | |
|---------|--|
| nodeid | Always 1 |
| numbers | List of phonenumber separated by “,” character |

Returns `rmonComError, rmonError, rmonOK`



rmonGetAllowedCallerList()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis rmonRet __stdcall rmonGetAllowedCallerList(int nodeid, char numbers[81])

Description Fetches list of allowed caller numbers set in rmonSetAllowedCallerList().
 The list of allowed callers may also be fetched from the RTCU-IDE (menu:
 Unit -> Communication -> List of Callers).
 (This function does the same as the VPL function gsmGetListOfCallers()).

Input

| | |
|--------|----------|
| nodeid | Always 1 |
|--------|----------|

Output

| | |
|---------|--|
| numbers | List of allowed caller numbers separated by “,” character. |
|---------|--|

Returns rmonComError, rmonError, rmonOK



rmonSetGSMPIN()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis

rmonRet __stdcall rmonSetGSMPIN (int nodeid, const char pin[5])

Description

Sets the SIM PIN code to use for the SIM card in the RTCU unit.
 This does NOT change the PIN code on the SIM card, it simply tells the RTCU unit which PIN code to use when powering up the GSM module !

An empty string will disable use of PIN code (SIM PIN code must be disabled on the SIM card, use a normal mobile telephone for doing this)
 Specifying a wrong GSM pin code will cause a RTCU fault.

If the RTCU is restarted more than 3 times with the wrong SIM pin code, the SIM card will be locked, and it must be unlocked in a normal mobile phone, using the GSM operator supplied PUK code !

Please notice the SIM PIN code may also be set using the RTCU IDE (menu: Unit -> Setup -> Set PIN code)
 (This function does the same as the VPL function gsmSetPin()).

Input

| | |
|--------|----------------------------|
| nodeid | Always 1 |
| pin | New SIM PIN code to be set |

Returns

rmonComError, rmonError, rmonOK



rmonGetGSMPIN()

RTCU model: Large & Small
Called in: Locally & Remotely
Connected State

Synopsis rmonRet __stdcall rmonGetGSMPIN (int nodeid, const char pin[5])

Description Fetches the GSM SIM PIN code from the RTCU (see rmonSetGSMPin() above).
An empty string denotes that the PIN code has been disabled.

Input

| | |
|--------|----------|
| nodeid | Always 1 |
|--------|----------|

Output

| | |
|-----|--|
| pin | Current SIM PIN code. Empty string specifies the PIN code to be currently disabled |
|-----|--|

Returns rmonComError, rmonOK



Filesystem functions

This is a set of functions that offers a broad range of operations on the filesystem present in the RTCU unit. The error-codes are identical to the corresponding VPL functions, except it is positive and offset by 100

| Symbolic name | Value | Description |
|---------------------|-------|-------------------------------------|
| RMONFS_INVALIDDRIVE | 101 | The media is not opened |
| RMONFS_NOTFOUND | 105 | The directory or file is not found |
| RMONFS_DUPLICATED | 106 | The directory or file already exist |
| RMONFS_NOMOREENTRY | 107 | The media is full. |
| RMONFS_NOTOPEN | 108 | The file is not open |
| RMONFS_LOCKED | 112 | The file is in use |
| RMONFS_NOTEMPTY | 114 | The directory is not empty |
| RMONFS_CARDREMOVED | 116 | The media is not present. |
| RMONFS_WRITEPROTECT | 123 | The media is write-protected |

In addition to the general limitations for the filesystem (See the RTCU-IDE Online-help / Manual) the filesystem transactions are limited by the following:

1. A client can only have one open file at a time. If more than one file is opened the already open file will be closed.
2. The working directory is shared between all clients.
It is therefore recommended to work with absolute path-names.

Note that the Filesystem API is only supported on the X32 generation of RTCU units.



rmonMediaPresent()

RTCU model: X32
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonMediaPresent(int* state, int* fserr)`

Description Queries whether the Media is present or not.

Input
None.

Output

| | |
|-------|---|
| State | =0 (zero) if media is not present. <>0 (zero) if media is present |
| Fserr | Error code from the filesystem |

Returns `rmonOK, rmonComError, rmonTargetError`

rmonMediaWriteprotected()

RTCU model: X32
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonMediaWriteprotected (int* state, int* fserr)`

Description Queries whether the Media is write protected or not.

Input
None.

Output

| | |
|-------|--|
| State | =0 (zero) if media is not write protected. <>0 (zero) if media is write protected. |
| Fserr | Error code from the filesystem |

Returns `rmonOK, rmonComError, rmonTargetError`



rmonMediaOpen()

RTCU model: X32
Called in: Locally & Remotely
Connected State

Synopsis rmonRet __stdcall rmonMediaopen (int* fserr)

Description Open the media for use with the filesystem.

Input
None.

Output

| | |
|-------|--------------------------------|
| Fserr | Error code from the filesystem |
|-------|--------------------------------|

Returns rmonOK, rmonComError, rmonTargetError

rmonMediaClose()

RTCU model: X32
Called in: Locally & Remotely
Connected State

Synopsis rmonRet __stdcall rmonMediaopen (int* fserr)

Description Close the media for use with the filesystem.

Input
None.

Output

| | |
|-------|--------------------------------|
| Fserr | Error code from the filesystem |
|-------|--------------------------------|

Returns rmonOK, rmonComError, rmonTargetError



rmonMediaQuickformat ()

RTCU model: X32
Called in: Locally & Remotely
 Connected State

Synopsis rmonRet __stdcall rmonMediaQuickformat (int* fserr)

Description Quick formats the media.

Input
 None.

Output

| | |
|-------|--|
| Fserr | Error code from the filesystem. (RMONFS_INVALIDDRIVE, RMONFS_CARDREMOVED, RMONFS_WRITEPROTECT) |
|-------|--|

Returns rmonOK, rmonComError, rmonTargetError

rmonMediaEject()

RTCU model: X32
Called in: Locally & Remotely
 Connected State

Synopsis rmonRet __stdcall rmonMediaEject(int* fserr)

Description Eject the media.

Input
 None.

Output

| | |
|-------|--|
| Fserr | Error code from the filesystem. (RMONFS_CARDREMOVED) |
|-------|--|

Returns rmonOK, rmonComError, rmonTargetError



rmonFSStatusLED()

RTCU model: X32
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonFSStatusLED(int* fserr)`

Description Using this function, it is possible to either enable or disable the filesystem status LED's.

Input

| | |
|--------|---------------------|
| Enable | 0=disable, 1=enable |
|--------|---------------------|

Output

| | |
|-------|--------------------------------|
| Fserr | Error code from the filesystem |
|-------|--------------------------------|

Returns `rmonOK, rmonComError, rmonTargetError`

rmonDirCreate()

RTCU model: X32
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonDirCreate(const char name[61], int* fserr)`

Description Create a new directory.

Input

| | |
|------|---|
| Name | Name of the directory to create. (60 characters + 0 (zero) terminator) Both absolute and relative path can be used. |
|------|---|

Output

| | |
|-------|---|
| Fserr | Error code from the filesystem. (RMONFS_INVALIDDRIVE, RMONFS_DUPLICATED, RMONFS_NOMOREENTRY, RMONFS_CARDREMOVED, RMONFS_WRITEPROTECT) |
|-------|---|

Returns `rmonOK, rmonComError, rmonTargetError`



rmonDirChange()

RTCU model: X32
Called in: Locally & Remotely
 Connected State

Synopsis rmonRet __stdcall rmonDirChange(const char path[61], int* fserr)

Description Change the working directory.

Input

| | |
|------|---|
| Path | Path to the new working directory. (60 characters + 0 (zero) terminator) Both absolute and relative path can be used. |
|------|---|

Output

| | |
|-------|--|
| Fserr | Error code from the filesystem. (RMONFS_INVALIDDRIVE, RMONFS_NOTFOUND, RMONFS_CARDREMOVED) |
|-------|--|

Returns rmonOK, rmonComError, rmonTargetError

rmonDirCurrent()

RTCU model: X32
Called in: Locally & Remotely
 Connected State

Synopsis rmonRet __stdcall rmonDirCurrent(char path[61], int* fserr)

Description Retrieve the absolute path to the working directory.

Input

None.

Output

| | |
|-------|---|
| Path | The absolute path to the working directory. (60 characters + 0 (zero) terminator) |
| Fserr | Error code from the file system. |

Returns rmonOK, rmonComError, rmonTargetError



rmonDirCatalog()

RTCU model: X32
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonDirCatalog(short index, char name[15], struct rmonRTCTime* time, long* length, int* fserr)`

Description Retrieves the information of a entry in the working directory.

Input

| | |
|-------|---|
| Index | The index of the directory entry to retrieve. |
|-------|---|

Output

| | |
|--------|---|
| Name | The name of the directory entry. (14 characters + zero terminator) |
| Time | The creation time of the file. Uses the same structure as rmonGetRTC. Not used for directories. |
| Length | The size of the file. Not used for directories. |
| Fserr | Error code from the filesystem. (RMONFS_NOTFOUND) |

Returns `rmonOK, rmonComError, rmonTargetError`

rmonDirDelete()

RTCU model: X32
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonDirDelete(const char name[61], int* fserr)`

Description Delete a directory.

Input

| | |
|------|---|
| Name | The name of the directory. (60 characters + zero terminator) Both absolute and relative path can be used. |
|------|---|

Output

| | |
|-------|--|
| Fserr | Error code from the filesystem. (RMONFS_INVALIDDRIVE, RMONFS_NOTFOUND, RMONFS_NOTEMPTY, RMONFS_CARDREMOVED, RMONFS_WRITEPROTECT) |
|-------|--|

Returns `rmonOK, rmonComError, rmonTargetError`



rmonFileCreate()

RTCU model: X32
Called in: Locally & Remotely
 Connected State

Synopsis rmonRet __stdcall rmonFileCreate(const char name[61], int* fserr)

Description Creates a new file. If a file is already open, it will be closed before the new file is created.

Input

| | |
|------|--|
| Name | The name of the file to create. (60 characters + zero terminator) Both absolute and relative path can be used. |
|------|--|

Output

| | |
|-------|--|
| Fserr | Error code from the filesystem. (RMONFS_INVALIDDRIVE, RMONFS_NOTOPEN, RMONFS_CARDREMOVED, RMONFS_WRITEPROTECT) |
|-------|--|

Returns rmonOK, rmonComError, rmonTargetError

rmonFileOpen()

RTCU model: X32
Called in: Locally & Remotely
 Connected State

Synopsis rmonRet __stdcall rmonFileOpen(const char name[61], int* fserr)

Description Opens a file. If a file is already open, it will be closed before the new file is opened.

Input

| | |
|------|--|
| Name | The name of the file to create. (60 characters + zero terminator) Both absolute and relative path can be used. |
|------|--|

Output

| | |
|-------|---|
| Fserr | Error code from the filesystem. (RMONFS_INVALIDDRIVE, RMONFS_NOTOPEN, RMONFS_CARDREMOVED) |
|-------|---|

Returns rmonOK, rmonComError, rmonTargetError



rmonFileExists()

RTCU model: X32
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonFileExists(const char name[61], char* state, int* fserr)`

Description Query whether a file exists.

Input

| | |
|------|--|
| Name | The name of the file to create. (60 characters + zero terminator) Both absolute and relative path can be used. |
|------|--|

Output

| | |
|-------|--|
| State | =0 (zero) if file does not exist. <>0 (zero) if file does exist. |
| Fserr | Error code from the filesystem. |

Returns `rmonOK, rmonComError, rmonTargetError`

rmonFileRename()

RTCU model: X32
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonFileRename(const char name_old[61], const char name_new[13], int* fserr)`

Description Renames a file

Input

| | |
|----------|--|
| Name_new | The new name of the file. (12 characters + zero terminator) |
| Name_old | The name of the file to rename. Both absolute and relative paths can be used (60 characters + zero terminator) |

Output

| | |
|-------|---|
| Fserr | Error code from the filesystem. (RMONFS_INVALIDDRIVE, RMONFS_NOTFOUND, RMONFS_DUPLICATED, RMONFS_LOCKED, RMONFS_CARDREMOVED, RMONFS_WRITEPROTECT) |
|-------|---|

Returns `rmonOK, rmonComError, rmonTargetError`



rmonFileDelete()

RTCU model: X32
Called in: Locally & Remotely
 Connected State

Synopsis rmonRet __stdcall rmonFileDelete(const char name[61], int* fserr)

Description Delete a file.

Input

| | |
|------|--|
| Name | The name of the file to create. (60 characters + zero terminator) Both absolute and relative path can be used. |
|------|--|

Output

| | |
|-------|--|
| State | =0 (zero) if file does not exist. <>0 (zero) if file does exist. |
| Fserr | Error code from the filesystem. (RMONFS_INVALIDDRIVE, RMONFS_NOTFOUND, RMONFS_LOCKED, RMONFS_CARDREMOVED, RMONFS_WRITEPROTECT) |

Returns rmonOK, rmonComError, rmonTargetError

rmonFileStatus()

RTCU model: X32
Called in: Locally & Remotely
 Connected State

Synopsis rmonRet __stdcall rmonFileStatus(int* status, int* fserr)

Description Retrieve the status of the open file.

Input

None.

Output

| | |
|--------|---|
| Status | The status of the file. Identical to the return value of the fsFileStatus VPL function. |
| Fserr | Error code from the filesystem. |

Returns rmonOK, rmonComError, rmonTargetError



rmonFileGetInfo()

RTCU model: X32
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonFileGetInfo(const char name[61], struct rmonRTCTime* time, long* length, int* fserr)`

Description Retrieve the size and creation timestamp of a file.

Input

| | |
|------|--|
| name | The name of the file. (60 characters + zero terminator) Both absolute and relative path can be used. |
|------|--|

Output

| | |
|--------|--|
| Time | The creation timestamp. Please refer to the definition of rmonRTCTime above (Page 40) |
| Length | The size of the file in bytes. |
| Fserr | Error code from the filesystem. (RMONFS_INVALIDDRIVE, RMONFS_NOTFOUND, RMONFS_CARDREMOVED) |

Returns rmonOK, rmonComError, rmonTargetError

rmonFileSeek()

RTCU model: X32
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonFileSeek(long offset, int* fserr)`

Description Moves the file pointer.

Input

| | |
|--------|---|
| Offset | The new position relative to the Start of file. >0 (zero) – Position in file. =0 (zero) – Start of file. -1 – End of file. |
|--------|---|

Output

| | |
|-------|--|
| Fserr | Error code from the filesystem. (RMONFS_NOTOPEN) |
|-------|--|

Returns rmonOK, rmonComError, rmonTargetError



rmonFilePosition()

RTCU model: X32
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonFilePosition(long* position, int* fserr)`

Description Retrieve the file pointer position of the open file.

Input
None.

Output

| | |
|----------|--|
| Position | The file pointer position |
| Fserr | Error code from the filesystem. (RMONFS_NOTOPEN) |

Returns `rmonOK, rmonComError, rmonTargetError`

rmonFileRead()

RTCU model: X32
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonFileRead(int elemcnt, char* buffer, int* elemread, int* fserr, rmoncbprogress pfunc, void* uptr)`

Description Read a block of data from file.

Input

| | |
|---------|--|
| Elemcnt | The number of bytes to read from file. |
| Pfunc | Pointer to function where progress is reported. |
| Arg | Pointer to user argument used when reporting progress. |

Output

| | |
|----------|---|
| Buffer | The buffer where the data read from the file is stored. |
| elemread | The number of bytes read from file. |
| Fserr | Error code from the filesystem. (RMONFS_NOTOPEN) |

Returns `rmonOK, rmonComError, rmonTargetError, rmonCancelled`



rmonFileReadString()

RTCU model: X32
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonFileReadString(char str[241], int* elemread, int* fserr)`

Description Reads a string from the file. The function will read until a <CR><LF> termination sequence is found or the buffer is full (240 characters).

Input
None.

Output

| | |
|----------|--|
| str | The buffer where the string read from the file is stored. (240 characters + zero terminator) |
| elemread | The number of bytes read from file. |
| Fserr | Error code from the filesystem. (RMONFS_NOTOPEN) |

Returns `rmonOK, rmonComError, rmonTargetError`

rmonFileWrite()

RTCU model: X32
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonFileWrite(int elemcnt, char* buffer, int* elemwr, int* fserr, rmoncbprogress pfunc, void* uptr)`

Description Write a block of data to file.

Input

| | |
|---------|--|
| elemcnt | The number of bytes to write to file. |
| Buffer | The buffer where the data to write is stored. |
| pfunc | Pointer to function where progress is reported. |
| uptr | Pointer to user argument used when reporting progress. |

Output

| | |
|--------|---|
| elemwr | The number of bytes written to file |
| Fserr | Error code from the filesystem. (RMONFS_NOTOPEN, RMONFS_WRITEPROTECT) |

Returns `rmonOK, rmonComError, rmonTargetError, rmonCancelled`



rmonFileWriteString()

RTCU model: X32
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonFileWriteString(const char str[241], int* elemwr, int* fserr)`

Description Write a string to file.

Input

| | |
|-----|---|
| str | The string to write to file. (240 characters + zero terminator) |
|-----|---|

Output

| | |
|--------|---|
| elemwr | The number of bytes written to file |
| Fserr | Error code from the filesystem. (RMONFS_NOTOPEN, RMONFS_WRITEPROTECT) |

Returns rmonOK, rmonComError, rmonTargetError

rmonFileWriteStringNL()

RTCU model: X32
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonFileWriteStringNL(const char str[241], int* elemwr, int* fserr)`

Description Write a string to file. <CR><LF> are appended.

Input

| | |
|-----|---|
| str | The string to write to file. (240 characters + zero terminator) |
|-----|---|

Output

| | |
|--------|---|
| elemwr | The number of bytes written to file |
| Fserr | Error code from the filesystem. (RMONFS_NOTOPEN, RMONFS_WRITEPROTECT) |

Returns rmonOK, rmonComError, rmonTargetError



rmonFileClose()

RTCU model: X32
Called in: Locally & Remotely
Connected State

Synopsis rmonRet __stdcall rmonFileClose(int* fserr)

Description Close the file.

Input
None.

Output

| | |
|-------|--|
| Fserr | Error code from the filesystem. (RMONFS_NOTOPEN) |
|-------|--|

Returns rmonOK, rmonComError, rmonTargetError

rmonFileFlush()

RTCU model: X32
Called in: Locally & Remotely
Connected State

Synopsis rmonRet __stdcall rmonFileFlush(int* fserr)

Description Flush cached write operations to media.

Input
None.

Output

| | |
|-------|--|
| Fserr | Error code from the filesystem. (RMONFS_NOTOPEN) |
|-------|--|

Returns rmonOK, rmonComError, rmonTargetError



Misc. functions

Below is a list of different “housekeeping” functions.

rmonReset()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis rmonRet __stdcall rmonReset(int nodeid)

Description This function will reset the connected RTCU unit. If the RTCU unit is remotely connected (modem or GPRS gateway) the reset will be delayed until the connection is lost (by calling rmonDisconnect() etc). However, if the connection is thru a direct cable connection, the RTCU unit executes the reset command immediately. The reset has the same effect as cycling power to the RTCU unit, the VPL program starts executing from the start again. This can also be carried out from the RTCU-IDE (menu: Unit -> Execution -> Reset)
 (This function does the same as the VPL function boardReset()).

Input

| | |
|--------|----------|
| nodeid | Always 1 |
|--------|----------|

Returns rmonComError, rmonOK

rmonHalt()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis rmonRet __stdcall rmonHalt(int nodeid)

Description Stops the currently executing VPL program in the RTCU.
 This can also be carried out from the RTCU-IDE (menu: Unit -> Execution -> Halt)
 The RTCU unit can be started again with the reset command (see above) or by cycling power.

Input

| | |
|--------|----------|
| Nodeid | Always 1 |
|--------|----------|

Returns rmonComError, rmonError, rmonOK



rmonGetSerialNumber()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis int __stdcall rmonGetSerialNumber(unsigned long *SerialNumber)

Description Returns the serial number of the connected RTCU.

Output

| | |
|--------------|------------------------------------|
| SerialNumber | The serialnumber of the RTCU unit. |
|--------------|------------------------------------|

Returns rmonOK – output parameters valid, otherwise rmonComError.

rmonIsUnitProgrammable()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis int __stdcall rmonIsUnitProgrammable(unsigned char *Programmable)

Description Checks if the connected RTCU is programmable (a MAX unit) or not programmable (a MICRO unit)

Output

| | |
|--------------|--|
| Programmable | 1 if unit is programmable (a MAX unit), 0 if not programmable (a MICRO unit) |
|--------------|--|

Returns rmonOK – output parameters valid, otherwise rmonComError.

rmonVer()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis int __stdcall rmonVer(int nodeid, int *ver)

Description Returns the Firmware version of the connected RTCU.
 Note that this function must be used to determine if RTCU unit is in monitor mode.

Input

| | |
|--------|----------|
| Nodeid | Always 1 |
|--------|----------|

Output

| | |
|-----|--|
| ver | Firmware version, always different from 0 and scaled by 100 (Version 4.66 is returned as 466). Note that a version higher than 90.00 means that the RTCU is in monitor mode. |
|-----|--|

Returns rmonOK – output parameters valid, otherwise rmonComError.



rmonGetTargetInfo()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonGetTargetInfo(int nodeid, int* targetID, int* firmwareVer);`

Description Fetches RTCU type and firmware version from the RTCU.

Input

| | |
|--------|----------|
| nodeid | Always 1 |
|--------|----------|

Output

| | |
|-------------|--|
| targetID | Please see the table below for a list of possible target ID's. |
| firmwareVer | Firmware version always different from 0 and scaled by 100 (Version 4.66 is returned as 466) |

Returns `rmonComError, rmonOK`

| Symbolic name | Value | Description |
|----------------|-------|--------------------|
| RMONTGT_ICP002 | 1 | RTCU-SA |
| RMONTGT_ICP003 | 2 | RTCU-DIN |
| RMONTGT_ICP004 | 4 | RTCU-D4 |
| RMONTGT_ICP005 | 5 | RTCU-A5 / RTCU-A5i |
| RMONTGT_ICP006 | 6 | RTCU-A6 |
| RMONTGT_ICP007 | 7 | RTCU-M7 |
| RMONTGT_ICP009 | 9 | RTCU-A9i |
| RMONTGT_ICP010 | 10 | RTCU-M10 Series |
| RMONTGT_ICP011 | 11 | RTCU-M11 Series |
| RMONTGT_ICP102 | 102 | RTCU-MX2 Series |



rmonSetPassword()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonSetPassword (int nodeid, const char password[21])`

Description Sets new password for access to RTCU.
 This is the password that is to be used in `rmonAuthenticate()`.
 An empty string will disable password protection.
 This can also be set from the RTCU-IDE (menu: Unit -> Setup -> Set Password)

Input

| | |
|----------|--|
| nodeid | Always 1 |
| password | New password to be set (zero terminated ASCII String). |

Returns `rmonComError, rmonError, rmonOK`

rmonReceiveDebugMsg()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis `rmonRet __stdcall rmonReceiveDebugMsg (int nodeid, char* msg, int maxsize)`

Description Receive any incoming Debug messages from RTCU.
 Please notice that `rmonReceiveDebugMsg` blocks and will not return before a debug message has been received (see also `rmonReceiveSMS()`).

Input

| | |
|---------|---|
| nodeid | Always 1 |
| maxsize | Maximum number of characters to receive |

Output

| | |
|-----|------------------------------------|
| msg | Buffer with received debug message |
|-----|------------------------------------|

Returns `rmonComError, rmonOK`



rmonGetDebugEnabled()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis rmonRet __stdcall rmonGetDebugEnabled(int nodeid, int* enabled)

Description Checks if Debug messages has been enabled or disabled in unit.

Input

| | |
|--------|----------|
| nodeid | Always 1 |
|--------|----------|

Output

| | |
|---------|---|
| enabled | 1 if Debug messages is enabled, 0 if Debug messages is disabled |
|---------|---|

Returns rmonComError, rmonOK

rmonSetDebugEnabled()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis rmonRet __stdcall rmonSetDebugEnabled(int nodeid, int enabled)

Description Enable or disable Debug messages from unit.

Input

| | |
|---------|---|
| nodeid | Always 1 |
| enabled | 1 to enable Debug messages, 0 to disable Debug messages |

Returns rmonComError, rmonOK



rmonVoiceMessagesAbove64K()

RTCU model: Large
Called in: Locally & Remotely
 Connected State

Synopsis rmonRet __stdcall rmonVoiceMessagesAbove64K (char *above)

Description Determine if voice messages are stored above 64k.
 This function is used to determine if voice messages will be overwritten by the rmonApplicationStartUpload function or the rmonFirmwareStartUpload function.

Output

| | |
|-------|---|
| above | 1 if Voice messages above 64k, 0 if no Voice messages above 64k |
|-------|---|

Returns rmonComError, rmonTargetError, rmonOK

rmonGetAppInfo()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis rmonRet __stdcall rmonGetAppInfo(char *Appname, int *Appver)

Description Fetches Application name and version from the RTCU.

Output

| | |
|---------|--|
| Appname | 0 (zero) terminated string containing the application name. Max. 15 characters long. |
| Appver | Application version scaled by 100 (Version 4.66 is returned as 466) |

Returns rmonComError, rmonNoData, rmonOK



rmonGetGPRSSettings()

RTCU model: Large
Called in: Locally & Remotely
 Connected State

Synopsis rmonRet __stdcall rmonGetGPRSSettings(rmonGPRSSettings* Settings);

Description This function fetches the TCP/IP settings the unit uses to connect over GPRS. The GPRS settings retrieved from the RTCU unit are identical to those retrieved with the “Fetch from RTCU” button in the RTCU IDE (Unit->GPRS->TCP/IP settings).

All the general TCP/IP parameters use a binary packed IP address (a.b.c.d) using this format:

- a bit 24..31
- b bit 16..23
- c bit 8..15
- d bit 0..7

Output

| | |
|----------|--|
| Settings | A structure containing the GPRS settings |
|----------|--|

Returns rmonComError, rmonOK, rmonNoData

```
typedef struct {
    // general TCP/IP parameters:
    unsigned long ip_address;
    unsigned long subnet_mask;
    unsigned long gateway;
    unsigned long dns_1;
    unsigned long dns_2;
    // PPP parameters:
    char username[34];
    char password[34];
    // Dialup/GPRS parameters:
    char APN[34];
    unsigned short authentication;
} rmonGPRSSettings;
```



rmonSetGPRSSettings()

RTCU model: Large
Called in: Locally & Remotely
 Connected State

Synopsis rmonRet __stdcall rmonSetGPRSSettings(rmonGPRSSettings Settings);

Description This function sets the TCP/IP settings the unit uses to connect over GPRS. This function is identical to the VPL function sockSetTCPIPParam, and the TCP/IP settings dialog (Unit->GPRS->TCP/IP settings) in the RTCU IDE.

All the general TCP/IP parameters use a binary packed IP address (a.b.c.d) using this format:

- a bit 24..31
- b bit 16..23
- c bit 8..15
- d bit 0..7

Input

| | |
|----------|--|
| Settings | A structure containing the GPRS settings |
|----------|--|

Returns rmonComError, rmonOK

```
typedef struct {
    // general TCP/IP parameters:
    unsigned long ip_address;
    unsigned long subnet_mask;
    unsigned long gateway;
    unsigned long dns_1;
    unsigned long dns_2;
    // PPP parameters:
    char username[34];
    char password[34];
    // Dialup/GPRS parameters:
    char APN[34];
    unsigned short authentication;
} rmonGPRSSettings;
```



rmonGetGatewaySettings()

RTCU model: Large
Called in: Locally & Remotely
 Connected State

Synopsis rmonRet __stdcall rmonGetGatewaySettings(rmonGWSettings* Settings);

Description This function fetches the settings the unit uses to connect to Gateway.
 The Gateway settings retrieved from the RTCU unit are identical to those retrieved in the RTCU IDE with the “Fetch from RTCU” button in the Gateway settings dialog (Unit->GPRS->Gateway settings).

Output

| | |
|-----------------|---|
| Settings | A structure containing the Gateway settings |
|-----------------|---|

Returns rmonComError, rmonOK, rmonNoData

```
typedef struct {
    // RTCU GPRS Gateway parameters:
    unsigned short gw_enabled;
    char          gw_ip[42];
    unsigned short gw_port;
    char          gw_key[10];
    char          phonenumber_sms[22];
    // advanced settings (modification not recommended):
    unsigned short max_connection_attempt;
    unsigned short max_send_req_attempt;
    unsigned short response_timeout;
    unsigned short alive_freq;
} rmonGWSettings;
```



rmonSetGatewaySettings()

RTCU model: Large
Called in: Locally & Remotely
 Connected State

Synopsis rmonRet __stdcall rmonSetGatewaySettings(rmonGWSettings Settings);

Description This function fetches the settings the unit uses to connect to Gateway.
 This function is identical to the VPL function sockSetGWParam, and the settings are identical to those written from the RTCU IDE with the "Write to RTCU" button in the Gateway settings dialog (Unit->GPRS->Gateway settings).

Input

| | |
|----------|---|
| Settings | A structure containing the Gateway settings |
|----------|---|

Returns rmonComError, rmonOK

```
typedef struct {
    // RTCU GPRS Gateway parameters:
    unsigned short gw_enabled;
    char          gw_ip[42];
    unsigned short gw_port;
    char          gw_key[10];
    char          phonenumber_sms[22];
    // advanced settings (modification not recommended):
    unsigned short max_connection_attempt;
    unsigned short max_send_req_attempt;
    unsigned short response_timeout;
    unsigned short alive_freq;
} rmonGWSettings;
```

rmonFaultLogClear()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis rmonRet __stdcall rmonFaultLogClear()

Description This function clears the fault log.
 This function is identical to the clear button in the Fault log in the RTCU IDE.

Returns rmonComError, rmonOK



rmonFaultLogRead()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis rmonRet __stdcall rmonFaultLogRead(rmonFault *Fault)

Description Fetches the Fault log from the RTCU.
 This function is identical to the fetch button in the fault log in the RTCU IDE

Output

| | |
|--------------|---|
| Fault | The function fills this structure with the fault log entries. |
|--------------|---|

Returns rmonComError, rmonError, rmonOK

```
typedef struct {
    unsigned short   year;
    unsigned char    month;
    unsigned char    date;
    unsigned char    hour;
    unsigned char    minute;
    unsigned char    second;
    unsigned char    Code;
} rmonFaultRecord;

typedef struct {
    unsigned char    NumRecords;
    unsigned char    NextIn;
    rmonFaultRecord Record[32];
} rmonFault;
```



rmonSoftwareUpgrade()

RTCU model: Large & Small
Called in: Locally & Remotely
 Connected State

Synopsis rmonRet __stdcall rmonSoftwareUpgrade(char string[35], int *res)

Description Upgrades the RTCU unit.

Input

| | |
|--------|--|
| String | 0 (zero) terminated string. The upgrade key. |
|--------|--|

Output

| | |
|-----|--------------------------------|
| res | The type of upgrade performed. |
|-----|--------------------------------|

Returns rmonComError, rmonError, rmonOK

| Value | Description |
|-------|----------------------------------|
| 0 | Not upgraded / Wrong upgrade key |
| 1 | GPRS enabled |
| 2 | Unit is programmable |
| 3 | LCD display enabled |
| 4 - 9 | Not used |
| 10 | Citect SCADA enabled |
| 11 | Web enabled |



Appendix A, simple application

```
//-----
// Small MONNT2.DLL sample program
//-----
#include <windows.h>
#include <process.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <rtcumon.h>

//-----
// Receive any incoming Debug messages from RTCU
//-----
static void thDebug(void *arg) {
    char buffer[512];
    int rc;
    for (;;) {
        // Wait for any debug messages from unit
        rc=rmonReceiveDebugMsg(1, buffer, sizeof(buffer));
        if (rc==0) {
            // Just print the debug message
            printf("Debug::[%s]\n", buffer);
        } else {
            Sleep(50);
        }
    }
}

//-----
// Callback function that will be called by rmonFirmwareUpload()
// to report progress in the upload process
//-----
static RMONCC cbfuncFW(void* uptr,int percentage, int dummy) {
    printf("Firmware upload finished: %3i\r", percentage);
    return 0;
}

//-----
// Callback function that will be called by rmonApplicationUpload()
// to report progress in the upload process
//-----
static RMONCC cbfuncApp(void* uptr,int percentage, int dummy) {
    printf("Application upload finished: %3i\r", percentage);
    return 0;
}

//-----
// Callback function that will be called by rmonVoiceUpload()
// to report progress in the upload process
//-----
static RMONCC cbfuncVoice(void* uptr,int percentage, int dummy) {
    printf("Voice upload finished: %3i\r", percentage);
    return 0;
}
```



```
//-----
// the main program
//-----
int main(int argc, char** argv) {
    int rc;

    // Select which comports to use for local and remote connections. (Use COM0 if no port is to be used)
    rmonSetComport("COM1", "COM0");

    // You could use the following to connect via modem:
    //rmonConnect("47114712");

    // Start the listener thread for incoming Debug messages
    _beginthread(thDebug, 0, NULL);

    // Wait for a connection to a RTCU unit
    while (1) {
        // Check and wait for connection (can be RMONCON_LOCAL, RMONCON_REMOTE, RMONCON_GW or RMONCON_NONE)
        if (rmonConnected() != RMONCON_NONE)
            break;
        Sleep(300);
    }
    printf("Connected to unit.\n");

    // Try to authenticate with an empty password:
    rc=rmonAuthenticate(1, "");
    switch (rc) {
        case rmonDenied: printf("Authenticate with empty password denied. Use correct password !\n");
        break;
        case rmonOK: printf("Authenticate with empty password accepted !\n"); break;
    }

    if (rc==rmonDenied) {
        printf("Not able to logon to unit !\n");
        return 1;
    }

    // Get information about the unit
    int targetid, firmwareversion;
    rmonGetTargetInfo(1, &targetid, &firmwareversion);
    printf("Target ID=%i, Firmware version=%i.%02i\n", targetid, firmwareversion/100,
firmwareversion%100);

    // Get the units serial number
    unsigned long SerialNumber;
    rmonGetSerialNumber(&SerialNumber);
    printf("Serialnumber of unit is %09i\n", SerialNumber);

    // Use this to upload new firmware to the unit:
    //printf("\nrc=%i\n", rmonFirmwareUpload("D:\\FirmwareFile.bin", cbfuncFW, (void*)0));

    // Use the following to upload a new application and voice messages:
    /*
    rmonHalt(1);
    printf("\nrc=%i\n", rmonApplicationUpload("D:\\APP\\APP.VSX", cbfuncApp, NULL));
    printf("\nrc=%i\n", rmonVoiceUpload("D:\\APP\\APP.PRJ", cbfuncVoice, NULL));
    rmonReset(1);
    */
}
```



```
printf("\n\nPress any key to end program...\n\n");
while (true) {
    if (getch())
        break;
}

return 0;
}
```



Appendix B, RTCUPROG 5.03 application

The RTCUPROG 5.02 program is a complete Microsoft Visual Studio C++ 6.00 project. This program demonstrates all aspects in making a robust application that will manage the connection to both local and remote RTCU unit. The application allows the user to upload new firmware to a unit, or upload a complete new project to the RTCU unit, including both VPL program and voice messages.