# UM0978
# User manual

## Using the Simple MAC nodetest application

# 1 Introduction

The Simple MAC nodetest application is a low-level test program designed for the functional testing of RF modules (either your own custom-manufactured devices or those provided in the STM32W108 Kits), including token viewing, range testing, received signal strength indicator (RSSI) measurements, and special transmission test modes as required for FCC and CE certification.

This document describes how to set up and get started running the Simple MAC nodetest application on an MB851 platform.

# Contents

# 2 Getting started

The Simple MAC software package provides the nodetest application in binary format: simplemac-test.s37. This image is contained in the prebuilt folder delivered with the Simple MAC software package.

## 2.1 Downloading the binary image

To download the binary image of the Simple MAC nodetest application for use on an MB851 board, follow these steps:

1. Connect the MB851 board to a PC USB port through a mini-USB cable.
2. Identify the USB virtual COM associated to the connected device (from My Computer, Manage, Device Manager and open Ports (COM & LPT)).
3. Open a command prompt on your PC and go to the: C:\Program Files\STMicroelectronics\ ST SimpleMAC 1.0.1\STM32W108\utility
4. Enter the following command:
   ```
   stm32w_flasher -p COMx -r -f "C:\Program
   Files\STMicroelectronics\ ST SimpleMAC 1.0.1\
   prebuilt\simplemac-test.s37"
   ```
   where COMx is the USB virtual com port previously detected.

*Note: If only one MB851 board is connected to your PC, you can just use the `-p auto` option for automatically detecting the related USB virtual COM.*

## 2.2 Accessing the nodetest application using HyperTerminal

### Configuring the HyperTerminal for use

To run the Simple MAC nodetest application, open a HyperTerminal session on the related USB COMx port with these settings:

- Bit rate: 115200
- Data bits: 8
- Parity: None
- Stop bits: 1
- Flow control: None

### Accessing the Simple MAC nodetest commands

Press 'enter' on the keyboard to access the Simple MAC nodetest commands.

Each time the Simple MAC nodetest starts execution, the following operations are automatically performed:

| | |
|---|---|
| `hwver`: | Displays the hardware version. |
| `Crashinfo`: | Displays any available crash information. |
| `Resetstring`: | Displays the reset reasons. |
| `initTokens`: | Initializes tokens. |
| `initRadio`: | Initializes the radio. |

`seedPnrg`: Seeds the PNRG with random data from the radio.

`setPerTestTx`: Sets up transmit parameters for the IEEE.802.15.4 PER test.

*Note: Enter `help` to display a list of all supported commands.*

# 3 Simple MAC nodetest command classes

The following classes of test commands are supported:

- led-test
- memory-test
- token-cortexm3-test & token-test
- reset-test
- phy-security
- phy-common
- phy-common-tx
- phy-common-rx
- phy-stm32w108xxclass
- phy-stm32w108xxclass-lib-timer

## Command format

The nodetest test commands have the following format. All input values are in hexadecimal (hex) format.

```
COMMAND [PARAMETERS] [- DESCRIPTION]
```

## Supported parameter types

*Table 1* list the supported parameter types. Commands are NOT case sensitive.

**Table 1. Simple MAC nodetest parameters types**

| Parameters | Type | Description |
|---|---|---|
| b | buffer | Buffer arguments are given as a string "...". Buffer arguments can also be given as a hex digit string using curly braces: {1A2B3C}. The hex digits are converted to bytes in pairs. |
| s1 | int8s | One-byte signed |
| s2 | int16s | Two-byte signed |
| u1 | int8u | One-byte unsigned |
| u2 | int16u | Two-byte unsigned |
| u4 | int32u | Four-byte unsigned |

## 3.1 LED test commands

LED test commands verify the correct operation of the STM32W108 MB851 board LEDs. *Table 2* describes the related supported test commands.

**Table 2. List of LED test commands**

| Test command | Test description |
|---|---|
| `ledTest` | Cycle the LEDs. |
| `ledOn u1` | Turns on LED u1. |
| `ledOff u1` | Turns off LED u1. |
| `ledToggle u1` | Toggles LED u1. |

### 3.1.1 Examples of LED test commands

To turn on the MB851 LED D1, enter:

```
> ledOn 1
```

To turn on the MB851 LED D3, enter:

```
> ledOn 2
```

## 3.2 Memory test commands

The memory tests commands carry out various memory tests. *Table 3* describes the related supported test commands.

**Table 3. Memory test commands**

| Test command | Test description |
|---|---|
| `getmemb u4u4` | Read y bytes from x address (byte aligned) |
| `getmemhw u4u4` | Read y halfwords from x address (halfword aligned) |
| `getmemw u4u4` | Read y words from x address (word aligned) |
| `getmem u1u4u4` | Generic get memory (bitsize, address, length) |
| `setmemb u4u1` | Store byte y in address x (byte aligned) |
| `setmemhw u4u2` | Store halfword y in address x (halfword aligned) |
| `setmemw u4u4` | Store word y in address x (word aligned) |
| `setmem u1u4u4` | Generic set memory (bitsize, address, value) |
| `hwver` | Display the hardware version |

### 3.2.1 Memory test command examples

To read 5 bytes from address 0x20000000, enter:

```
> getmemb 0x20000000 5
```

The following information is displayed:

```
#{{(getmemb)} Addresses 0x20000000 - 0x20000004 contain: {addr}
{data}}
{[{0x20000000}] = {0x03}}
{[{0x20000001}] = {0x0A}}
{[{0x20000002}] = {0x3C}}
{[{0x20000003}] = {0xC3}}
{[{0x20000004}] = {0x00}}
```

## 3.3 Token test commands

The token test commands are used to obtain information about the STM32W108 token system. *Table 4* describes the related supported test commands.

**Table 4. Token test commands**

| Test command | Test description |
|---|---|
| tokMap | Print the Token Memory Map |
| initTokens | Invoke top level token initialization |
| tokRead u2 | Read all data of a token (u2 = creator code) |
| tokDump | Dump the entire token data set |
| crashinfo | Print crash information |

### 3.3.1 Token test commands examples

To display the complete token data set, enter:

```
> tokmap
```

The following information is displayed:

```
Start Address - End Address -- [creator] name
0x0804077E - 0x08040795  -- [C344] MFG_CHIP_DATA
0x08040796 - 0x0804079B  -- [F064] MFG_PART_DATA
0x0804079C - 0x080407A1  -- [F464] MFG_TESTER_DATA
0x080407A2 - 0x080407A9  -- [E545] MFG_ST_EUI_64
0x080407AA - 0x080407BD  -- [F46E] MFG_ANALOG_TRIM_NORMAL
0x080407BE - 0x080407D1  -- [F442] MFG_ANALOG_TRIM_BOOST
0x080407D2 - 0x080407DB  -- [F462] MFG_ANALOG_TRIM_BOTH
0x080407DC - 0x080407DD  -- [F274] MFG_REG_TRIM
0x080407DE - 0x080407DF  -- [F276] MFG_1V8_REG_VOLTAGE
0x080407E0 - 0x080407E1  -- [F676] MFG_VREF_VOLTAGE
```

```
0x080407E2 - 0x080407E3  -- [F463] MFG_TEMP_CAL
0x080407F4 - 0x080407F5  -- [FF09] MFG_FIB_VERSION
0x080407F6 - 0x080407F7  -- [E663] MFG_FIB_CHECKSUM
0x080407F8 - 0x080407FF  -- [E66F] MFG_FIB_OBS
0x08040800 - 0x0804080F  -- [E36F] MFG_CIB_OBS
0x08040810 - 0x08040811  -- [C356] MFG_CUSTOM_VERSION
0x08040812 - 0x08040819  -- [E345] MFG_CUSTOM_EUI_64
0x0804081A - 0x08040829  -- [ED73] MFG_STRING
0x0804082A - 0x08040839  -- [C24E] MFG_BOARD_NAME
0x0804083A - 0x0804083B  -- [C944] MFG_MANUF_ID
0x0804083C - 0x0804083D  -- [D043] MFG_PHY_CONFIG
0x0804083E - 0x0804084D  -- [C24B] MFG_BOOTLOAD_AES_KEY
0x0804084E - 0x08040855  -- [CD53] MFG_EZSP_STORAGE
0x08040856 - 0x0804087D  -- [C143] MFG_ASH_CONFIG
0x0804087E - 0x080408D9  -- [C342] MFG_CBKE_DATA
0x080408DA - 0x080408ED  -- [C943] MFG_INSTALLATION_CODE
0x080408EE - 0x080408EF  -- [B254] MFG_OSC24M_BIAS_TRIM
(Virtual Token)          -- [B634] MFG_EUI_64
```

To dump the entire token data set, enter:

```
> tokdump
```

The following information is displayed (little endian, all data represented as a byte array):

```
[C344] MFG_CHIP_DATA AA 55 F9 38 39 34 37 39 36 43 FF FF FF FF 19
00 FF FF FF FF AA 55 F9 38
[F064] MFG_PART_DATA 01 00 02 00 01 FF
[F464] MFG_TESTER_DATA F9 38 FF FF FF FF
[E545] MFG_ST_EUI_64 37 05 00 00 02 E1 80 00
[F46E] MFG_ANALOG_TRIM_NORMAL EF 02 14 02 EF 02 EF 12 27 00 34 00
34 00 EF 02 00 00 34 00
[F442] MFG_ANALOG_TRIM_BOOST EF 02 54 30 EF 02 EF 12 27 00 34 00
34 00 EF 02 00 1B 34 00
[F462] MFG_ANALOG_TRIM_BOTH 73 06 F1 02 34 00 F4 01 F1 02
[F274] MFG_REG_TRIM 01 05
[F276] MFG_1V8_REG_VOLTAGE D7 46
[F676] MFG_VREF_VOLTAGE F4 2E
[F463] MFG_TEMP_CAL 84 45
[FF09] MFG_FIB_VERSION FE 01
[E663] MFG_FIB_CHECKSUM FF FF
[E66F] MFG_FIB_OBS FF FF 02 FD 55 AA FF FF
[E36F] MFG_CIB_OBS A5 5A FF FF FF FF FF FF FF 00 FF 00 FF 00 FF
00
[C356] MFG_CUSTOM_VERSION 01 FE
[E345] MFG_CUSTOM_EUI_64 FF FF FF FF FF FF FF FF
```

```
[ED73] MFG_STRING FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
[C24E] MFG_BOARD_NAME 4D 42 38 35 31 20 41 FF FF FF FF FF FF FF
FF FF
[C944] MFG_MANUF_ID FF FF
[D043] MFG_PHY_CONFIG FF FF
[C24B] MFG_BOOTLOAD_AES_KEY FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF
[CD53] MFG_EZSP_STORAGE FF FF FF FF FF FF FF FF
[C143] MFG_ASH_CONFIG FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF
[C342] MFG_CBKE_DATA FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF
[C943] MFG_INSTALLATION_CODE FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF
[B254] MFG_OSC24M_BIAS_TRIM FF FF
[B634] MFG_EUI_64 37 05 00 00 02 E1 80 00
```

## 3.4 Reset test commands

The reset tests are used to test different reset scenarios and display the reset reason. *Table 5* describes the related supported test commands.

**Table 5. Reset test commands**

| Test command | Test description |
|---|---|
| `resetstring` | Displays the ResetString. |
| `assertme` | Resets the chip using assert(). |
| `reboot` | Resets the chip using halReboot(). |
| `forceReset u2` | Resets with a specific forced reset type. |
| `wdogreset` | Triggers a reset using the watchdog. |

### 3.4.1 Reset test commands examples

To display the reason for the reset string, enter:

```
> resetstring
```

The following information is displayed (depending on the reset reason):

`RESET:EXT-PIN` (when resetting through the MB851 RST1 button)

*Table 6* describes the possible reasons for reset.

**Table 6. List of reset reasons**

| Reason | Description |
|---|---|
| PWR-HV | Power-on reset type - High voltage poweron |
| EXT-PIN | External reset trigger - External pin reset |
| SW-RBT | Software triggered reset - General software reboot |
| FIB-GO | Reset originated from the FIB bootloader - FIB bootloader caused a reset in main Flash memory |
| CRS-AST | Software crash - a self-check assert in the code failed |
| WDG-LWM | Watchdog reset occurred - Watchdog low watermark expired and caught extended info |
| FLASH | Flash failure cause reset |
| FATAL | A non-recoverable fatal error occurred |
| FAULT | An access fault occurred |
| UNK | Unknown cause |

## 3.5 PHY security test commands

The PHY security tests are used to test the STM32W108 AES encryption features. *Table 7* describes the related supported test commands.

**Table 7. PHY security test commands**

| Test command | Test description |
|---|---|
| `getAesKey` | Display the AES key |
| `setAesKey u1u1u1u1u1` | (o d0 d1 d2 d3) Set AES key data d0-3 at offset o |
| `getPlainText` | Display the plaintext |
| `setPlainText u1u1u1u1u1` | (o d0 d1 d2 d3) Set plaintext d0-3 at offset o |
| `aesEncrypt` | Encrypt the plaintext using the AES key and display the ciphertext |

### 3.5.1 PHY security test commands examples

To set AES Key data at offset 0, 4, 8, c, enter the following commands:

```
> setaeskey 0 11 11 11 11
> setaeskey 4 22 22 22 22
> setaeskey 8 33 33 33 33
> setaeskey c 44 44 44 44
```

To display the set AES key, enter:

```
> getaeskey
```

The following data is displayed:

```
0x00: 0x11 0x11 0x11 0x11
0x04: 0x22 0x22 0x22 0x22
0x08: 0x33 0x33 0x33 0x33
0x0C: 0x44 0x44 0x44 0x44
```

To set the data to be encrypted, enter:

> **setPlainText 0 55 55 55 55**

> **setPlainText 4 66 66 66 66**

> **setPlainText 8 77 77 77 77**

> **setPlainText c 88 88 88 88**

To display the data to be encrypted, enter:

```
> getplaintext
```

The following data is displayed:

```
0x00: 0x55 0x55 0x55 0x55
0x04: 0x66 0x66 0x66 0x66
0x08: 0x77 0x77 0x77 0x77
0x0C: 0x88 0x88 0x88 0x88
```

To encrypt the set data with the set key, enter:

```
> aesEncrypt
```

The encrypted data is displayed:

```
0x00: 0x55 0x1E 0x94 0x3F
0x04: 0x25 0x28 0x4C 0xFF
0x08: 0x0D 0xFA 0x3A 0xD5
0x0C: 0x83 0xA2 0xFB 0x0D
```

## 3.6 PHY common test commands

The PHY common tests provide a set of commands used to setup the radio and be able to send and receive IEEE 802.15.4 compliant packets. In particular, the user can initialize the radio, seed the random generator with a seed number coming from the radio, set the radio operating channel, sleep/wake up the radio, display the channel state (busy/clear), display the current RSSI and energy value, set the radio receiver filters for PANID and short long address, set the energy threshold used for the Clear Channel Assessment (CCA). Furthermore, the user can enable the Packet Trace Interface inside the STM32W108. A command for enabling a Packet Error Rate test is also provided.

**Table 8. PHY common test commands**

| Test command | Test description |
|---|---|
| `initRadio` | Initialize the radio |
| `seedPnrg` | Seed the random number generator with random data from the radio |
| `setPerTest u1` | (m) Set Packet Error Rate test mode to m (0 = Disable; 1 = Enable) |
| `getChannel` | Display the current channel |

**Table 8. PHY common test commands (continued)**

| Test command | Test description |
|---|---|
| `setChannel u1` | (c) Set the radio channel to c |
| `gRadPowState` | Display the radio power state |
| `sRadPowState u1` | (s) Set the radio power state to s (0 = Asleep; 1 = Awake) |
| `getCca` | Display the Clear Channel Assessment. 1 = Clear; 0 = Busy. |
| `getRssi` | Display the RSSI in dBm over 8 symbol periods. |
| `getEd` | Get 802.15.4 ED (-100 dBm to -36 dBm: 0x00 to 0xFF) over 8 symbol periods. |
| `getPanId` | Display the local host PANID. |
| `setPanId u2` | Set local host PANID. |
| `getShortAddr` | Display the local host short address. |
| `setShortAddr u2` | Set local host short address. |
| `getLongAddr` | Display the local host extended address. |
| `setAutoCal u1` | (m) Set auto calibration mode in RX and TX to m (0 = Disable; 1 = Enable) |
| `getCcaThresh` | Display the energy detection clear channel assessment threshold in dBm. |
| `setCcaThresh s1` | (e) Set energy detection clear channel assessment threshold to e in dBm. |
| `getPtiOutput` | Indicates whether Packet Trace Interface output is enabled or not. |
| `setPtiOutput u1` | (p) Enable (p = 1, default) or disable (p = 0) Packet Trace output. |

### 3.6.1 Phy common test commands examples

To display the current channel (range: [11,26], default: 11), enter:

```
> getchannel
```

The following information is displayed:

```
{{(getChannel)} Radio channel {channel:0x0B}}
```

To set the radio channel to 15, enter:

```
> setchannel f
```

The following information is displayed:

```
{{(setChannel)} Setting channel and calibrating (as
needed)...{status:0x00}}
{{(getChannel)} Radio channel {channel:0x0F}}
```

To display the current channel status, enter:

```
> getcca
```

The following information is displayed (1: channel is clear; 0: channel is busy):

```
{{(getcca)} {CCA:1}}
```

To set the radio PANID to 0x3344, enter:

```
> setpanid 0x3344
```

To display the current radio PANID, enter:

```
> getpanid
```

The following information is displayed:

```
{{(getpanid)} {PAN id:0x3344}}
```

To display the channel RSSI value, enter:

```
> getrssi
```

The following information is displayed:

```
{{(getrssi)} {RSSI:-99} [dBm]}
```

## 3.7 PHY common TX test commands

The PHY common TX tests provide a set of commands used to set up the radio for the transmission of IEEE 802.15.4 compliant packets. In particular, the user can specify the transmission output power, the IEEE 802.15.4 packet length and content, configure certain parameters affecting the packet transmission (wait or not for packet acknowledgment if packet request ack, perform or not the CCA before transmission, add or not CRC to the transmitted packet, set the number of backoffs and exponents involved in the unslotted CSMA-CCA algorithm). Furthermore, the user can choose to enable the SFD send event notification for transmitted packets.

Certain commands for sending a single carrier frequency (tone) or continuous stream of random symbols are also provided.

**Table 9. PHY common TX test commands**

| Test command | Test description |
|---|---|
| `setPerTestTx` | Setup TX for IEEE 802.15.4 PER test |
| `getTxPower` | Display the current radio TX power in dBm |
| `setTxPower s1` | (p) Set the radio TX power to p dBm |
| `getTxDelay` | Display the radio TX interpacket delay in us |
| `setTxDelay u2` | (d) Set the radio TX interpacket delay to d us |
| `tx u2` | (n) Transmit n packets (0 = infinite) |
| `txStream` | Transmit continuous stream of random symbols |
| `txTone` | Transmit single carrier frequency |
| `getTxPacket` | Display the TX packet length and payload |
| `setTxLength u1` | Set length of transmit payload |
| `setTxPayload u1u1u1u1u1` | (o d0 d1 d2 d3) Set TX packet payload data at offset o |
| `getTxConfig` | Output waitForAck, checkCca, ccaAttemptMax, backoffExponentMin, backoffExponentMax |
| `waitForAck u1` | (b) Enable (b = 1) or disable (b = 0) hardware wait for ack |
| `setCheckCca u1` | (b) Enable (b = 1) or disable (b = 0) hardware backoff CCA and check |
| `setCcaMax u1` | (n) Set the number of times to check CCA to n |

**Table 9. PHY common TX test commands (continued)**

| Test command | Test description |
|---|---|
| `setBoExpMin u1` | (e) Set the minimum backoff exponent to e |
| `setBoExpMax u1` | (e) Set the maximum backoff exponent to e |
| `appendCrc u1` | (a) Enable (a = 1) or disable (a = 0) hardware appended CRC. |
| `setSfdSent u1` | (a) Enable (a = 1) or disable (a = 0) SFD sent callback. |

### 3.7.1 PHY common TX test command examples

To display the current output power (default is 3), enter:

```
> gettxpower
```

The following information is displayed:

```
{{(getTxPower)} {actualPower:3}dBm}
```

To display the current radio transmit configuration setting, enter:

```
> gettxconfig
```

The following information is displayed (1: enabled; 0: disabled):

```
{{(gettxconfig)}
{waitForAck:1}
{checkCca:0}
{ccaAttemptMax:4}
{backoffExponentMin:3}
{backoffExponentMax:5}
{appendCrc:1}}
```

To display the current output power (default is 3), enter:

```
> gettxpower
```

The following information is displayed:

```
{{(getTxPower)} {actualPower:3}dBm}
```

To set the packet length of 11 (including 2 bytes for the CRC), enter:

```
> setTxLength 0xb
```

To set the IEEE 802.15.4 packet content 0x21080012230222F241, enter:

```
> setTxPayload 0 21 08 00 12
> setTxPayload 4 23 02 22 F2
> setTxPayload 8 41 00 00 00
```

To display the current packet, enter:

```
> gettxpacket
```

The following information is displayed:

```
len: 0x0B
0x00: 0x21 0x08 0x00 0x12
0x04: 0x23 0x02 0x22 0xF2
0x08: 0x41 0x00 0x00
```

To send 5 packets, enter:

```
> tx 5
```

When the packet transmission is completed, the following information is displayed:

```
Txing 5 packets 50000 us apart. 'e'nd...
5 packets transmitted.
Last packet status: 0x0000
Last packet backoff: 0x0000
Frame pending in last ACK: No.
SFD time (via TX complete callback): 0x7C56
```

*Note: To stop the packet transmission, enter* `e`.

## 3.8 PHY common RX test commands

The PHY common RX tests provide a set of commands used to set up the radio for the reception of IEEE 802.15.4 compliant packets. In particular, the user can select if displaying or not the received packet payload, enabling or disabling the hardware filters for the packet PANID and short or long addresses, enabling or disabling the automatic transmission of acknowledgment on reception of packet which requests ack, enable/disable the coordinator feature, enable/disable the check of the CRC value on the received packet, enable/disable the notification of overflow event on the receiver DMA. Furthermore, the user can display the current receiver status (if filters, automatic acknowledge, coordinator and overflow features are enabled).

**Table 10. PHY common RX test commands**

| Test command | Test description |
|---|---|
| `showPayload u1` | Enable (1) or disable (0, default) PHY payload display in RX output |
| `rx` | Receive mode |
| `setAddrFilt u1` | (f) Enable (f = 1) or disable (f = 0, default) hardware address filtering |
| `setAutoAck u1` | (a) Enable (a = 1) or disable (a = 0, default) auto-ack to ack request packet |
| `setCoord u1` | (c) Enable (c = 1) or disable (c = 0, default) this device as coordinator |
| `setCheckCrc u1` | (c) Enable (c = 1, default) or disable (c = 0) discarding packets with bad CRC |
| `setOverflow u1` | (c) Enable (c = 1, default) or disable (c = 0) notification of overflow events |
| `getRxConfig` | Display RX settings, e.g. filtering, auto ack, etc. |

### 3.8.1 PHY common RX test command examples

To display the current receiver status, enter:

```
> getrxconfig
```

The following information is displayed:

```
{{(getrxconfig)}
{addressFilter:0}
{autoAck      :0}
{coordinator  :0}
{overflow     :0}
}
```

*Note: Using the default settings (no hardware filters and automatic acknowledge enabled), the radio is able to receive all the packets sent in the set channel (radio is in Promiscuous mode).*

To display the received packet payload, enter:

```
> showpayload 1
```

To set the radio in Receive mode, enter:

```
> rx
```

The following information is displayed:

```
{{(rx)} test start ('e'nd)}
#{{(rx)}
{num}   {oflo}  {seq}   {per}  {err} {lqi}  {rssi}{ed}   {gain}
{status} {time}       {fp}{length}{payload}}
```

*Table 11* describes the meaning of the RX fields.

**Table 11. RX command fields description**

| Field | Description |
|---|---|
| num | Number of packets received |
| oflo | Number of detected RX overflows |
| seq | Packet sequence number |
| per | Calculated packet error rate. |
| err | The number of correlator errors in the packet. |
| lqi | Calculated Link Quality Indication |
| rssi | Received Signal Strength Indication |
| ed | ED converted from RSSI |
| gain | Receiver gain settings |
| status | Hardware status |
| time | MAC timer (resolution: 1µs) when the packet was received |
| fp | TRUE if the library set the Frame Pending bit in the hardware-generated MAC ACK to this packet, FALSE otherwise |

**Table 11. RX command fields description (continued)**

| Field | Description |
|---|---|
| `length` | MAC packet length |
| `payload` | MAC Packet (the 2-bytes CRC is not displayed) |

*Note: To stop the packet receive mode, type* `e`.

## 3.9 PHY stm32w108xxclass test commands

The stm32w108xxclass provide a set of commands used to set a channel and force the related calibration, set the power mode (normal or boost) and if an external power amplifier is used in user RF design, check if the temperature conditions require the radio to be re-calibrated.

*Note: Each use of the calChannel command cause unnecessary wear of the Flash memory due to writing the token value. This command should be only used to recover from hardware-related calibration problems, which should only occur during hardware development. If there are problems with the hardware design it may be possible for erroneous calibration values to be stored in the token. In this case, the token data must be erased.*

**Table 12. PHY stm32w108xxclass test commands**

| Test command | Test description |
|---|---|
| `calChannel u1` | Set the channel; perform full calibration |
| `getTxPowMode` | Display the TX power level and PA settings |
| `setTxPowMode u1u1` | (l p) Set Tx power level l (0 = normal, 1 = boost) and PA p (0 = int, 1 = ext) |
| `checkRadio u1` | (t) Monitor checkRadio() for calibration needed, optionally transmit throughout (t = 1) |

### 3.9.1 PHY stm32w108xxclass test command examples

To check if calibration is needed, enter:

```
> checkradio 1
```

The following information is displayed (temperature (°C) and radio calibration values):

```
radio is ASLEEP
tx during test: yes
auto-calibration: enabled
checkRadio mode. 'e'nd...
#{{temp} {vco}  {mod}  {lna}}
{{   23} {0x20} {0x41} {0x0C}}
```

*Note: To stop the Checkradio mode, type* `e`.

*Table 13* describes meaning of the checkRadio fields.

**Table 13. checkRadio calibration values**

| Field | Description |
|---|---|
| `temp` | Temperature (in degrees Celsius) |
| `vco` | Voltage Controlled Oscillator |
| `mod` | Modulation Digital to Analog Converter |
| `lna` | Low Noise Amplifier |

## 3.10 PHY stm32w108xxclass-lib-timer test commands

The phy-stm32w108xxclass-lib-timer provides a set of commands for using the radio MAC Timer. The MAC timer is 20 bits long with each LSB tick representing 1 µs. The MAC timer rolls over to zero approximately once every second. The MAC timer is free-running from the time that the radio is initialized.

**Table 14. PHY stm32w108xxclass test commands**

| Test command | Test description |
|---|---|
| `getMacTmr` | Display the current value of the MAC timer |
| `macTmrCmp u1` | (n) Count n MAC timer compare interrupts, n = 0 for infinite |
| `enaMacTmrCmp u1` | (e) Enable (e = 1) or disable (e = 0) MAC timer compare event |
| `setMacTmrCmp u2` | (c) Set MAC timer compare event to occur at count c |
| `getMacTmrCfg` | Display the MAC timer compare configuration |

### 3.10.1 Phy stm32w108xxclass test commands examples

To display the current MAC timer value, enter:

```
> getMacTmr
```

The following information is displayed:

```
{{(getMacTmr)} {MAC Timer:0x000BAED3}}
```

# 4 Simple MAC Library nodetest test scenarios

## 4.1 Transmit single carrier frequency (unmodulated signal) on a specific channel

To ensure FCC or CE compliance, certain tests require transmitting an unmodulated carrier wave (tone) over a specific channel. To send an unmodulated carrier wave (tone), follow these steps:

1. Setup a node (TX) as described in *Section 2: Getting started*.
2. On the TX node, set the radio channel (range: [11,26], default: 11) where `x` is the channel (in hexadecimal format):

   ```
   > setchannel x
   ```
3. On the TX node set the radio power (range: [-43,8] dBm, default: 3) where x is the power (in hex):

   ```
   > settxpower x
   ```
4. On the TX node, start a tone transmission:

   ```
   > txTone
   ```

*Note: To stop the txTone test, type `e` on the related Hyper Terminal.*

## 4.2 Transmit continuous stream of random symbols on a specific channel

To ensure FCC or CE compliance, certain tests require transmitting a continuous stream of random symbols over a specific channel. To send a continuous stream of random symbols, follow these steps:

1. Setup a node (TX) as described in *Section 2: Getting started*.
2. On the TX node, set the radio channel (range: [11,26], default: 11) where `x` is the channel (in hexadecimal format):

   ```
   > setchannel x
   ```
3. On the TX node set the radio power (range: [-43,8] dBm, default: 3) where x is the power (in hex):

   ```
   > settxpower x
   ```
4. On the TX node, start sending a continuous stream of random symbols:

   ```
   > txStream
   ```

*Note: To stop the txStream test, type `e` on the related Hyper Terminal.*

## 4.3 Packet error rate test

The packet error rate (PER) test consists of sending a set of packets from a transmitter node (TX) and receiving them on a receiver node (RX). The related error rate is calculated and displayed on the RX node. To perform the packet error test, follow these steps:

1. Setup 2 nodes (TX = transmitter node, RX = receiver node) as described in *Section 2: Getting started*.
2. On both nodes (TX and RX), type the following command for enabling the packet error test:

   **> setpertest 1**
3. On both nodes (TX and RX), set the radio channel (range: [11,26], default: 11) where `x` is the channel (in hex):

   **> setchannel x**
4. On TX node, set the radio power (range: [-43,8] dBm, default: 3) where x is the power (in hex):

   **> settxpower x**
5. On the RX node, enable Receive mode:

   **> rx**
6. On the TX node, transmit `n` packets (`n` = 0: infinite):

   **> tx n**
7. On the RX node, the HyperTerminal displays `n` packets with associated information as described in *Section 3.8.1: PHY common RX test command examples*.

*Table 15* shows the execution steps for performing a per test by sending 5 packets on the TX node.

**Table 15. Packet Error Rate test example**

| TX node per test commands | RX node per test commands |
|---|---|
| > **setpertest 1**<br>{{(setPerTest)} {PER Test Mode:enabled}} | > **setpertest 1**<br>{{(setPerTest)} {PER Test Mode:enabled}} |
| | > **rx**<br>{{(rx)} test start ('e'nd)}<br>#{{(rx)}<br>{num} {oflo} {seq} {**per**} {err} {lqi}<br>{rssi}{ed} {gain} {status} {time}<br>{fp}{length}} |
| > **tx 5**<br>Txing 5 packets 25500 us apart. 'e'nd...<br>5 packets transmitted.<br>Last packet status: 0x0000<br>Last packet backoff: 0x0000<br>Frame pending in last ACK: No.<br>SFD time (via TX complete callback): 0x49F9 | { { 1} { 0} { 1} { **0**} { 0} {0xFF} {-47} {0xD4} {0xB1} {0x4000} {0x00034095} {0} {0x12} }<br>{ { 2} { 0} { 2} { **0**} { 0} {0xFF} {-47} {0xD4} {0xB1} {0x4000} {0x0003A962} {0} {0x12} }<br>{ { 3} { 0} { 3} { **0**} { 0} {0xFF} {-47} {0xD4} {0xB1} {0x4000} {0x0004122E} {0} {0x12} }<br>{ { 4} { 0} { 4} { **0**} { 0} {0xFF} {-47} {0xD4} {0xB1} {0x4000} {0x00047AF9} {0} {0x12} }<br>{ { 5} { 0} { 5} { **0**} { 0} {0xFF} {-47} {0xD4} {0xB1} {0x4000} {0x0004E3C6} {0} {0x12} } |

The fourth column in the display output (labeled {per}) shows the packet error rate. For this value to be accurate, the receiver should not hear any other devices. Exiting the test and restarting clears the values and resets the values being displayed.

*Note: 1 Nodetest attempts to display the packet data as fast as it can, but it is possible to receive packets faster than nodetest can display the data. Therefore, there may be gaps in the displayed packets.*

*2 To stop the TX or RX tests, type `e` on the related HyperTerminal.*

## 4.4 TX and RX test scenarios

The Simple MAC nodetest commands can be used to target different types of simple transmission/reception scenarios:

1. Receiver is able to receive all the packets on the selected radio channel (Promiscuous mode).
2. Receiver enables hardware filters on PANID, short or long address for receiving packets with the specific destination PANID and short or long address.
3. Receiver enables the Coordinator mode and a filter on a specific PANID for receiving packets with no destination addresses but with source PANID matching the receiver PANID.

### 4.4.1 TX and RX test with RX in promiscuous mode

1. Setup 2 nodes (TX = transmitter node, RX = receiver node) as described in *Section 2: Getting started*.
2. On both nodes (TX and RX), set the same radio channel (range: [11,26], default: 11) where x is the channel (in hex):

   ```
   > setchannel x
   ```

3. On the TX node, set the radio power (range: [-43,8] dBm, default:3) where x is the power (in hex):

   ```
   > settxpower x
   ```

4. On the TX node, display the current packet (user can change the default length and content using the `setTxLength` and `setTxPayload` commands):

   ```
   > gettxpacket
   ```

5. On the RX node, ensure that the radio filters are disabled (the Simple MAC nodetest configures each node in promiscuous node as default). Enter the following command to display the radio filters state:

   ```
   > getrxconfig
   {{(getrxconfig)}
    {addressFilter:0}
    {autoAck      :0}
    {coordinator  :0}
    {overflow     :0}
    }
   ```

If the `addressFilter` and `autoAck` values are '1', disable them using the related commands:

```
> setaddrfilt 0
> setAutoAck 0
```

6. On the RX node, enable the payload display:

```
> showpayload 1
```

7. On the RX node, enable Receive mode:

```
> rx
```

8. On the TX node, transmit n packets (n = 0: infinite):

```
> tx n
```

9. On the RX node, the HyperTerminal displays `n` packets with associated information as described in *Section 3.8.1: PHY common RX test command examples*.

The following table shows an example of the execution steps required for performing a simple TX, RX test with RX in Promiscuous mode (the channel, power setting commands are not reported):

**Table 16. TX and RX test with RX in Promiscuous mode**

| TX node commands | RX node commands |
|---|---|
| **> gettxpacket**<br>len: 0x14<br>0x00: 0x00 0x01 0x02 0x03<br>0x04: 0x04 0x05 0x06 0x07<br>0x08: 0x08 0x09 0x0A 0x0B<br>0x0C: 0x0C 0x0D 0x0E 0x0F<br>0x10: 0x10 0x11 0x12 0x13 | **> getrxconfig**<br>{{(getrxconfig)}<br>{addressFilter:0}<br>{autoAck :0}<br>{coordinator :0}<br>{overflow :0}<br>} |
| | **> showpayload 1** |
| | **> rx**<br>{{(rx)} test start ('e'nd)}<br>#{{(rx)}<br>{num} {oflo} {seq} {per} {err} {lqi} {rssi}{ed} {gain} {status} {time} {fp}{length}} |
| **> tx 2**<br>Txing 2 packets 25500 us apart. 'e'nd...<br>2 packets transmitted.<br>Last packet status: 0x0000<br>Last packet backoff: 0x0000<br>Frame pending in last ACK: No.<br>SFD time (via TX complete callback): 0x62F9 | { { 1} { 0} { 4368} {100} { 0} {0xFF} {-47} {0xD4} {0xB1} {0x4000} {0x000725D3} {0} {0x12} { 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x0B 0x0C 0x0D 0x0E 0x0F 0x10 0x11} }<br>{ { 2} { 0} { 4368} {100} { 0} {0xFF} {-47} {0xD4} {0xB1} {0x4000} {0x00078E9D} {0} {0x12} { 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x0B 0x0C 0x0D 0x0E 0x0F 0x10 0x11} } |

### 4.4.2 TX and RX test with RX in Filter mode (PANID and short address)

1. Setup 2 nodes (TX = transmitter node, RX = receiver node) as described in *Section 2: Getting started*.
2. On both nodes (TX and RX), set the same radio channel (range: [11,26], default: 11) where x is the channel (in hex):

   **> setchannel x**
3. On the TX node, set the radio power (range: [-43,8] dBm, default: 3) where x is the power (in hex):

   **> settxpower x**
4. On the RX node, enable the radio filter and automatic acknowledge and then verify the radio RX configuration:

   **> setaddrfilt 1**
   **> setautoack 1**
   **> getrxconfig**
   {{(getrxconfig)}
   {addressFilter:1}
   {autoAck :1}
   {coordinator :0}
   {overflow :0}
   }
5. On the TX node, set the current packet length and payload with the RX PANID and short ID as destination addresses (FCF: 0x0821), and then verify the packet content.

Example: Build a packet with FCF 0x0821, destination PANID 0x2312 and destination short address 0x2202.

**> setTxLength 0xb**

**> setTxPayload 0 21 08 00 12**

**> setTxPayload 4 23 02 22 F2**

**> setTxPayload 8 41 00 00 00**

**> gettxpacket**

len: 0x0B

0x00: 0x21 0x08 0x00 0x12

0x04: 0x23 0x02 0x22 0xF2

0x08: 0x41 0x00 0x00

6. On the RX node, set filters for PANID and short address.

   Example: Set filters on PANID 0x2312 and short address 0x2202:

   **> setPanId 0x2312**
   **> setShortAddr 0x2202**
7. On the RX node, enable payload display:

   **> showpayload 1**
8. On the RX node, enable Receive mode:

   **> rx**
9. On the TX node, transmit n packets (n = 0: infinite):

   **> tx n**
10. On the RX node, the HyperTerminal displays n packets with associated information as described in *Section 3.8.1: PHY common RX test command examples*.

*Table 17* shows an example of the execution steps required for performing a simple TX and RX test with RX filtering on destination PANID and short address (channel and power setting commands are not reported).

**Table 17. TX and RX test with RX filtering on destination PANID and short address**

| TX node commands | RX node commands |
|---|---|
| **> setTxLength 0xb**<br>**> setTxPayload 0 21 08 00 12**<br>**> setTxPayload 4 23 02 22 F2**<br>**> setTxPayload 8 41 00 00**<br>**> gettxpacket**<br>len: 0x0B<br>0x00: 0x21 0x08 0x00 0x12<br>0x04: 0x23 0x02 0x22 0xF2<br>0x08: 0x41 0x00 0x00 | **> setaddrfilt 1**<br>**> setautoack 1**<br>**> getrxconfig**<br>{{(getrxconfig)}<br>{addressFilter:1}<br>{autoAck :1}<br>{coordinator :0}<br>{overflow :0}<br>} |
| | **> setPanId 0x2312**<br>**> setShortAddr 0x2202** |
| | **> showpayload 1** |
| | **> rx**<br>{{(rx)} test start ('e'nd)}<br>#{{(rx)}<br>{num} {oflo} {seq} {per} {err} {lqi} {rssi}{ed} {gain} {status} {time} {fp}{length}} |
| **> tx 2**<br>Txing 2 packets 25500 us apart. 'e'nd...<br>2 packets transmitted.<br>Last packet status: 0x0000<br>Last packet backoff: 0x0000<br>Frame pending in last ACK: No.<br>SFD time (via TX complete callback): 0x6B6F | { { 1} { 0} {16882} {100} { 0} {0xFF} {-45} {0xDC} {0xB0} {0x6000} {0x00092C5D} {0} {0x09} { 0x21 0x08 0x00 0x12 0x23 0x02 0x22 0xF2 0x41} }<br>{ { 2} { 0} {16882} {100} { 0} {0xFF} {-45} {0xDC} {0xB0} {0x6000} {0x000995F3} {0} {0x09} { 0x21 0x08 0x00 0x12 0x23 0x02 0x22 0xF2 0x41} } |

### 4.4.3 TX and RX test with RX in Coordinator mode

1. Setup 2 nodes (TX = transmitter node, RX = receiver node) as described in *Section 2: Getting started*.
2. On both nodes (TX and RX), set the same radio channel (range: [11,26], default: 11) where x is the channel (in hex):

   ```
   > setchannel x
   ```
3. On the TX node, set the radio power (range: [-43,8] dBm, default: 3) where x is the power (in hex):

   ```
   > settxpower x
   ```
4. On the RX node, enable the Coordinator mode, radio filter and automatic acknowledge and then verify the radio RX configuration:

   ```
   > setaddrfilt 1
   > setautoack 1
   > setcoord 1
   ```

```
> getrxconfig
{{(getrxconfig)}
 {addressFilter:1}
 {autoAck      :1}
 {coordinator  :1}
 {overflow     :0}
 }
```

5. On the TX node, set the current packet length and payload with only the source PANID (FCF:0xC021), and then verify the packet content.

   Example: Build a packet with FCF 0xC021 and source PANID 0x2312:

```
> setTxLength   0xb
> setTxPayload 0 21 C0 00 12
> setTxPayload 4 23 71 72 73
> setTxPayload 8 74 00 00
> gettxpacket
len: 0x0B
0x00: 0x21 0xC0 0x00 0x12
0x04: 0x23 0x71 0x72 0x73
0x08: 0x74 0x00 0x00
```

6. On the RX node, set filters for PANID.

   Example: Set filters on PANID 0x2312.

```
> setPanId 0x2312
```

7. On the RX node, enable the payload display:

```
> showpayload 1
```

8. On the RX node, enable Receive mode:

```
> rx
```

9. On the TX node, transmit n packets (n = 0: infinite):

```
> tx n
```

10. On the RX node, the HyperTerminal displays `n` packets with associated information as described in *Section 3.8.1: PHY common RX test command examples*.

*Table 18* shows an example of the execution steps required for performing a simple TX and RX test with RX in Coordinator mode (channel and power setting commands are not reported).

**Table 18. TX and RX test with RX in Coordinator mode**

| TX node commands | RX node commands |
|---|---|
| **> setTxLength 0xb**<br>**> setTxPayload 0 21 C0 00 12**<br>**> setTxPayload 4 23 71 72 73**<br>**> setTxPayload 8 74 0 0 0**<br>**> gettxpacket**<br>len: 0x0B<br>0x00: 0x21 0xC0 0x00 0x12<br>0x04: 0x23 0x71 0x72 0x73<br>0x08: 0x74 0x00 0x00 | **> setaddrfilt 1**<br>**> setautoack 1**<br>**> setcoord 1**<br>**> getrxconfig**<br>{{(getrxconfig)}<br>{addressFilter:1}<br>{autoAck :1}<br>{coordinator :1}<br>{overflow :0}<br>}<br>} |
| | **> setPanId 0x2312** |
| | **> showpayload 1** |
| | **> rx**<br>{{(rx)} test start ('e'nd)}<br>#{{(rx)}<br>{num} {oflo} {seq} {per} {err} {lqi} {rssi}{ed} {gain} {status} {time} {fp}{length}} |
| **> tx 2**<br>Txing 2 packets 25500 us apart. 'e'nd...<br>2 packets transmitted.<br>Last packet status: 0x0000<br>Last packet backoff: 0x0000<br>Frame pending in last ACK: No.<br>SFD time (via TX complete callback): 0x7F3E | { { 1} { 0} {29811} {100} { 0} {0xFF} {-50} {0xC8} {0xB1} {0x6000} {0x000DE34E} {0} {0x09} { 0x21 0xC0 0x00 0x12 0x23 0x71 0x72 0x73 0x74} }<br>{ { 2} { 0} {29811} {100} { 0} {0xFF} {-50} {0xC8} {0xB1} {0x6000} {0x000E4CE3} {0} {0x09} { 0x21 0xC0 0x00 0x12 0x23 0x71 0x72 0x73 0x74} } |

# 5 Revision history

Table 19. Document revision history

| Date | Revision | Changes |
|---|---|---|
| 30-Jul-2010 | 1 | Initial release. |

**Please Read Carefully:**

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

**UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.**

**UNLESS EXPRESSLY APPROVED IN WRITING BY AN AUTHORIZED ST REPRESENTATIVE, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS "AUTOMOTIVE GRADE" MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER’S OWN RISK.**

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.



STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

**www.st.com**