# Revision History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Notes</th>
<th>Contributors</th>
<th>Approver</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>27 Jan 2017</td>
<td>Initial version</td>
<td></td>
<td>Jonathan Kaye</td>
</tr>
<tr>
<td>1.1</td>
<td>2 Aug 2017</td>
<td>Updated to stack version 4.4.0. Included Class C functionality and various bug fixes.</td>
<td>Jonathan Kaye</td>
<td></td>
</tr>
<tr>
<td>1.2</td>
<td>11 Oct 2017</td>
<td>Updates for latest stack.</td>
<td>Colin Anderson</td>
<td></td>
</tr>
<tr>
<td>1.3</td>
<td>13 June 2018</td>
<td>Converted to new template. Fixed sub-band 8 typo</td>
<td>Seokwoo Yoon</td>
<td>Colin Anderson</td>
</tr>
</tbody>
</table>
## CONTENTS

1. Introduction .......................................................................................................................... 4  
   1.1 Documentation Overview ................................................................................................. 4  
   1.2 What Does a LoRa/ BLE Module Contain? ....................................................................... 4  
2. Transmit/Receive Sequence ................................................................................................. 5  
3. Interactive Mode Commands ............................................................................................... 6  
4. Core Language Built-in Routines ......................................................................................... 8  
5. LoRa Extensions Built-in Routines ....................................................................................... 9  
   5.1 LoRaMAC APIs .................................................................................................................. 9  
   5.2 Setting RM191 ChannelsMask ......................................................................................... 19  
   5.3 Events and Messages ....................................................................................................... 21  
6. Acknowledgements ............................................................................................................. 24  
7. Index of smartBASIC Commands ....................................................................................... 26
1 INTRODUCTION

1.1 Documentation Overview

This RM1xx LoRa Extension Functionality user guide provides detailed information on LoRa-specific smartBASIC extensions which provide a high level managed interface to the underlying LoRa device and Bluetooth stack to manage the following:

- Joining a LoRa gateway and transmitting/receiving data payload
- Link checking on LoRa connection
- Managing LoRa sleep intervals and reading chipset registers
- Events related to the above

This document deals specifically with the smartBASIC APIs relating to the LoRa functionality in the RM1xx series of modules. For other details of programming the RM1xx, see any of the following documents:

- RM1xx LoRaMAC BLE Central Extensions Guide (central BLE functions)
- RM1xx BLE Peripheral Extensions Guide (peripheral BLE functions for the RM186_PE or RM191_PE modules)
- smartBASIC Core Reference Guide (common functions across all smartBASIC modules)

It is also very important to read and understand the appropriate Interfacing with LoRaWAN document. These documents explain how the LoRa modules actually work and how the apis and events described below should be used.

All the above documents are found in the documentation tab of the RM1xx product page: http://www.lairdtech.com/products/rm1xx-lora-modules

1.2 What Does a LoRa/BLE Module Contain?

Laird’s smartBASIC-based LoRa/BLE modules are designed to provide a complete wireless processing solution and contain the following:

- A highly integrated radio with an integrated antenna (external antenna options are also available)
- BLE Physical and Link Layer
- Higher level stack
- Multiple GPIO and ADC
- Wired communication interfaces such as UART, I2C, and SPI
- A smartBASIC run-time engine
- Program-accessible flash memory which contains a robust flash file system exposing a conventional file system and a database for storing user configuration data
- Voltage regulators and brown-out detectors

For simple end devices, these modules can completely replace an embedded processing system.

The following block diagram (Figure 1) illustrates the structure of the BLE + LoRa smartBASIC module from a hardware perspective on the left and a firmware/software perspective on the right.
2 Transmit/Receive Sequence

Before looking at the various commands and events it is very important to understand that once you send a packet into the LoRa stack you are starting a predefined sequence that must be allowed to complete before you can send another packet.

In LoRaWAN an uplink packet refers to a packet transmitted by the LoRa module and a downlink packet refers to a packet transmitted by the network server/Lora Gateway.

In Class A devices, once an uplink packet is transmitted, the module waits a predefined time for a downlink packet. This packet may or may not be transmitted by the gateway, however, the module must still open up receive windows to receive it if necessary.

For unconfirmed packets this is the end of the cycle. There are no resends. The module assumes that the transmitted packet has been received at the gateway and onto the server. The server may have data to send back to the module and these should be picked up in whichever Receive window they coincide with and dealt with accordingly by the stack.

However, in the case of confirmed uplink packets the cycle is more complicated as the stack actively waits for an acknowledgement. If that acknowledgment isn’t received the stack automatically resends the packet a set number of times. The sequence only finishes after an acknowledgement is received or all the retries are attempted or an error has occurred causing the sequence to abort.

Similarly, for the JoinRequest, if the stack fails to receive a JoinAccept, it resends the JoinRequest a set number of times. Again, the sequence only finishes after a JoinAccept is received or all retries are attempted or an error has occurred.

This can be very offputting in the RM1xx because there is no indication of all that was happening in the background; it could seem that the module had frozen. However, this is not the case so it is very important not to interfere with this sequence by sending another packet to the stack.

The events that mark the end of the sequence are indicated in section 5.3 Events and Messages. It is very important to monitor all these sequence complete events and act on them accordingly.

For testing purposes, it is possible to use the LoramacSetDebug command or monitor the EVLORAMACTXDONE or LORAMACNOSYNC events. These should reassure that there is actually something happening in the background. In the case of the US/AU modules this should be quite quick as there are no duty cycle restrictions. In the case of the EU modules,
however, it is likely to be minutes between resends. However here you can use the LoramacGetOption(LORAMAC_OPT_NEXT_TX) command to check if there is another packet due for transmission.

There is a more complete description of this process in the Interfacing with LoRaWAN documents, which are available on the documentation tab of the RM1xx product page: http://www.lairdtech.com/products/rm1xx-lora-modules.

3 INTERACTIVE MODE COMMANDS

Interactive mode commands allow a host processor or terminal emulator to interrogate and control the operation of a smartBASIC-based module. Many of these emulate the functionality of AT commands. Others add extra functionality for controlling the filing system and compilation process.

Syntax

Unlike commands for AT modems, a space character must be inserted between AT, the command, and subsequent parameters. This allows the smartBASIC tokeniser to efficiently distinguish between AT commands and other tokens or variables starting with the letters AT.

Example:
AT I 3

The response to every Interactive mode command has the following form:

`<linefeed character> response text <carriage return>`

This format simplifies the parsing within the host processor. The response may be one or multiple lines. Where more than one line is returned, the last line has one of the following formats:

- `<lf>00<cr>` for a successful outcome, or
- `<lf>01<tab> hex number <tab> optional verbose explanation <cr>` for failure.

Note: In the case of the 01 response, the `<tab>optional_verbose_explanation` is missing in resource constrained platforms like the RM1xx modules. The `verbose explanation` is a constant string and since there are over 1000 error codes, these verbose strings can occupy more than 10 kilobytes of flash memory.

The hex number in the response is the error result code consisting of two digits which can be used to help investigate the problem causing the failure. Rather than provide a list of all the error codes in this manual, you can use UWTerminalX to obtain a verbose description of an error when it is not provided on a platform.

To get the verbose description, click the BASIC tab (in UWTerminalX) and, if the error value is hhhh, enter the command ER 0xhhhh and note the 0x prefix to hhhh. This is illustrated in Figure 2.

Figure 2: Optional verbose explanation
You can also obtain a verbose description of an error by highlighting the error value, right-clicking, and selecting Lookup Selected ErrorCode in the Terminal window.

If you get the text UNKNOWN RESULT CODE 0xHHHH, please contact Laird for the latest version of UWterminal.

### 3.1.1 AT+CFG

**COMMAND**

AT+CFG is used to set a non-volatile configuration key with an integer. The syntax of this command is defined in the smartBASIC Core Functionality Manual.

The following configuration key IDs are specific to the RM1xx module.

<table>
<thead>
<tr>
<th>Key ID</th>
<th>Definition</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>Class of Device</td>
<td>Sets the Class of the module. Only 2 classes are supported, A and C. 0 – Class A 1 – Not supported at the moment 2 – Class C</td>
</tr>
<tr>
<td>1001</td>
<td>Sub-band</td>
<td>Sets the channels map by means of a sub-band. There are eight sub-bands available (1-8). Please refer section 5.2, SettingRM191 ChannelsMask, for more details of this command. Must be used in conjunction with the at+cfg 1002 command below. Only relevant to the RM191 and RM191_PE series of modules.</td>
</tr>
<tr>
<td>1002</td>
<td>Channels Map select type</td>
<td>This parameter defines how the channels map is defined 0 – default (all channels are enabled) 1 – defined by the value set by at+cfg 1001 2 – defined by the value set by at+cfgex 1009 Only relevant to the RM191 and RM191_PE series of modules.</td>
</tr>
<tr>
<td>1003</td>
<td>Sequence number incremental step</td>
<td>When using personalisation the module must keep track of the sequence number over a module reset. Therefore a value is stored in the serial EEPROM that will be used to initialise the sequence number when the module resets. This value in EEPROM is incremented by this step whenever the sequence number matches the value. The default incremental step is 256 and this is the maximum possible value for the step. This reduces the amount of writes to the serial EEPROM. See the Interfacing with LoRaWAN document for more information regarding this feature, available in the documentation tab of the RM1xx Product Page.</td>
</tr>
</tbody>
</table>

### 3.1.2 AT+CFGEX

**COMMAND**

AT+CFGEX is used to set a non-volatile configuration key with a string. The syntax of this command is defined in the smartBASIC Core Functionality Manual.

The following configuration key IDs are specific to the RM1xx module.

<table>
<thead>
<tr>
<th>Key ID</th>
<th>Definition</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1009</td>
<td>ChannelsMask</td>
<td>Sets the ChannelsMask. Only valid for the RM191. See the Setting RM191 ChannelsMask section below</td>
</tr>
</tbody>
</table>
The at+cfgex command returns an invalid key error (7312) if an invalid Key id is entered or the length of the entered string is incorrect for that specific Key id.

The new config value is only available for use after a system reset.

Note: The NwkSKey, AppSKey and AppKey values are write only. These values cannot be read back using the at+cfgex xxxx? command.

Prior to firmware versions 17/18.4.1.0 the Key Ids in the table above were in the range of 1000 – 1005 instead of the new range of 1010-1015.

4 CORE LANGUAGE BUILT-IN Routines

Core Language built-in routines are present in every implementation of smartBASIC. These routines provide the basic programming functionality. They are augmented with target specific routines for different platforms which are described in the next chapter.

4.1.1 SYSINFO

FUNCTION

Returns an informational value depending on the value of the varid argument

SYSINFO (varId)

<table>
<thead>
<tr>
<th>Returns</th>
<th>INTEGER – value corresponding to the varId requested</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arguments</td>
<td>byVal varId as INTEGER</td>
</tr>
<tr>
<td>varId</td>
<td>ID</td>
</tr>
<tr>
<td>1010</td>
<td>Module version</td>
</tr>
<tr>
<td></td>
<td>100 – EU BLE Central</td>
</tr>
<tr>
<td></td>
<td>101 – US BLE Central</td>
</tr>
<tr>
<td></td>
<td>102 – AU BLE Central</td>
</tr>
<tr>
<td></td>
<td>103 – AS BLE Central</td>
</tr>
<tr>
<td></td>
<td>110 – EU BLE Peripheral</td>
</tr>
<tr>
<td></td>
<td>111 – US BLE Peripheral</td>
</tr>
<tr>
<td></td>
<td>112 - AU BLE Peripheral</td>
</tr>
<tr>
<td></td>
<td>113 - AS BLE Peripheral</td>
</tr>
<tr>
<td>Interactive Command</td>
<td>No</td>
</tr>
</tbody>
</table>

SYSINFO is a core language function.
5  **LoRa Extensions Built-in Routines**

5.1 LoRaMAC APIs

The following commands are specific to the LoRa functionality of the RM1xx.

5.1.1 LoRaMACSleepMode

**FUNCTION**

This command places the LoRa chipset in ultra-low power sleep mode. This closes the SPI driver and places the chipset in an unusable state. To put the BLE chipset into deep sleep the SystemStateSet(0) command must also be called.

**LORAMACSleepMode()**

<table>
<thead>
<tr>
<th>Returns</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
</tbody>
</table>

Interactive Command

```
rc = Loramasleepmode()
rc = SystemStateSet(0)
```

LORAMACSleepMode is an extension function.

5.1.2 LoRaMACJoin

**FUNCTION**

This command begins the Join process to connect a module to a LoRa network server. Before starting the Join process, the module reverts to its default state, specifically with respect to the data rate, transmit power and, in the case of the RM186, the default mandatory frequency channels. All appropriate network parameters must be configured using the AT+CFGEX commands prior to making this call.

There are two possible Join options:

- **OTAA** – where the RM1xx sends IDs to the server and both ends of the link use these IDs to calculate the NwkSKey and the AppSKey that are used in the encryption and decryption of the subsequent data packets.
- **Personalization** – where both the RM1xx and server are preconfigured with the NwkSKey, AppSKey keys and the DevAddr

For more information on the Join options, see the Interfacing with LoRaWAN app notes on the documentation tab of the RM1xx product page: [http://www.lairdtech.com/products/rm1xx-lo-wan-modules](http://www.lairdtech.com/products/rm1xx-lo-wan-modules)

Once the LORAMACJoin command is sent an EVLORAMACJOINING event is thrown. In the case of personalization there is no actual joining process, the module just transitions to the Joined state. However, the event is still thrown.

Then, on successful completion of the process, the EVLORAMACJOINED event is thrown.

5.1.3 LoRaMACJoin (nFlags)

| Returns | 0x0000 = Successfully started Join process  
|         | 0xnnnn = resultCode |
| Arguments: |

| nFlags | ByVal nFlags INTEGER |
|        | LORAMAC_JOIN_BY_PERSONALIZATION (0) |
|        | LORAMAC_JOIN_BY_REQUEST (1) |

Interactive Command

No
LORAMACJoin is an extension function.

5.1.4 LORAMACLlinkCheck

**FUNCTION**

This command sends a link check request upstream to the gateway. When a link check response is received from the gateway device, an EVLORAMACLlinkCHECKRESPMSG message is sent to the application. This message contains two 8-bit values. The first represents the gateway’s link margin, in dB, of the last successfully received link check request. The second represents the number of gateways that successfully received the last link check request.

**LORAMACLlinkCheck**

<table>
<thead>
<tr>
<th>Returns</th>
<th>0x0000 = Successfully sent a LoRa Link Check request message upstream</th>
<th>0xnnnn = resultCode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arguments: None</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interactive Command</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

```plaintext
// This handler is called when a Link Check Response is received
//==========================================================================
FUNCTION HandlerLoRaLinkCheckResponse(BYVAL nMargin AS INTEGER, BYVAL nGwCnt AS INTEGER) AS INTEGER
    PRINT "Link Check Response: Margin = ";nMargin;"dB  Gateway Count = ";nGwCnt;"\nendfunc 1
OnEvent EVLORAMACLlinkCHECKRESPMSG call HandlerLoRaLinkCheckResponse
... LORAMACLlinkCheck()
```

LORAMACLlinkCheck is an extension function.
5.1.5 LORAMACTxData

**FUNCTION**

This command sends data upstream to the LoRa gateway and Network Server on the specified port.

**Note:** Port 0 is reserved for MAC commands between the gateway and the end node.

### LORAMACTxData (nPort, Data$, nFlags)

| Returns | 0x0000 – Successfully queued data for upstream transmission  
|         | 0xnnnn – resultCode |

**Arguments:**

- **nPort**
  - ByVal nPort INTEGER
  - The port to be used by the transmission.

- **Data$**
  - ByRef data$ STRING
  - The data to be sent upstream.

- **nFlags**
  - ByVal nFlags INTEGER
  - Bit mask for options.
    - 0 – Do not request confirmation
    - 1 – Request confirmation

**Interactive Command**

No

```plaintext
#define LORAMAC_JOIN_BY_REQUEST             1 // Used with LORAMACJoin
#define LORAMAC_JOIN_BY_PERSONALIZATION     0 // Used with LORAMACJoin

DIM rc
DIM data$
DIM ncurrentsize, nmaxsize
DIM joined

FUNCTION HandlerLoramacJoining() As Integer
    print "\nJoining"
endfunc 1

FUNCTION HandlerLoramacJoined() As Integer
    //JoinAccept has been received. You are now free to transmit data packets
    print "\nJoined"
    joined = 1
endfunc 1

FUNCTION HandlerLoramacRxTimeout() As Integer
    // A JoinAccept has still not been received after the JoinRequest has been
    // attempted a configured number of times
    print "\nRxTimeout"
endfunc 1

FUNCTION HandlerLoramacTxTimeout() As Integer
    // Something has gone wrong so you need to resend the JoinRequest or data packet
    print "\nTxTimeout"
endfunc 1
```
FUNCTION HandlerLoramacRxError() As Integer
  // Something has gone wrong so you need to resend the JoinRequest
  print "\nRxError"
endfunc

FUNCTION HandlerLoramacTxComplete() As Integer
  // TxComplete has been received
  print "\nTxComplete"
endfunc

FUNCTION HandlerLoramacRxComplete() As Integer
  // RxComplete has been received
  print "\nRxComplete"
endfunc

FUNCTION HandlerNextTx() As Integer
  print "\nNextTx"
  if joined == 1 then
    print "\n--------------------------------------\n"
    rc = LoramacQueryTxPossible(strlen(data$), ncurrentsize, nmaxsize)
    if rc == 0 then
      print "\ncurrent ";ncurrentsize
      print "\nmmax ";nmaxsize
    else
      print "current ";ncurrentsize
    endif
    rc = LORAMACTxData(2, data$, 1)
    if rc != 0 then
      print "\nData payload too large"
    endif
  else
    rc = LORAMACJoin(LORAMAC_JOIN_BY_REQUEST)
  endif
endfunc

FUNCTION HandlerSequenceComplete(flag, netxtime) As Integer
  print "\nSequence complete ";flag
  print "\nNext time ",netxtime
endfunc

FUNCTION HandlerLoramacRxData() As Integer
  dim rxdata$
  dim nRSSI, nPort, nSNR, nFramePending, nPacketType
  rc = LORAMACRxData(rxdata$, nRSSI, nPort, nSNR, nFramePending, nPacketType)
  print "\nStatus ";nPort;"nRSSI: ";nRSSI;" SNR: ";nSNR;"\n  print "\nData ";rxdata$
endfunc

ONEVENT EVLORAMACJOINING CALL HandlerLoramacJoining
ONEVENT EVLORAMACJOINED CALL HandlerLoramacJoined
ONEVENT EVLORAMACRXTIMEOUT CALL HandlerLoramacRxTimeout
ONEVENT EVLORAMACTXTIMEOUT CALL HandlerLoramacTxTimeout
ONEVENT EVLORAMACRXERROR CALL HandlerLoramacRxError
LORAMACTxData is an extension function.

### 5.1.6 LORAMACRxData

**FUNCTION**

This function returns downlink data received from the LoRa gateway. It should only be called from the EVLORAMACRXDATA event handler.

**LORAMACRxData (stData$, pRxRSSI, pRxPort, pRxSNR, pFramePending, pPacketType)**

<table>
<thead>
<tr>
<th>Returns</th>
<th>0x0000 : Success</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arguments:</strong></td>
<td></td>
</tr>
<tr>
<td>stData$</td>
<td>ByRef stData$ STRING</td>
</tr>
<tr>
<td>The data read from the received packet.</td>
<td></td>
</tr>
<tr>
<td>pRxRSSI</td>
<td>ByRef pRxRSSI INTEGER</td>
</tr>
<tr>
<td>The RSSI value of the received packet.</td>
<td></td>
</tr>
<tr>
<td>pRxPort</td>
<td>ByRef pRxPort INTEGER</td>
</tr>
<tr>
<td>The port on which the packet is received.</td>
<td></td>
</tr>
<tr>
<td>pRxSNR</td>
<td>ByRef pRxSNR INTEGER</td>
</tr>
<tr>
<td>The SNR value of the received packet.</td>
<td></td>
</tr>
<tr>
<td>pFramePending</td>
<td>ByRef pFramePending INTEGER</td>
</tr>
<tr>
<td>0 – No downlink packets remaining in the server</td>
<td></td>
</tr>
<tr>
<td>1 – Downlink packets remaining in the server</td>
<td></td>
</tr>
<tr>
<td>pPacketType</td>
<td>ByRef pPacketType INTEGER</td>
</tr>
<tr>
<td>Indicates whether the downlink packet was a confirmed or unconfirmed packet. If it is a confirmed packet the stack will automatically send an acknowledgement in the next uplink packet.</td>
<td></td>
</tr>
<tr>
<td>3 – unconfirmed packet</td>
<td></td>
</tr>
<tr>
<td>5 – confirmed packet</td>
<td></td>
</tr>
</tbody>
</table>

**Interactive Command**

No

See LORAMACTXDATA above for an example of this functionality.

LORAMACRxData is an extension function.
5.1.7 LORAMACGetOption

**FUNCTION**

This function retrieves the value for a specified option.

**Note:** A list of options and their descriptions can be found in Table 1. RM1xx-defs.h should be included by any smartBASIC app using this function. It can be downloaded from the Laird website and will be included with every firmware release.

### LORAMACGetOption (nOptID, optValue$)

| Returns | 0x0000 – Successfully retrieved option value
| 0xnnnn – Result code |

**Arguments:**

| nOptID | ByVal nOptID INTEGER
The Option ID as defined in TargetRM1xx-defs.h |
| optValue$ | ByRef optValue$ STRING
The string in which to return the option value. |

**Interactive Command** No

```
//Example Code
dim reg
dim stringVal$
dim rc

// Retrieve the current data rate and print it
rc = LORAMACGetOption(LORAMAC_OPT_DATA_RATE, stringVal$)
print stringVal$
```

LORAMACGetOption is an extension function.

5.1.8 LORAMACSetOption

**FUNCTION**

This function sets the value for a specified option.

**Note:** Values set using this command are not persisted and are lost over a power cycle. Certain of the values listed in the LORAMAC Option List below can be set and saved using the at+cfg or at+cfgex commands above.

**Note:** A list of options and their descriptions can be found in Table 1. RM1xx-defs.h should be included by any smartBASIC app using this function. It can be downloaded from the Laird website and will be included with every firmware release.

### LORAMACSetOption (nOptID, optValue$)

| Returns | 0x0000 – Successfully set option value
| 0xnnnn – Result code |

```
//Example Code
dim reg
dim stringVal$
dim rc

// Set the current data rate
rc = LORAMACSetOption(LORAMAC_OPT_DATA_RATE, stringVal$)
print stringVal$
```
Arguments:

<table>
<thead>
<tr>
<th>nOptID</th>
<th>ByVal nOptID INTEGER</th>
<th>The Option ID as defined in TargetRM1xx-defs.h</th>
</tr>
</thead>
<tbody>
<tr>
<td>optValue$</td>
<td>ByRef optValue$ STRING</td>
<td>The value to set.</td>
</tr>
</tbody>
</table>

Interactive Command

No

//Set the data rate to 5
temp$ = "5"
rc = LORAMACSetOption(LORAMAC_OPT_DATA_RATE, temp$)

LORAMACSetOption is an extension function.

5.1.9 LORAMAC Option List

These options are available for use with the LORAMACGetOption and LORAMACSetOption (as specified in the Get and Set columns. Enumerations of these options can be found in RM1xx-defs.h.

Table 1: LORAMAC Option List

<table>
<thead>
<tr>
<th>Option</th>
<th>Option Name</th>
<th>Get</th>
<th>Set</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LORAMAC_OPT_TX_POWER</td>
<td>YES</td>
<td>YES</td>
<td>The output power in dB of the LoRa radio. The value entered must be one of the values specified in the LoRaWAN specification. For the RM186 valid values are 2, 5, 8, 11, 14 and 20dBm and for the RM191 10 to 30dBm in 2dBm steps. If an invalid value is entered, the code sets the power to the nearest value lower than the one entered. <strong>Note:</strong> The RM1xx is limited to a maximum output power of 13dBm. If a value higher than that is entered, the code automatically limits the power.</td>
</tr>
<tr>
<td>2</td>
<td>LORAMAC_OPT_DATA_RATE</td>
<td>YES</td>
<td>YES</td>
<td>The data rate of the LoRa radio. Valid values are 0 to 5 for the RM186 and 0 to 4 for the RM191.</td>
</tr>
<tr>
<td>3</td>
<td>LORAMAC_OPT_JOIN_STATE</td>
<td>YES</td>
<td>NO</td>
<td>LoRaWAN network joined attribute</td>
</tr>
<tr>
<td>4</td>
<td>LORAMAC_OPT_DEV_EUI</td>
<td>YES</td>
<td>NO</td>
<td>The device EUI assigned by Laird</td>
</tr>
<tr>
<td>5</td>
<td>LORAMAC_OPT_CUSTOM_DEV_EUI</td>
<td>YES</td>
<td>YES</td>
<td>An optional custom device EUI <em>not</em> provided by Laird</td>
</tr>
<tr>
<td>6</td>
<td>LORAMAC_OPT_DEV_ADDR</td>
<td>YES</td>
<td>NO</td>
<td>The end-device address</td>
</tr>
<tr>
<td>7</td>
<td>LORAMAC_OPT_APP_EUI</td>
<td>YES</td>
<td>YES</td>
<td>The application EUI</td>
</tr>
<tr>
<td>8</td>
<td>LORAMAC_OPT_APP_KEY</td>
<td>NO</td>
<td>YES</td>
<td>The application key</td>
</tr>
<tr>
<td>9</td>
<td>LORAMAC_OPT_VERSION</td>
<td>YES</td>
<td>NO</td>
<td>The version number of the device</td>
</tr>
<tr>
<td>Option</td>
<td>Option Name</td>
<td>Get</td>
<td>Set</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>----------------------------------</td>
<td>-----</td>
<td>------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>10</td>
<td>LORAMAC_OPT_RSSI</td>
<td>YES</td>
<td>NO</td>
<td>The Received Signal Strength Indicator of the last received packet available after an RX Complete event</td>
</tr>
<tr>
<td>11</td>
<td>LORAMAC_OPT_SNR</td>
<td>YES</td>
<td>NO</td>
<td>The Signal-to-Noise Ratio of the last received packet available after an RX Complete event.</td>
</tr>
<tr>
<td>12</td>
<td>LORAMAC_OPT_DOWNLINK_COUNTER</td>
<td>YES</td>
<td>NO</td>
<td>Number of down-link packets received</td>
</tr>
<tr>
<td>13</td>
<td>LORAMAC_OPT_UPLINK_COUNTER</td>
<td>YES</td>
<td>NO</td>
<td>Number of up-link packets sent</td>
</tr>
<tr>
<td>14</td>
<td>LORAMAC_OPT_SOURCE_VOLTAGE</td>
<td>YES</td>
<td>NO</td>
<td>The supply voltage of the device in millivolts</td>
</tr>
<tr>
<td>15</td>
<td>LORAMAC_OPT_915_HYBRID_MODE</td>
<td>-</td>
<td>-</td>
<td>Deprecated in version 17/18.4.1.0 in RM1xx and never supported in RM1xx_PE.</td>
</tr>
<tr>
<td>16</td>
<td>LORAMAC_OPT_BIRTHDAY</td>
<td>YES</td>
<td>NO</td>
<td>The date the device is created</td>
</tr>
<tr>
<td>17</td>
<td>LORAMAC_OPT_ADR_ENABLE</td>
<td>YES</td>
<td>YES</td>
<td>Enable (1) and disable (0) automatic data rate adaptation.</td>
</tr>
<tr>
<td>18</td>
<td>LORAMAC_OPT_CHANNELLIST</td>
<td>YES</td>
<td>NO</td>
<td>Lists the channel number, frequency and maximum data rate of all the enabled channels. Only valid in EU mode.</td>
</tr>
<tr>
<td>19</td>
<td>LORAMAC_OPT_CHANNELMASK</td>
<td>YES</td>
<td>YES / NO</td>
<td>The channels mask designating the enabled channels. Set option can only be used on the 915(US) MHz radio. Get option is valid for both sets of modules. The set option overrides the current value, but is not persisted. After a reboot, it reverts to the default or configured value. See the Setting RM191 ChannelsMask section below.</td>
</tr>
<tr>
<td>20</td>
<td>LORAMAC_OPT_NEXT_TX</td>
<td>YES</td>
<td>NO</td>
<td>Returns the time, in seconds, until the packet just loaded is transmitted to the gateway.</td>
</tr>
<tr>
<td>21</td>
<td>LORAMAC_OPT_TEMPERATURE</td>
<td>YES</td>
<td>NO</td>
<td>Temperature in degrees Celsius.</td>
</tr>
<tr>
<td>22</td>
<td>LORAMAC_OPT_TEMP_COMP_FACTOR</td>
<td>YES</td>
<td>NO</td>
<td>Temperature compensation factor – device specific.</td>
</tr>
<tr>
<td>23</td>
<td>LORAMAC_OPT_FREQ_ERROR</td>
<td>YES</td>
<td>NO</td>
<td>Frequency error (Hz) of the SX1272.</td>
</tr>
<tr>
<td>24</td>
<td>LORAMAC_OPT_FREQ_OFFSET</td>
<td>YES</td>
<td>NO</td>
<td>SX1272 frequency offset of the SX1272.</td>
</tr>
<tr>
<td>25</td>
<td>LORAMAC_OPT_MAX_RETRIES</td>
<td>YES</td>
<td>YES</td>
<td>Sets the number of times the module attempts to resend a confirmed packet if an acknowledgment is not received. The maximum and default values are 8.</td>
</tr>
<tr>
<td>26</td>
<td>LORAMAC_OPT_DEVICE_CLASS</td>
<td>YES</td>
<td>YES</td>
<td>Sets the Device class. 0 – Class A 2 – Class C</td>
</tr>
<tr>
<td>27</td>
<td>LORAMAC_OPT_MAX_JOIN_ATTEMPTS</td>
<td>YES</td>
<td>YES</td>
<td>Defines the number of Join Request attempts that are made using the same packet before an EVLORAMACRXTIMEOUT event is thrown. This command is only valid for the RM191 and RM191_PE series of modules.</td>
</tr>
<tr>
<td>Option</td>
<td>Option Name</td>
<td>Get</td>
<td>Set</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>-----</td>
<td>-----</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| 28     | LORAMAC_OPT_SUBBAND | YES | YES | This command must be sent before the first JoinRequest is transmitted. Sets the channels map by means of a sub-band value. There are 8 available sub-bands, 1-8. See the Setting RM191 ChannelsMask section below for more details of this command.  
**Note:** This command is only valid for the RM191 and RM191_PE series of modules. |
| 29     | LORAMAC_OPT_NEXTTX_TIME | NO  | YES | Returns the time to the NextTxEvent. This is also returned as part of the EVLORAMACSEQUENCECOMPLETE event. This will always be 0 for the US and AU modules. |
| 30     | LORAMAC_OPT_SEQUENCE_INCREMENT | YES | YES | The maximum incremental step the sequence counter could jump after a module reset. This value is not persisted and temporarily replaces the value stored in Flash using the at+cfg 1003 command.  
This command is only valid when using Personalisation.  
See the Interfacing with Lorawan documents for more details of this command. |
| 31     | LORAMAC_OPT_EEPROM_UPCOUNTER | YES | YES | The value that is stored in the serial EEPROM that will be used to initialise the uplink sequence number after a module reboot. This command can also be used to set the counter value and is persisted over a reset.  
The get option returns 2 values, the first is the local value, the second is the value stored in the EEPROM.  
This command is only valid when using Personalisation.  
See the Interfacing with LoRaWAN documents for more details of this command. |
| 32     | LORAMAC_OPT_EEPROM_DOWNCOUNTER | YES | YES | The value that is stored in the serial EEPROM that will be used to initialise the downlink sequence number after a module reboot. This command can also be used to set the counter value and is persisted over a reset.  
The get option returns 2 values, the first is the local value, the second is the value stored in the EEPROM.  
This command is only valid when using Personalisation. |
<table>
<thead>
<tr>
<th>Option</th>
<th>Option Name</th>
<th>Get</th>
<th>Set</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>See the Interfacing with Lorawan documents for more details.</td>
</tr>
</tbody>
</table>

Note: There is a subtle difference between options LORAMAC_OPT_NEXT_TX (20) and LORAMAC_OPT_NEWTX_TIME (29). The former is only returns a valid value once a packet has been set to the LoRa stack. The latter returns the time until a radio band duty cycle will be available. For US and AU modules both these values will always be 0.

### 5.1.10 LORAMACSetDebug

This command should only be used during development. It is not recommended that this command is left active in any production version of a smartBASIC application.

**FUNCTION**

This function sets the module up to output certain debug information, either as a text string or as a waveform.

**LORAMACSetDebug (nDebug, nTxSio, nRxSio)**

**Returns**

0x0000 – Successfully actioned command  
0xnnnn – Result code

**Arguments:**

| nDebug       | ByVal nDebug INTEGER  
|--------------|-----------------------|
|              | 0 – Disables debug mode  
|              | 1 – Enables debug mode |

| nTxSio       | ByVal nTxSio INTEGER  
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The SIO pin that the Tx waveform is output on.</td>
</tr>
</tbody>
</table>

| nRxSio       | ByVal nRxSio INTEGER  
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The SIO pin that the Rx waveform is output on.</td>
</tr>
</tbody>
</table>

**Interactive Command**

No

If debug mode is enabled, the module outputs the frequency on which a packet is transmitted, the data rate of that packet when transmitting, and the spreading factor/symbols timeout when receiving.

It also outputs two waveforms, on the selected SIOs, that mark when the module is in transmit or receive mode.

TxSio and RxSio cannot be the same value unless debug mode is being disabled.

Debug mode is not persisted. If debug mode was enabled and the module was then reset, it boots up with debug mode disabled.

More information on this functionality is available in the RM1xx Debug Features document, available under the documentation tab of the RM1xx product page: [http://www.lairdtech.com/products/rm1xx-lora-modules](http://www.lairdtech.com/products/rm1xx-lora-modules)

LORAMACSetOption is an extension function.

### 5.1.11 LoramacQueryTxPossible

This command allows the user to query the stack as to what is the maximum packet size for the current configuration.

**FUNCTION**

This function queries the module for the current maximum size of an uplink packet.
LORAMACQueryTxPossible (nSize, nCurrentSize, nMaxSize)

Returns
0x0000 – Successfully actioned command
0xnnnn – Result code
0x730E – Data packet is too large for the present configuration.

Arguments:

<table>
<thead>
<tr>
<th>nSize</th>
<th>ByVal nSize INTEGER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The size of the packet you wish to send</td>
</tr>
<tr>
<td>nCurrentSize</td>
<td>ByRef nCurrentSize INTEGER</td>
</tr>
<tr>
<td></td>
<td>The theoretical maximum size of an uplink packet, which is derived from the current data rate.</td>
</tr>
<tr>
<td>nMaxSize</td>
<td>ByRef nMaxSize INTEGER</td>
</tr>
<tr>
<td></td>
<td>The actual maximum size of an uplink packet once all the internal options are taken into account.</td>
</tr>
</tbody>
</table>

```basic
dim rc
dim currentsize,maxsize
dim data$

data$ = "hello"
rc = LoramacQueryTxPossible(strlen(data$),currentsize,maxsize)
if rc == 0 then
    // Everything ok
    print "current ";currentsize
    print " max ";maxsize
else
    // Error has occurred
    print "current ";currentsize
endif
```

LORAMACSetOption is an extension function.

5.2 Setting RM191 ChannelsMask

Note that this section is only relevant to the US and AU series of modules. The setting of the ChannelsMask in the RM186 is handled completely differently and requires no input by the user.

The ChannelsMask is a hex string where each individual bit refers to a specific RF channel.

The US and AU modules modules have 64 125kHz upstream channels (channels 0-63) with a 200kHz spacing between consecutive channels and 8 500KHz upstream channels (channels 64-71) with 1.6MHz spacing between consecutive frequencies. The actual frequencies that these channels represent differ between LoRaWan regions. The values for the US and AU regions are shown in Table 2 below.

To make configuration easier most gateway/network providers also use the concept of a sub-band. Each sub-band contains 8 consecutive 125 kHz channels and 1 500kHz channel. So sub-band 1 contains channels 0-7 and 64, sub-band 2 contains channels 8-15 and 65 etc.

By default, on power up all channels are enabled by setting their bit in the ChannelsMask to 1. When the module is tasked with transmitting a packet to the gateway, it randomly selects one of the enabled channels and the packet is transmitted on that frequency. At the moment, most gateways only support 8 125 kHz channels and 1 500KHz channel so there is a one in eight chance of selecting a channel that is supported by a gateway.
There are MAC commands available to the network server to modify the channelsmask on a module but these are only available after a module has joined the network. However, it can take several attempts before a join request is transmitted on an enabled frequency.

Also, once joined, there is no guarantee that a network server supports altering the channelsmask, so it is possible that you may need several attempts to successfully transmit any packet to the gateway and onto the network server.

So for both the JoinRequest and transmitting data packets the module may need to send the same packet several times before it is eventually transmitted on a channel supported by the gateway.

In the RM191 and RM191_ PE series of modules, however, it is possible to manually set the channels mask. Before using any of the commands below, it is important that you contact your network provider and ask which channels their gateways support. If you set up a module with the wrong channels mask, then it is not able to communicate with the network.

Note that the method the module uses to set the chanmas mask is governed by the \texttt{at+cfg 1002} command, which determines if the module reverts to the default value or uses one of the options below. This value is persisted.

The simplest method is to use either the \texttt{at+cfg 1001} or \texttt{LORAMACSetOption(LORAMAC_OPT_SUBBAND, x)} commands. This allows the user to set a specific sub-band. As most gateways can only be configured to the sub-band level, these commands should suffice in the vast majority of cases.

The difference in the commands is that the value set using the \texttt{at+cfg} command is persisted in memory, so survives a power cycle. The \texttt{LORAMACSetOption} is not persisted and so the value reverts to the value determined by the \texttt{at+cfg 1002} command.

Certain gateways, however, allow you to be more flexible when setting the channels map. The Laird RG1xx Sentrius, for example, allows you to set the channels map over multiple sub-bands. To support this, these modules allow you to enable individual channels by setting the bit that corresponds to their channel number in a hex string that is either loaded during bootup from a default value or from a value stored in Flash using the \texttt{at+cfgex 1009} command.

The value that must be set is governed by which frequency channels are supported by the gateway or gateways with which you want to communicate; for that, you must consult your network administrator.

It is very unlikely you would want to use the \texttt{at+cfgex 1009} method if a gateway is configured for a sub-band. In this case you are better using the sub-band methods described above. However, Table 2 shows the required strings as examples.

### Table 2: ChannelMask commands

<table>
<thead>
<tr>
<th>Sub-Band</th>
<th>Frequency Range (MHz)</th>
<th>Channels</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>US 902.3–903.7</td>
<td>0-7</td>
<td>at+cfgex 1009 &quot;000100000000000000ff&quot;</td>
</tr>
<tr>
<td></td>
<td>AU 915.2-916.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>US 903.9–905.3</td>
<td>8-15</td>
<td>at+cfgex 1009 &quot;0002000000000000ff00&quot;</td>
</tr>
<tr>
<td></td>
<td>AU 916.8-918.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>US 905.5–906.9</td>
<td>16-23</td>
<td>at+cfgex 1009 &quot;000400000000000000&quot;</td>
</tr>
<tr>
<td></td>
<td>AU 918.4-919.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>US 907.1–908.5</td>
<td>24-31</td>
<td>at+cfgex 1009 &quot;000800000000ff000000&quot;</td>
</tr>
<tr>
<td></td>
<td>AU 920.0-921.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>US 908.7–910.1</td>
<td>32-39</td>
<td>at+cfgex 1009 &quot;0010000000ff00000000&quot;</td>
</tr>
<tr>
<td></td>
<td>AU 921.6-923.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>US 910.3–911.7</td>
<td>40-47</td>
<td>at+cfgex 1009 &quot;002000000ff0000000000&quot;</td>
</tr>
<tr>
<td></td>
<td>AU 923.2-924.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>US 911.9–913.3</td>
<td>48-55</td>
<td>at+cfgex 1009 &quot;004000ff000000000000&quot;</td>
</tr>
<tr>
<td></td>
<td>AU 924.8-926.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>US 913.5–914.9</td>
<td>56-63</td>
<td>at+cfgex 1009 &quot;0080ff00000000000000&quot;</td>
</tr>
<tr>
<td></td>
<td>AU 926.4-927.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>All bands</td>
<td>902.3–914.9</td>
<td>0-63</td>
<td>at+cfgex 1009 &quot;00xffffffffffffffff&quot;</td>
</tr>
</tbody>
</table>

In the table above, channel 0 is the least significant bit on the extreme right-hand side of the string.
The *All Bands* option could be used if you don’t know the configuration of a gateway. With this ChannelsMask, the RM191 continues to transmit a packet on random channels until it finds one that worked. However, it is much easier to set `at+cfg 1002` to 0.

Most gateways at present only support the one sub-band option as this is determined by the configuration of the SX1301 module. However, if a gateway contains more than one of these modules, then it is simply a case of enabling the extra bits. For example, to enable sub-bands 1 and 4, the following string is entered:

```
at+cfgex 1009 “0009000000ff0000ff”
```

While it is possible to set the hex string to any channel configuration, the present method of configuring the gateways means that the ChannelsMask is most likely grouped in bands of three, four, or five channels. For example, if a gateway supported channels 4-7, 28-31, and 67, then the following string is required:

```
at+cfgex 1009 “0008000000f0000f0”
```

Note that the 500kHz channel must fit in one of the 125kHz channel sub-bands.

However, again, it is important to stress that you must contact your network administrator to determine which settings you require.

As has previously been mentioned, the `at+cfgex 1009 “xxx…”` command is stored in flash and you must configure the module before running a smartBASIC app. However, you may also modify the channelsmask from a smartBASIC application using the LoramacSetOption(`LORAMAC_OPT_CHANNELMASK,xxx…`).

If you wanted to set the RM191 to sub-band 2, then you can enter:

```
LoramacSetOption(LORAMAC_OPT_CHANNELMASK, 0002000000000000ff00)
```

Note with this method you do not require the “” around the hex string. ChannelsMasks entered this way are not persisted. When the module reboots it reverts to the stored or default value.

### 5.3 Events and Messages

#### 5.3.1 EVLORAMACJOINING

The device started the Join procedure. In the case of OTAA a Join Request message is sent and the device awaits the Join Response message from the server. For personalization the module just transitions to the Joined state.

#### 5.3.2 EVLORAMACJOINED

The device successfully completed the Join procedure. If using the OTAA Join procedure, this indicates that a successful Join Response message was received from the server. This event is thrown immediately when joining the network using Activation by Personalization (ABP).

This event marks the end of the Join sequence.

#### 5.3.3 EVLORAMACJOINFAIL

An event indicating that a JoinAccept message contains a mic error. This will result in the module retransmitting the uplink packet.

No action should be taken on receipt of this event. It is for information only. This event DOES NOT mark the end of the uplink/downlink sequence.

#### 5.3.4 EVLORAMACTXCOMPLETE

The module successfully sent an uplink packet. This event is sent at the end of the uplink/downlink sequence and not at the same time as the EVLORAMACTXDONE event as you might expect. So in the case of confirmed packets or downlink data...
packets where an EVLORAMACRXCOMPLETE event would be raised, the 2 events are sent at the same time. In the case of confirmed packets, if the downlink packet is not received this event will not be sent.

This event marks the end of the uplink/downlink sequence.

### 5.3.5 EVLORAMACRXCOMPLETE

The device has received a downlink packet. This downlink may contain data or just an acknowledgment to a confirmed uplink. If the packet contains data there will also be an EVLORAMACRXDATA event.

This event marks the end of the uplink/downlink sequence.

### 5.3.6 EVLORAMACTXTIMEOUT

An ACK was not received for a confirmed uplink message

This event marks the end of the uplink/downlink sequence.

### 5.3.7 EVLORAMACRXTIMEOUT

No acknowledgement was received for a confirmed uplink packet. This event will only be sent at the end of the uplink/downlink cycle (see Transmit/Receive Sequence) so several attempts will have been made to transmit the packet.

This event marks the end of the uplink/downlink sequence.

### 5.3.8 EVLORAMACRXERROR

An error occurred in the receive path.

This event marks the end of the uplink/downlink sequence.

### 5.3.9 EVLORAMACRXDATA

Downstream data received from the gateway. The application can read the data using LORAMACRxData().

### 5.3.10 EVLORAMAACLINKCHECKRESPMSG

This message is returned from the LoRa stack when a Link Check response is received from a LoRa gateway. The demodulation margin is reported as an 8-bit unsigned integer named Margin which ranges from 0 to 254. Margin indicates the link margin in dB of the last successfully reported LinkCheckReq command. The gateway count is returned as a variable named GwCnt and represents the number of gateways that successfully received the last LinkCheckReq command. See section 5.1 of the LoRaWAN specification for more details.

### 5.3.11 EVLORAMACTXDONE

This event indicates that a packet has successfully been transmitted from the radio. It is this event that all subsequent receive windows are timed with respect to.

No action should be taken on receipt of this event. It is for information only.

### 5.3.12 EVLORAMACNOSYNC

This event indicates that the receiver has failed to detect a sync message from the gateway during a receive window and so has exited the receive state. As there can be two potential receive windows for every transmitted packet, in worse case scenarios there can be two of these events for every uplink packet.

No action should be taken on receipt of this event. It is for information only.
5.3.13 EVLORAMACADR

This event indicates that the RM1xx has received an ADR command from the gateway/server and that a parameter setting on the module may have changed. The parameters that this command can change are the TxPower, Datarate or ChannelsMask.

This event also returns 2 parameters indicating the downlink packet type and whether there is any other downlink frames pending.

The downlink packet types are as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Packet Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Unconfirmed downlink</td>
</tr>
<tr>
<td>5</td>
<td>Confirmed downlink</td>
</tr>
</tbody>
</table>

If another downlink frame is pending the value returned will be a 1.

```c
FUNCTION HandlerLoramacAdrMessage(packettype,pending)
    sprint #strT$,"Adr packet: type ";packettype
    PrintMsg(strT$)
    sprint #strT$,"Pending ";pending
    PrintMsg(strT$)
endfunc
```

5.3.14 EVLORAMACNEXTTX

This event indicates that there as available duty cycle on one of the sub-bands. If a LORAMACTxData command is sent on receipt of this event, the data is sent without any delays. This event is only really valid in the modules that have duty cycle restrictions, such as the EU and AS regions.

For those modules that don’t have this duty cycle restriction (US and AU), a packet can be sent to the stack on receipt of an end of uplink/downlink sequence event and will be transmitted immediately. However there is harm in still using this event, it is just not strictly necessary.

5.3.15 EVLORAMACDOWNLINKREPEATED

Event indicating that the module has received a downlink packet with the same sequence number as the previous downlink packet.

5.3.16 EVLORAMACMICFAILED

An event indicating that the module has received a downlink packet containing a MIC error. This will result in the module retransmitting the uplink packet.

No action should be taken on receipt of this event. It is for information only. This event DOES NOT mark the end of the uplink/downlink sequence.

5.3.17 EVLORAMACSEQUENCECOMPLETE

This event is an amalgamation of some of the above events and marks the end of uplink/downlink sequence and that it now possible to load a new uplink packet into the stack. In EU modules that packet may not be transmitted immediately due to duty cycle constraints.

This event also returns 2 parameters the flag indicating which event triggered the end of the sequence and the time to the next EVLORAMACNEXTTX event.
The possible flag values are as follows:

**Table 3: Module version numbers**

<table>
<thead>
<tr>
<th>Value</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>EVLORAMACTXCOMPLETE</td>
</tr>
<tr>
<td>2</td>
<td>EVLORAMACRXCOMPLETE</td>
</tr>
<tr>
<td>3</td>
<td>EVLORAMACRXTIMEOUT</td>
</tr>
<tr>
<td>4</td>
<td>EVLORAMACRCRCERROR</td>
</tr>
<tr>
<td>5</td>
<td>EVLORAMACTXTIMEOUT</td>
</tr>
<tr>
<td>6</td>
<td>EVLORAMACRXERROR</td>
</tr>
<tr>
<td>7</td>
<td>EVLORAMACTXDRPAYLOADSIZEERROR</td>
</tr>
</tbody>
</table>

The function `HandlerSequenceComplete` is defined as follows:

```vbnet
FUNCTION HandlerSequenceComplete(flag, nexttime) As Integer
    sprint #strT$, "Sequence complete ";flag
    PrintMsg(strT$)
    sprint #strT$, "Next time ",nexttime
    PrintMsg(strT$)
endfunc
```

**ONEVENT EVLORAMACSEQUENCECOMPLETE** CALL HandlerSequenceComplete

### 5.3.18 EVLORAMACFRAMESLOSS

This event indicates that the difference between the actual and expected receive downlink sequence number is larger than the maximum allowed value. This event should not be acted upon directly, the stack should correct this automatically.

### 5.3.19 EVLORAMACTXDRPAYLOADSIZEERROR

This event will only occur when a module has entered the resend sequence, due to the non-receipt of an acknowledgement. As part of the resend procedure the datarate could reduce which could mean that the packet size that was originally within limits is now too large. If this occurs this event will be thrown.

This event marks the end of the uplink/downlink sequence.

### 6 ACKNOWLEDGEMENTS

The following are required acknowledgements to address our use of open source code on the RM1xx to implement AES encryption.

Laird’s implementation includes the following files: `aes.c` and `aes.h`.

Copyright (c) 1998-2008, Brian Gladman, Worcester, UK. All rights reserved.

#### 6.1.11 License Terms

The redistribution and use of this software (with or without changes) is allowed without the payment of fees or royalties providing the following:

- Source code distributions include the above copyright notice, this list of conditions and the following disclaimer;
- Binary distributions include the above copyright notice, this list of conditions and the following disclaimer in their documentation;
- The name of the copyright holder is not used to endorse products built using this software without specific written permission.
6.1.1.2 Disclaimer

This software is provided 'as is' with no explicit or implied warranties in respect of its properties, including, but not limited to, correctness and/or fitness for purpose.

Issue 09/09/2006

This is an AES implementation that uses only 8-bit byte operations on the cipher state (there are options to use 32-bit types if available).

The combination of mix columns and byte substitution used here is based on that developed by Karl Malbrain. His contribution is acknowledged.
## 7 Index of smartBASIC Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LORAMAC Option List</td>
<td>16</td>
</tr>
<tr>
<td>LORAMACGetOption</td>
<td>14</td>
</tr>
<tr>
<td>LORAMACJoin</td>
<td>9</td>
</tr>
<tr>
<td>LORAMACJoin (nFlags)</td>
<td>10</td>
</tr>
<tr>
<td>LORAMACLinkCheck</td>
<td>11</td>
</tr>
<tr>
<td>LORAMACRxData</td>
<td>13</td>
</tr>
<tr>
<td>LORAMACSetDebug</td>
<td>19</td>
</tr>
<tr>
<td>LORAMACSetOption</td>
<td>15</td>
</tr>
<tr>
<td>LoRaMACSleepMode</td>
<td>9</td>
</tr>
<tr>
<td>LORAMACTxData</td>
<td>11</td>
</tr>
</tbody>
</table>