nRF91M1 AT Commands

Command Reference Guide

v1.0+dev



Contents

Revision history **1**

1 Introduction **2**

2 AT command syntax **3**

3 Generic AT commands **4**
3.1 UART baud rate AT+IPR . . . 4
3.2 Echo E0/E1 . . . 5
3.3 Serial Modem version #XSMVER . . . 6
3.4 SM proprietary command list #XCLAC . . . 8
3.5 Power saving #XSLEEP . . . 9
3.6 Power off #XSHUTDOWN . . . 11
3.7 Reset #XRESET . . . 12
3.8 Modem reset #XMODEMRESET . . . 12
3.9 Device UUID #XUUID . . . 13
3.10 MCUboot bootloader info #XBOOTINFO . . . 14
3.11 Modem fault #XMODEM . . . 16
3.12 Memory statistics #XDBGSTATSMEM . . . 17

4 CMUX AT commands **19**
4.1 CMUX setup +CMUX . . . 19
4.2 CMUX setup #XCMUX . . . 21
4.3 CMUX close down #XCMUXCLD . . . 23
4.4 URC channel #XCMUXURC . . . 24

5 CoAP client AT commands **26**
5.1 CoAP request #XCOAPCREQ . . . 26
5.2 CoAP data pull #XCOAPCDATA . . . 33
5.3 CoAP request cancel #XCOAPCCANCEL . . . 35

6 DFU AT commands **37**
6.1 DFU types . . . 37
6.2 DFU initialize #XDFUINIT . . . 37
6.3 DFU write #XDFUWRITE . . . 39
6.4 DFU apply #XDFUAPPLY . . . 40
6.5 DFU unsolicited notification #XDFU . . . 42
6.6 Complete DFU examples . . . 42
6.7 DFU host example . . . 47

7 FOTA AT commands **49**
7.1 FOTA request #XFOTA . . . 49

8 GNSS AT commands **55**
8.1 Control GNSS #XGNSS . . . 55
8.2 Delete GNSS data #XGNSSDEL . . . 61

9 HTTP client AT commands **64**
9.1 HTTP request #XHTTPCREQ . . . 64
9.2 HTTP data pull #XHTTPCDATA . . . 71
9.3 HTTP request cancel #XHTTPCCANCEL . . . 73

10 ICMP AT commands **75**
10.1 ICMP echo request #XPING . . . 75

11 MQTT client AT commands **78**
11.1 MQTT event notification #XMQTTEVT . . . 78
11.2 MQTT configure #XMQTTCFG . . . 79
11.3 MQTT connect #XMQTTCON . . . 81
11.4 Incoming message notification #XMQTTMSG . . . 84
11.5 MQTT subscribe #XMQTTSUB . . . 84
11.6 MQTT unsubscribe #XMQTTUNSUB . . . 86
11.7 MQTT publish #XMQTTPUB . . . 87

12 nRF Cloud AT commands **89**
12.1 nRF Cloud access #XNRFCLOUD . . . 89
12.2 nRF Cloud location #XNRFCLOUDPOS . . . 93
12.3 nRF Cloud FOTA #XNRFCLOUDFOTA . . . 96
12.4 On-demand upload #XNRFCLOUDOBSUPLOAD . . . 99
12.5 Metrics heartbeat #XNRFCLOUDOBSHEARTBEAT . . . 101
12.6 Chunk forwarding #XNRFCLOUDOBSFORWARD . . . 102
12.7 Core dump upload #XNRFCLOUDOBSCOREDUMP . . . 103
12.8 Automatic upload #XNRFCLOUDOBSAUTO . . . 104

13 nRF Provisioning AT commands **107**
13.1 Trigger provisioning #XNRFPROV . . . 107

14 PPP AT commands **110**
14.1 Enter data state +CGDATA . . . 110
14.2 Control PPP #XPPP . . . 111

15 SMS AT commands **117**
15.1 SMS support #XSMS . . . 117

16 Socket AT commands **120**
16.1 Socket #XSOCKET . . . 120
16.2 Secure socket #XSSOCKET . . . 123
16.3 Close socket #XCLOSE . . . 126
16.4 Socket options #XSOCKETOPT . . . 128
16.5 Secure socket options #XSSOCKETOPT . . . 130
16.6 Socket binding #XBIND . . . 133
16.7 Connection #XCONNECT . . . 133
16.8 Send data #XSEND . . . 135
16.9 Receive data #XRECV . . . 138
16.10 UDP send data #XSENDTO . . . 139
16.11 UDP receive data #XRECVFROM . . . 143

16.12 Asynchronous socket polling #XAPOLL 144
16.13 Configure socket receive #XRECVCFG 150
16.14 Resolve hostname #XGETADDRINFO 156
16.15 Socket listen #XLISTEN 157
16.16 Socket accept #XACCEPT 159

17 Trace AT commands **163**
17.1 Application log AT#XLOG 163
17.2 Modem trace AT#XTRACE 165

18 Data mode **167**
18.1 Overview 167
18.2 Data mode control #XDATACTRL 169
18.3 Exit data mode notification #XDATAMODE 170

A UART configuration **172**
A.1 UART interface 172
A.2 nRF91M1 module pin assignment 173
A.3 Controlling the UART power state 174
A.4 Operating without hardware flow control 177
A.5 Operating with the nRF9151 DK 178
A.6 UART security considerations 179

B Logging and tracing **181**
B.1 Application logging 181
B.2 Modem traces 182
B.3 nRF Cloud observability 183

C Cellular PPP modem **185**
C.1 Overview 185
C.2 PPP and CMUX 185
C.3 Operational notes 188

D Release notes **189**

Revision history

Date	Version	Description
	1.x	Major update. • Updated and fixed several things.
07-Oct-2026	1.0	First release. • Document proprietary application-level AT commands for the nRF91M1 module. • Supported modem firmware: `mfw_nrf91x1` v2.0.4 and `mfw_nrf9151-ntn` v1.0.1 (see *Introduction*).

1 Introduction

This document describes the AT commands used to control the nRF91M1 module. This allows you to run your application on a separate host MCU. The module accepts both the modem specific AT commands and proprietary application level AT commands. The AT commands are documented in the following guides:

- nRF91x1 modem specific AT commands
  - `mfw_nrf91x1` - nRF91x1 AT Commands Reference Guide
  - `mfw_nrf9151-ntn` - nRF91x1 NTN AT Commands Reference Guide
- Proprietary application level AT commands are documented in this document.

The nRF91M1 module supports the following modem firmware:

- `mfw_nrf91x1` v2.0.4 (pre-programmed on the module)
- `mfw_nrf9151-ntn` v1.0.1 (can be updated using the *DFU AT commands*)

2 AT command syntax

The AT Commands have standardized syntax rules.

Words enclosed in <angle brackets> are references to syntactical elements. Words enclosed in [square brackets] represent optional items that can be left out of the command line at the specified point. The brackets are not used when the words appear in the command line.

<CR>, <LF> and <CR><LF> are allowed in an AT command sent by an application.

A string type parameter input must be enclosed between quotation marks (`"string"`).

There are 3 types of AT commands:

- Set command `<CMD>[=...]`. Set commands set values or perform actions.
- Read command `<CMD>?`. Read commands check the current values of subparameters.
- Test command `<CMD>=?`. Test commands test the existence of the command and provide information about the type of its subparameters. Some test commands can also have other functionality.

AT responds to all commands with a final response.

The maximum AT command length is 8190 bytes, including the terminator character.

3 Generic AT commands

This page describes generic AT commands.

3.1 UART baud rate AT+IPR

The AT+IPR command sets the UART baud rate.

3.1.1 Set command

The set command sets the UART baud rate. nRF91M1 module does not support automatic baud rate detection. When the device resets, the baud rate is set to the default value set in the devicetree.

Syntax

```
AT+IPR=<baud_rate>
```

The parameters and their defined values are the following:

<baud_rate>
Integer. The desired baud rate.

Note:

The baud rate change takes effect after the modem responds with OK. The host must switch to the new baud rate to continue communication.

Example

Set the baud rate to 115200.

```
AT+IPR=115200
OK
```

3.1.2 Read command

The read command reads the current UART baud rate.

Syntax

```
AT+IPR?
```

Response syntax

```
+IPR: <baud_rate>
```

Example

```
AT+IPR?
+IPR: 115200
OK
```

3.1.3 Test command

The test command lists the supported baud rates.

Syntax

```
AT+IPR=?
```

Response syntax

```
+IPR: (list of supported autodetectable baud rates)[,(list of fixed-
only baud rates)]
```

Example

```
AT+IPR=?
+IPR: (),(115200,230400,460800,921600,1000000)
OK
```

3.2 Echo E0/E1

The E command enables or disables the AT command echo feature of the nRF91M1 module.

3.2.1 Set command

The set command enables or disables the AT command echo feature. While enabled, the nRF91M1 module echoes back all the characters that are received in the AT command mode.

Note:

When working with a terminal, you should disable local echo to avoid double echoing of characters.

Syntax

```
ATE1 - Enable AT command echo.
ATE0 - Disable AT command echo (default).
```

Example

Set the AT command echo:

```
ATE1
OK

AT
AT    // Echoed command
OK

ATE0
ATE0  // Echoed command
OK

AT
OK
```

3.2.2 Read command

The read command is not supported.

3.2.3 Test command

The test command is not supported.

3.3 Serial Modem version #XSMVER

The #XSMVER command returns the versions of the Serial Modem application firmware and nRF Connect SDK in which the application firmware is built.

3.3.1 Set command

The set command returns the versions of the Serial Modem application firmware and nRF Connect SDK.

Syntax

```
AT#XSMVER
```

Response syntax

```
#XSMVER: "<sm_version>","<ncs_version>"[,"<customer_version>"]
```

The parameters and their defined values are the following:

<sm_version>
String.

The version of the Serial Modem application firmware in `v<major>.<minor>.<patch>` format.

Hosts can use the version to verify that application DFU or FOTA moved to a newer image. To check, compare the AT#XSMVER response before and after the update.

<ncs_version>
String. The version of the nRF Connect SDK in `<major>.<minor>.<patch>` format.

<customer_version>
String. This parameter is missing on official release builds. It is used in special cases such as FOTA-TEST builds.

Example

The following command examples read the versions:

```
// v2.0.0 release
AT#XSMVER
#XSMVER: "v2.0.0","3.4.1"
OK
```

```
// FOTA test package for v2.0.0 release
AT#XSMVER
#XSMVER: "v2.0.0","3.4.1","FOTA-TEST"
OK
```

3.3.2 Read command

The read command is not supported.

3.3.3 Test command

The test command is not supported.

3.4 SM proprietary command list #XCLAC

The #XCLAC command requests the list of the proprietary nRF91M1 module commands.

3.4.1 Set command

The set command requests the list of the proprietary nRF91M1 module commands. It is an add-on for AT+CLAC, which lists all modem AT commands.

Syntax

```
AT#XCLAC
```

Response syntax

```
<command list>
```

The parameters and their defined values are the following:

<command list>
String. A list of values representing all the #X* commands followed by <CR><LF>.

Example

```
AT#XCLAC
AT#XSMVER
AT#XSLEEP
AT#XCLAC
AT#XSOCKET
AT#XBIND
AT#XCONNECT
AT#XSEND
AT#XRECV
AT#XSENDTO
AT#XRECVFROM
AT#XPING
AT#XGNSS
OK
```

3.4.2 Read command

The read command is not supported.

3.4.3 Test command

The test command is not supported.

3.5 Power saving #XSLEEP

The #XSLEEP command makes the nRF91M1 module to enter idle or sleep mode.

Note:

The #XSLEEP command is intended for experimentation and power consumption measurements and must not be used in production.

You can use the DTR (Data Terminal Ready) and RI (Ring Indicator) signals to control the power state of the UART between the nRF91M1 module and the host. See the UART configuration section for more information about DTR and RI.

Note:

If you want to do power measurements on the nRF91 Series development kit while running the nRF91M1 module, remember to disable unused peripherals.

3.5.1 Set command

The set command makes the nRF91M1 module enter either Idle or Sleep mode.

Syntax

```
AT#XSLEEP=<sleep_mode>
```

The parameters and their defined values are the following:

<sleep_mode>

- 1 - Enter Sleep. In this mode, both the nRF91M1 module service and the LTE connection are terminated.

  nRF91M1 module can be woken up using the DTR pin.

- 2 - Enter Idle. In this mode, both the nRF91M1 module service and the LTE connection are maintained, but the UART is disabled to save power. Received data is buffered and sent to the host after idle mode is exited.

  nRF91M1 module can exit the idle mode using the DTR pin. When the nRF91M1 module is in idle mode, and there is data to be read by the host, the RI pin is asserted for a short period of time to notify the host. The host can then deassert and assert DTR to exit idle mode and read the data.

Note:

- If the modem is on, entering Sleep mode (by issuing `AT#XSLEEP=1`) sends a `+CFUN=0` command to the modem, which causes a write to non-volatile memory (NVM). Take the NVM wear into account, or put the modem in flight mode by issuing `AT+CFUN=4` before Sleep mode.

Examples

```
AT#XSLEEP=0
ERROR
```

```
AT#XSLEEP=1
OK
```

See the following for an example of when the modem is on:

```
AT+CFUN=4
OK

AT#XSLEEP=1
OK
```

```
AT#XSLEEP=2
OK
```

3.5.2 Read command

The read command is not supported.

3.5.3 Test command

The test command tests the existence of the AT command and provides information about the type of its subparameters.

Syntax

```
AT#XSLEEP=?
```

Response syntax

```
#XSLEEP: <list of shutdown_mode>
```

Example

```
AT#XSLEEP=?
#XSLEEP: (1,2)
OK
```

3.6 Power off #XSHUTDOWN

The #XSHUTDOWN command makes the nRF91 Series SiP enter System OFF mode, which is the deepest power saving mode.

3.6.1 Set command

The set command makes the nRF91 Series SiP enter System OFF mode.

Syntax

```
AT#XSHUTDOWN
```

Note:

In this case the nRF91 Series SiP cannot be woken up using the DTR pin.

Example

```
AT#XSHUTDOWN
OK
```

3.6.2 Read command

The read command is not supported.

3.6.3 Test command

The test command is not supported.

3.7 Reset #XRESET

The #XRESET command performs a soft reset of the nRF91 Series SiP.

3.7.1 Set command

The set command resets the nRF91 Series SiP.

Syntax

```
AT#XRESET
```

Example

```
AT#XRESET
OK
Ready
```

3.7.2 Read command

The read command is not supported.

3.7.3 Test command

The test command is not supported.

3.8 Modem reset #XMODEMRESET

The #XMODEMRESET command performs a reset of the modem.

The modem is set to minimal function mode (via +CFUN=0) before being reset. The nRF91M1 module is not restarted. After the command returns, the modem will be in minimal function mode.

3.8.1 Set command

The set command resets the modem.

Syntax

```
AT#XMODEMRESET
```

Response syntax

```
#XMODEMRESET: <result>[,<error_code>]
```

The parameters and their defined values are the following:

<result>

- `0` - Success.
- *Positive value* - On failure, indicates the step that failed.

<error_code>
Integer. Only printed when the modem reset was not successful. It is the error code indicating the reason for the failure.

Example

```
AT#XMODEMRESET

#XMODEMRESET: 0

OK
```

3.8.2 Read command

The read command is not supported.

3.8.3 Test command

The test command is not supported.

3.9 Device UUID #XUUID

The `#XUUID` command requests the device UUID.

3.9.1 Set command

The set command returns the device UUID.

Syntax

```
AT#XUUID
```

Response syntax

```
#XUUID: <device-uuid>
```

The parameters and their defined values are the following:

<device-uuid>
String. The UUID of the device.

Example

```
AT#XUUID

#XUUID: 50503041-3633-4261-803d-1e2b8f70111a

OK
```

3.9.2 Read command

The read command is not supported.

3.9.3 Test command

The test command is not supported.

3.10 MCUboot bootloader info #XBOOTINFO

Query information about the MCUboot second-stage bootloader.

This command is only available when the firmware is built with the NSIB (B0) and MCUboot as a second-stage bootloader.

3.10.1 Set command

```
AT#XBOOTINFO=<op>
```

The parameters and their defined values are the following:

<op>

- `0` - Query the firmware version of the active MCUboot slot.
- `1` - Query which MCUboot slot is currently active.

Response syntax

```
#XBOOTINFO: <value>
```

The parameters and their defined values are the following:

<value>
Integer.

- When <op> is 0, it is an unsigned integer representing the firmware version of the active MCUboot slot, used by NSIB as a downgrade-protection counter.
- When <op> is 1, it is 0 if slot s0 is active and 1 if slot s1 is active.

Examples

Query the firmware version of the active MCUboot slot:

```
AT#XBOOTINFO=0

#XBOOTINFO: 1

OK
```

Query the active MCUboot slot:

```
AT#XBOOTINFO=1

#XBOOTINFO: 0

OK
```

3.10.2 Test command

```
AT#XBOOTINFO=?
```

Response syntax

```
#XBOOTINFO: (0,1)
```

Example

```
AT#XBOOTINFO=?
#XBOOTINFO: (0,1)
OK
```

3.11 Modem fault #XMODEM

The application monitors the modem status. When the application detects a *modem fault*, it sends the #XMODEM unsolicited notification.

3.11.1 Unsolicited notification

The application sends the following unsolicited notification when it detects a modem fault:

```
#XMODEM: FAULT,<reason>,<program_count>
```

The parameters and their defined values are the following:

<reason>
Integer. A hexadecimal integer indicating the reason of the modem fault.

<program_count>
Integer. A hexadecimal integer indicating the address of the modem fault.

The application sends the following unsolicited notification when it shuts down libmodem:

```
#XMODEM: SHUTDOWN,<result>
```

The parameters and their defined values are the following:

<result>
Integer. The result of the shutdown of libmodem.

The application sends the following unsolicited notification when it re-initializes libmodem:

```
#XMODEM: INIT,<result>
```

The parameters and their defined values are the following:

<result>
Integer. The result of the re-initialization of libmodem.

Note:

After libmodem is re-initialized, the MCU side must restart the current active service as follows:

1. Stopping the service. For example, disconnecting the TCP connection and closing the socket.
2. Connecting again using LTE.
3. Restarting the service. For example, opening the socket and re-establishing the TCP connection.

3.12 Memory statistics #XDBGSTATSMEM

Note:

This AT command is Experimental.

Print heap and thread stack memory statistics to the log at the `INF` level. Since this is using the logging as an output, you must enable logging first by issuing the `AT#XLOG=1` command.

The following statistics are printed:

- System heap (`malloc` heap) - Free bytes, allocated bytes, and peak allocated bytes.
- Kernel heap (`k_malloc` heap) - Free bytes, allocated bytes, and peak allocated bytes.

3.12.1 Set command

```
AT#XDBGSTATSMEM
```

The command prints memory statistics to the log and returns `OK`. No memory information is returned in the AT response.

3.12.2 Read command

The read command is not supported.

3.12.3 Test command

The test command is not supported.

3.12.4 Example

Query memory statistics:

```
AT#XLOG=1
OK
AT#XDBGSTATSMEM
OK
```

These commands will print the following output in the log:

```
[00:00:36.270,965] <inf> sm_at: System heap stats:
[00:00:36.270,996] <inf> sm_at:   free:            45844
[00:00:36.270,996] <inf> sm_at:   allocated:         472
[00:00:36.271,026] <inf> sm_at:   max. allocated:    472
[00:00:36.271,026] <inf> sm_at: Kernel heap stats:
```

(continues on next page)

(continued from previous page)

```
[00:00:36.271,057] <inf> sm_at:   free:             1892
[00:00:36.271,057] <inf> sm_at:   allocated:           0
[00:00:36.271,057] <inf> sm_at:   max. allocated:   1280
```

4 CMUX AT commands

This page describes CMUX-related AT commands.

The GSM 0710 multiplexer protocol (CMUX) enables multiplexing multiple data streams through a single serial link, setting up one channel per data stream. For example, it can be used to exchange AT data and have a Point-to-Point Protocol (PPP) link up at the same time on a single UART. nRF91M1 module implements the basic option of the CMUX protocol with only UIH frames as described in the 3GPP TS 27.010 specification.

The maximum length of the information field in UIH frames is 127 bytes.

Note:

To use the nRF91 Series SiP as a cellular dial-up PPP modem, see Cellular PPP modem.

CMUX is enabled in nRF91M1 module by compiling it with the appropriate configuration files, depending on your use case. See the Configuration files section for more information.

Note:

- Only basic mode (mode 0) is supported.
- Only UIH frames are used.
- The speed used is the configured baud rate of nRF91M1 module's UART.
- No system parameters (N1, T1, T2, T3, k, m) are configurable.

4.1 CMUX setup +CMUX

The `AT+CMUX` command starts the CMUX multiplexer. It is defined in 3GPP TS 27.007 (section 5.7) and the underlying protocol is specified in 3GPP TS 27.010.

Only basic mode (`<mode>=0`) with subset `0` is supported. All other parameter values are rejected.

4.1.1 Set command

The set command allows you to start the CMUX multiplexer.

An `OK` response is sent before CMUX is started, after which only CMUX framing is accepted on the serial link.

Syntax

```
AT+CMUX=<mode>[,<subset>]
```

The parameters and their defined values are the following:

<mode>
Integer. Selects the operation mode. Only `0` (basic mode) is supported.

<subset>
Integer. Selects the subset of mode 0. Only `0` is supported. Default value is `0`.

4.1.2 Read command

The read command returns the current multiplexer configuration.

Syntax

```
AT+CMUX?
```

Response syntax

```
+CMUX: <mode>,<subset>
```

The parameters and their defined values are the following:

<mode>
Integer. Always `0` (basic mode).

<subset>
Integer. Always `0`.

4.1.3 Test command

The test command returns the supported parameter ranges.

Syntax

```
AT+CMUX=?
```

Response syntax

```
+CMUX: (0),(0)
```

4.1.4 Example

```
AT+CMUX=?

+CMUX: (0),(0)

OK
AT+CMUX?

+CMUX: 0,0

OK
AT+CMUX=0

OK
// CMUX is now started. Open the CMUX channels to continue
communication.
AT+CMUX=0,0

OK
// Equivalent to AT+CMUX=0.
```

4.2 CMUX setup #XCMUX

Note:

AT#XCMUX is a compatibility command for Serial Modem v1.x.x and is not recommended for new designs. Use the standard AT+CMUX=0 command instead.

The #XCMUX command manages the configuration of CMUX over the serial link.

4.2.1 Set command

The set command allows you to start CMUX and assign the CMUX channels.

The CMUX link is closed down automatically when the remote end sends the CMUX Multiplexer Close Down (CLD) sequence. The CMUX link can be closed manually with the AT#XCMUXCLD command.

Syntax

```
AT#XCMUX[=<AT_channel>]
```

The parameters and their defined values are the following:

<AT_channel>
Integer. The address of the AT channel. The AT channel denotes the CMUX channel

where AT data (commands, responses, notifications) is exchanged. If specified, it must be 1, unless PPP is enabled. If PPP is enabled, it can also be 2 (to allocate the first CMUX channel to PPP). If not specified, the previously used address is used. If no address has been previously specified, the default address is 1.

Note:

If there is more than one CMUX channel (such as when using PPP), the non-AT channels will automatically get assigned to addresses other than the one used for the AT channel. For example, if PPP is enabled and CMUX is started with the AT#XCMUX=2 command, the AT channel will be assigned to address 2 and the PPP channel to address 1.

An OK response is sent if the command is accepted, after which CMUX is started. This means that after successfully running this command, you must set up the CMUX link and open the channels appropriately. The AT channel will be available at the configured address. You can send an empty AT command to make sure that the protocol is set up properly.

Caution:

Using AT#XCMUX to switch the AT command channel at runtime is error-prone and not recommended. The channel switching might run out of sync with automatic PPP startup and recovery enabled by AT#XPPP=1.

4.2.2 Read command

The read command allows you to read the address of the AT channel and the total number of channels.

Syntax

```
AT#XCMUX?
```

Response syntax

```
#XCMUX: <AT_channel>,<channel_count>
```

The parameters and their defined values are the following:

<AT_channel>
Integer. The address of the AT channel. It is between 1 and <channel_count>.

<channel_count>
Integer. The total number of CMUX channels. It depends on what features are enabled (for example, PPP).

4.2.3 Example

Without PPP:

```
AT#XCMUX?

#XCMUX: 1,1

OK
AT#XCMUX

OK
// Here, CMUX is started and communication can now happen only
through it (until a reset).
// Open the AT channel, which is the only one, to continue
exchanging AT data.
AT

OK
```

With PPP:

```
AT#XCMUX?

#XCMUX: 1,2

OK
AT#XCMUX=2

OK
// Start up CMUX and open the channels. The AT channel is now at
address 2.
AT#XCMUX?

#XCMUX: 2,2

OK
```

4.3 CMUX close down #XCMUXCLD

The #XCMUXCLD command closes down the CMUX service.

This command can be used on host devices that do not support sending the CMUX Multiplexer close-down sequence. Once CMUX is closed down, the serial link returns to AT command mode.

4.3.1 Set command

The set command allows you to close down the CMUX link.

Syntax

```
AT#XCMUXCLD
```

An `OK` response is sent in the CMUX channel if the command is accepted, after which CMUX is closed down.

4.3.2 Read command

The read command is not supported.

4.4 URC channel #XCMUXURC

The `#XCMUXURC` command configures the CMUX DLC channel to which URCs are routed.

By default (channel `0`), URCs are sent to the first open AT channel that is not in data mode. Routing URCs to a fixed channel is recommended when multiple CMUX channels are in use, for example, when one channel is occupied by a PPP session.

4.4.1 Set command

The set command sets the CMUX channel used for URC delivery.

Syntax

```
AT#XCMUXURC=<channel>
```

The parameters and their defined values are the following:

<channel>

- `0` - Auto-select: URCs are sent to the first open AT channel not in data mode.
- `99` - All channels: URCs are sent to all open AT channels not in data mode.
- Any other value routes URCs to the specified DLC channel.

4.4.2 Read command

The read command returns the currently configured URC channel.

Syntax

```
AT#XCMUXURC?
```

Response syntax

```
#XCMUXURC: <channel>
```

The parameters and their defined values are the following:

<channel>
Integer. The currently configured URC channel (see the set command).

4.4.3 Test command

The test command returns the parameter description.

Syntax

```
AT#XCMUXURC=?
```

Response syntax

```
#XCMUXURC=<channel>
```

4.4.4 Example

Route URCs to DLC channel 2 before starting CMUX:

```
AT#XCMUXURC=2

OK
AT#XCMUXURC?

#XCMUXURC: 2

OK
AT+CMUX=0

OK
// CMUX is now started. URCs are delivered on DLC channel 2.
```

5 CoAP client AT commands

This page describes AT commands for the CoAP client. The CoAP client operates on sockets managed by the *Socket AT commands*. You can perform the following using the Socket AT commands:

- Create a UDP socket with AT#XSOCKET or AT#XSSOCKET.
- Connect it with AT#XCONNECT.
- Set socket options with AT#XSOCKETOPT or AT#XSSOCKETOPT.
- Close with AT#XCLOSE.

Only one CoAP request might be active at a time across all sockets.

5.1 CoAP request #XCOAPCREQ

The #XCOAPCREQ command starts an asynchronous CoAP request on a connected UDP socket.

5.1.1 Set command

The set command sends a CoAP request and returns OK immediately. The server's response is delivered through unsolicited #XCOAPCDATA and #XCOAPCSTAT notifications.

Syntax

```
AT#XCOAPCREQ=<handle>,<path>,<method>[,<auto_reception>[,<format>[,
<confirmable>[,<content_format>[,<payload_len>[,<opt_num_1>,<opt_
val_1>[,<opt_num_2>,<opt_val_2>[...]]]]]]]]
```

The parameters and their defined values are the following:

<handle>
Integer. Identifies the connected socket handle returned by AT#XSOCKET and connected by AT#XCONNECT.

<path>
String. The URI-path of the resource, for example "/sensors/temperature".

<method>

- 1 - GET.

- 2 - POST.
- 3 - PUT.
- 4 - DELETE.
- 5 - FETCH.
- 6 - PATCH.
- 7 - iPATCH.

<auto_reception>
Integer. Optional. When omitted, automatic reception is used.

- `1` - Automatic mode (default). Response bytes are forwarded to the host automatically through `#XCOAPCDATA` URCs.
- `0` - Manual mode. The firmware buffers each response block and emits `#XCOAPCHEAD`. The host must then retrieve the block using `AT#XCOAPCDATA`.

<format>
Integer. Optional. Controls the encoding used to deliver response payload bytes to the host.

- `0` - Binary (default). Response payload bytes are forwarded to the host as raw binary.
- `1` - Hex string. Response payload bytes are encoded as a lowercase ASCII hex string before delivery. Each raw byte becomes two hex characters. The `<length>` field in `#XCOAPCDATA` URCs and responses always reports the raw byte count. The actual data transmitted to the host is `2×<length>` ASCII hex characters.

<confirmable>
Integer. Optional.

- `0` - Non-confirmable (NON) message (default).
- `1` - Confirmable (CON) message.

<content_format>
Integer. Optional. The CoAP Content-Format option number for the request payload (default `0` = `text/plain`). Common values:

- `0` - text/plain.
- `42` - application/octet-stream.
- `50` - application/json.
- `60` - application/cbor.

<payload_len>
Integer. Optional. When `0` or omitted and no option pairs follow, no request payload is sent and the command returns `OK` immediately. A positive integer specifies the total payload length in bytes and the command then returns `OK` and enters data mode.

Note:

When option pairs follow, <payload_len> must be written explicitly. Use 0 if no payload is needed. Omitting it shifts the option pairs one position left, which always causes the command to return ERROR.

The host must send exactly this many bytes as the request payload. Data mode exits and #XDATAMODE: 0 is reported when all bytes have been consumed. Data mode is the only mechanism for supplying the payload - there is no inline parameter alternative.

The exact sending behavior depends on payload size:

- Small payload (≤ 512 bytes) - All bytes are buffered until data mode exits, then the CoAP request is sent as a single packet.
- Large payload (> 512 bytes) - The CoAP request starts transmitting while data mode is still active. As soon as the first full block of 512 bytes has arrived, the first CoAP Block1 packet is sent to the server. Each subsequent block is sent as more data arrives, so the server exchange runs concurrently with the host's serial upload.

<opt_num_X>,<opt_val_X>

Optional CoAP option pairs.

- When present, <payload_len> must also be present as the positional separator before the first pair.
- <opt_num_X> is a decimal integer option number.
- <opt_val_X> is a quoted string.
- A value prefixed with 0x or 0X is decoded as raw hex bytes (for example, "0x3c" → byte 0x3c).
- The hex part after the prefix must be non-empty and even-length, otherwise, the command returns ERROR.
- Any value without the 0x prefix is used verbatim, including strings that look like hex (for example "abcd" → 4 ASCII bytes).
- Common option numbers:
  - 6 - Observe ("0x00" = register, "0x01" = deregister).
  - 17 - Accept (content format the client will accept, hex-encoded).
  - 35 - Proxy-Uri (full URI passed to a proxy, verbatim string).

Examples:

```
17,"0x00"                Accept: text/plain (0x00)
17,"0x3c"                Accept: application/cbor (0x3c = 60)
35,"coap://proxy/res"    Proxy-Uri verbatim string
6,"0x00",17,"0x3c"       Observe register + Accept cbor
```

Note:

The Observe option (option 6) with the register value "0x00" starts a persistent subscription. The request stays active and the firmware delivers each server notification through #XCOAPCDATA (automatic mode) or #XCOAPCHEAD (manual mode). No #XCOAPCSTAT is emitted between notifications. To stop the subscription, use AT#XCOAPCCANCEL. You cannot send a separate deregister request (6,"0x01") while the subscription is active, because only one CoAP request can be active at a time.

Unsolicited notification

#XCOAPCDATA is emitted in automatic mode for each received response block.

```
#XCOAPCDATA: <handle>,<offset>,<length>
```

The notification line is terminated with \r\n, the response data follows immediately, and a \r\n terminator follows the data bytes. In binary mode (default), the data is <length> raw bytes. In hex mode (<format>=1), the data is 2×<length> lower-case ASCII hex characters.

The parameters and their defined values are the following:

<handle>
Integer. Identifies the socket.

<offset>
Integer. The number of response bytes already delivered before this block.

<length>
Integer. The raw byte count of the response data in this block. In hex mode (<format>=1), the payload following the header line is 2×<length> ASCII hex characters.

#XCOAPCHEAD is emitted in manual mode for each received response block.

```
#XCOAPCHEAD: <handle>,<code>,<block_len>
```

The parameters and their defined values are the following:

<handle>
Integer. Identifies the socket.

<code>
Integer. The CoAP response code for this block.

<block_len>
Integer. The raw byte count of the buffered block. In hex mode (<format>=1), AT#XCOAPCDATA will deliver 2×<block_len> ASCII hex characters. The host must call AT#XCOAPCDATA to retrieve this block before the next one arrives.

#XCOAPCSTAT is emitted when the request completes, fails, or is cancelled.

```
#XCOAPCSTAT: <handle>,<status_code>,<total_bytes>
```

The parameters and their defined values are the following:

<handle>
Integer. Identifies the socket.

<status_code>
Integer. The CoAP response code on success, or `-1` on failure or cancel. CoAP response codes are encoded as `(class << 5) | detail`. For example, `2.05 Content` is encoded as 69 and `2.04 Changed` as 68.

<total_bytes>
Integer. The total number of raw response payload bytes received. In hex mode (`<format>=1`), this is the raw byte count, not the number of hex characters delivered.

Note:

Block2 response transfer (server-side fragmentation of large responses) is handled automatically by the CoAP client library. All blocks are forwarded to the host in sequence. `#XCOAPCSTAT` is emitted only after the final block is received and delivered.

Examples

CoAP GET (automatic mode, binary):

```
AT#XSOCKET=1,2,0
#XSOCKET: 0,2,17
OK

AT#XCONNECT=0,"coap.example.com",5683
#XCONNECT: 0,1
OK

AT#XCOAPCREQ=0,"/sensors/temperature",1
OK

#XCOAPCDATA: 0,0,12
23.5 Celsius

#XCOAPCSTAT: 0,69,12

AT#XCLOSE=0
OK
```

Note:

In automatic mode, a `\r\n` terminator is appended after the data bytes of each

`#XCOAPCDATA` notification.

CoAP GET (automatic mode, hex-encoded response, `format=1`):

```
AT#XCOAPCREQ=0,"/sensors/temperature",1,1,1
OK

#XCOAPCDATA: 0,0,12
32332e352043656c73697573

#XCOAPCSTAT: 0,69,12
```

Note:

The hex string and the line break after it represent the `2×<length>` hex characters followed by the `\r\n` terminator.

CoAP GET (manual mode):

```
AT#XCOAPCREQ=0,"/sensors/temperature",1,0
OK

#XCOAPCHEAD: 0,69,12

AT#XCOAPCDATA=0
#XCOAPCDATA: 0,0,12
23.5 Celsius
OK

#XCOAPCSTAT: 0,69,12
```

CoAP POST with JSON payload (`content_format=50` = `application/json`):

```
AT#XCOAPCREQ=0,"/data",2,1,0,0,50,13
OK
{"value": 42}
#XDATAMODE: 0

#XCOAPCSTAT: 0,68,0
```

CoAP GET through proxy with Proxy-URI option (option 35, value is a plain URI string):

```
AT#XCOAPCREQ=0,"proxy",1,1,0,0,0,0,35,"coap://remote-host/resource"
OK
```

(continues on next page)

(continued from previous page)

```
#XCOAPCDATA: 0,0,128
<128 bytes>

#XCOAPCSTAT: 0,69,128
```

CoAP GET with Observe and Accept options (option 6 = Observe register `"0x00"`, option 17 = Accept `"0x3c"` = 0x3c = application/cbor):

```
AT#XCOAPCREQ=0,"/obs",1,1,0,1,0,0,6,"0x00",17,"0x3c"
OK

#XCOAPCDATA: 0,0,18
<18 bytes CBOR>

#XCOAPCDATA: 0,0,18
<18 bytes CBOR>

AT#XCOAPCCANCEL=0
OK

#XCOAPCSTAT: 0,-1,0
```

The subscription stays active and delivers a `#XCOAPCDATA` notification for each server update until you cancel it with `AT#XCOAPCCANCEL`. The `#XCOAPCSTAT` notification is emitted only when the subscription ends.

5.1.2 Test command

The test command tests the existence of the command and provides information about the type of its subparameters.

Syntax

```
AT#XCOAPCREQ=?
```

Response syntax

```
#XCOAPCREQ: <handle>,<path>,<method>[,<auto_reception>[,<format>[,
<confirmable>[,<content_format>[,<payload_len>[,<opt_num>,<opt_val>.
..]]]]]]
```

Example

```
AT#XCOAPCREQ=?
#XCOAPCREQ: <handle>,<path>,<method>[,<auto_reception>[,<format>[,
<confirmable>[,<content_format>[,<payload_len>[,<opt_num>,<opt_val>.
..]]]]]]
OK
```

5.2 CoAP data pull #XCOAPCDATA

The #XCOAPCDATA command pulls a buffered response block for a manual-mode CoAP request.

5.2.1 Set command

The set command delivers one buffered response block to the host and unblocks the CoAP client to request the next Block2 block, if any.

Syntax

```
AT#XCOAPCDATA=<handle>[,<length>]
```

The parameters and their defined values are the following:

<handle>
Integer. Identifies the socket used when starting the manual-mode request.

<length>
Integer. Optional. The maximum number of bytes to return in this pull.

When omitted, the full block buffer size (512 bytes) is used.

Response syntax

When a block is available:

```
#XCOAPCDATA: <handle>,<offset>,<length>
<data>
OK
```

In binary mode (default), `<data>` is `<length>` raw bytes. In hex mode (`<format>=1`), `<data>` is 2×`<length>` lower-case ASCII hex characters.

The `#XCOAPCDATA:` line is terminated with `\r\n`. `<length>` raw response bytes follow immediately with no additional separator. `OK` follows on its own line after the response bytes.

When no block has been buffered yet (the firmware is still waiting for the server response).

```
#XCOAPCDATA: <handle>,<offset>,<length>
OK
```

The parameters and their defined values are the following:

<handle>
Integer. Identifies the socket.

<offset>
Integer. The number of response bytes already delivered before this pull.

<length>
Integer. The number of response bytes delivered in this pull. A value of `0` means no block is ready yet.

After the last block is drained, `#XCOAPCSTAT` is emitted as a URC after the final `OK`.

Note:

If `AT#XCOAPCDATA` is not called within 30 seconds of a `#XCOAPCHEAD` notification, the request is aborted and `#XCOAPCSTAT: <handle>,-1,<total_bytes>` is emitted.

Example

```
AT#XCOAPCREQ=0,"/large",1,0
OK

#XCOAPCHEAD: 0,69,512

AT#XCOAPCDATA=0,512
#XCOAPCDATA: 0,0,512
<512 bytes>
OK

#XCOAPCHEAD: 0,69,512

AT#XCOAPCDATA=0,512
#XCOAPCDATA: 0,512,512
<512 bytes>
OK

#XCOAPCSTAT: 0,69,1024
```

5.2.2 Test command

The test command tests the existence of the command and provides information about the type of its subparameters.

Syntax

```
AT#XCOAPCDATA=?
```

Response syntax

```
#XCOAPCDATA: <handle>[,<length>]
```

Example

```
AT#XCOAPCDATA=?
#XCOAPCDATA: <handle>[,<length>]
OK
```

5.3 CoAP request cancel #XCOAPCCANCEL

The #XCOAPCCANCEL command cancels the active CoAP request. It is also the mechanism for stopping a CoAP Observe subscription started with the Observe register option (`6,"0x00"`).

5.3.1 Set command

The set command cancels the currently active CoAP request.

Syntax

```
AT#XCOAPCCANCEL=<handle>
```

The parameters and their defined values are the following:

<handle>
: Integer. Must match the socket handle that was passed to AT#XCOAPCREQ when the request was started. This acts as an ownership assertion: the command returns ERROR if no request is active or if the active request belongs to a different handle.

Note:

Only one CoAP request can be active at a time. The <handle> check is an ownership guard, not a per-socket filter. Cancellation always applies to the single in-flight request.

An unsolicited #XCOAPCSTAT: <handle>,-1,<total_bytes> notification is emitted after cancellation, where <total_bytes> is the number of response bytes already delivered to the host.

Example

```
AT#XCOAPCCANCEL=0
OK
#XCOAPCSTAT: 0,-1,0
```

5.3.2 Test command

The test command tests the existence of the command and provides information about the type of its subparameters.

Syntax

```
AT#XCOAPCCANCEL=?
```

Response syntax

```
#XCOAPCCANCEL: <handle>
```

Example

```
AT#XCOAPCCANCEL=?
#XCOAPCCANCEL: <handle>
OK
```

6 DFU AT commands

This page describes AT commands related to Device Firmware Update (DFU) operations. These commands allow you to update the application firmware, delta modem firmware, full modem firmware, or the MCUboot second-stage bootloader through UART.

Note:

The DFU commands use data mode to receive firmware data. See the *Data mode* section for more information about data mode.

6.1 DFU types

The following DFU image types are supported:

- `0` - Application firmware (MCUboot)
- `1` - Delta modem firmware
- `2` - Full modem firmware
- `3` - MCUboot second-stage bootloader self-update. Only available when the device uses the NSIB (B0) and MCUboot as a second-stage bootloader. The host must stream the signed image for the **inactive** `s0`/`s1` slot.

Caution:

If the full modem firmware update fails after firmware data has been programmed to modem flash, the modem remains unusable until a successful update is completed.

6.2 DFU initialize #XDFUINIT

The `#XDFUINIT` command initializes the DFU target for a specific image type.

6.2.1 Set command

The set command initializes the DFU target.

Syntax

```
AT#XDFUINIT=<type>[,<size>]
```

The parameters and their defined values are the following:

<type>
Integer. The DFU image type as specified in the *DFU types* section.

<size>
Integer. The total firmware image size in bytes. Required for application (type 0), delta modem firmware (type 1), and MCUboot bootloader (type 3) updates.

For full modem firmware (type 2), the command triggers an immediate reboot into bootloader mode.

The #XDFUINIT, #XDFUWRITE, and #XDFUAPPLY commands use the same modem DFU area as the #XFOTA and #XNRFCLOUDFOTA commands. For delta modem firmware (type 1), they return ERROR in one of the following cases:

- A FOTA update started with the #XFOTA or #XNRFCLOUDFOTA command is in progress.
- Such an update is downloaded, but not yet activated.

When one of these commands succeeds, it reserves the modem DFU area for direct DFU until the modem is reset, for example with the #XMODEMRESET or #XRESET command. While the area is reserved, the #XFOTA and #XNRFCLOUDFOTA commands cannot start a modem firmware update or erase the modem DFU area.

Example

```
AT#XDFUINIT=0,123456
OK

AT#XDFUINIT=2
// device reboots into bootloader mode
Bootloader mode ready
```

6.2.2 Read command

The read command is not supported.

6.2.3 Test command

The test command tests the existence of the command and provides information about the type of its subparameters.

Syntax

```
AT#XDFUINIT=?
```

Response syntax

```
#XDFUINIT: <list of supported types>,<size>
```

Example

```
AT#XDFUINIT=?

#XDFUINIT: (0,1,2,3),<size>

OK
```

6.3 DFU write #XDFUWRITE

The #XDFUWRITE command writes firmware data to the DFU target. The command enters data mode to receive the firmware chunk.

6.3.1 Set command

The set command initiates a firmware data write operation.

Syntax

```
AT#XDFUWRITE=<type>,<addr>,<len>
```

The parameters and their defined values are the following:

<type>
Integer. The DFU image type as specified in the *DFU types* section.

<addr>
Integer. The address offset for the data.

<len>
Integer. The length of the data chunk to write.

After the command returns OK, the nRF91M1 module enters data mode to receive exactly <len> bytes of firmware data. When the data has been received and written, the nRF91M1 module sends a #XDFU unsolicited notification with the status. For delta modem firmware (type 1), the command returns ERROR under the same conditions as the #XDFUINIT command.

Example

```
AT#XDFUWRITE=0,0,4096
OK
// 4096 bytes of firmware data
#XDATAMODE: 0
#XDFU: 0,1,0
```

6.3.2 Read command

The read command is not supported.

6.3.3 Test command

The test command tests the existence of the command and provides information about the type of its subparameters.

Syntax

```
AT#XDFUWRITE=?
```

Response syntax

```
#XDFUWRITE: <list of supported types>,<addr>,<len>
```

Example

```
AT#XDFUWRITE=?

#XDFUWRITE: (0,1,2,3),<addr>,<len>

OK
```

6.4 DFU apply #XDFUAPPLY

The #XDFUAPPLY command finalizes the firmware update and schedules it for activation.

6.4.1 Set command

The set command applies the firmware update.

Syntax

```
AT#XDFUAPPLY=<type>
```

The parameters and their defined values are the following:

<type>
Integer. The DFU image type as specified in the *DFU types* section.

For application (type 0), delta modem firmware (type 1), and MCUboot bootloader (type 3), the update is scheduled and will be activated on the next reset, which can be done with the AT#XRESET command. For full modem firmware (type 2), the command applies the current segment (bootloader or firmware) and triggers a reboot if needed. For delta modem firmware (type 1), the command returns ERROR under the same conditions as the #XDFUINIT command.

Example

```
AT#XDFUAPPLY=0
OK
#XDFU: 0,2,0

AT#XRESET
OK
Ready
```

6.4.2 Read command

The read command is not supported.

6.4.3 Test command

The test command tests the existence of the command and provides information about the type of its subparameters.

Syntax

```
AT#XDFUAPPLY=?
```

Response syntax

```
#XDFUAPPLY: <list of supported types>
```

Example

```
AT#XDFUAPPLY=?

#XDFUAPPLY: (0,1,2,3)

OK
```

6.5 DFU unsolicited notification #XDFU

The nRF91M1 module sends the #XDFU unsolicited notification to report the status of DFU write and apply operations.

6.5.1 Unsolicited notification

```
#XDFU: <type>,<operation>,<status>
```

The parameters and their defined values are the following:

<type>
Integer. The DFU image type as specified in the *DFU types* section.

<operation>

- 1 - Data write completed.
- 2 - Apply update completed.

<status>

- 0 - Success.
- -1 - Failure.

6.6 Complete DFU examples

The following sections showcase a complete example of application, delta modem, and full modem firmware update.

6.6.1 Application firmware update

The following example shows a complete application firmware update:

```
// Note the current application version (major.minor.patch from app/
VERSION)
AT#XSMVER
#XSMVER: "v1.99.0-7-gdf00f46-dirty","3.3.0-rc1"
OK
```

(continues on next page)

(continued from previous page)

```
// Initialize DFU for application firmware (total size 123456 bytes)
AT#XDFUINIT=0,123456
OK

// Write first chunk (4096 bytes at offset 0)
AT#XDFUWRITE=0,0,4096
OK
// 4096 bytes of firmware data
#XDATAMODE: 0
#XDFU: 0,1,0

// Write second chunk (4096 bytes at offset 4096)
AT#XDFUWRITE=0,4096,4096
OK
// 4096 bytes of firmware data
#XDATAMODE: 0
#XDFU: 0,1,0

// ... continue writing chunks ...

// Apply the update
AT#XDFUAPPLY=0
OK
#XDFU: 0,2,0

// Reset to activate the new firmware
AT#XRESET
OK
Ready

// Verify the VERSION prefix increased (here 1.99.0 -> 1.99.1)
AT#XSMVER
#XSMVER: "v1.99.1-8-gf0fd0f46-dirty","3.3.0-rc1"
OK
```

6.6.2 Delta modem firmware update

The following example shows a complete delta modem firmware update:

```
// Note the current modem firmware
AT%SHORTSWVER
%SHORTSWVER: nrf91x1_2.0.0
OK

// Initialize DFU for delta modem firmware (total size 65536 bytes)
AT#XDFUINIT=1,65536
OK
```

(continues on next page)

(continued from previous page)

```
// Write first chunk (4096 bytes at offset 0)
AT#XDFUWRITE=1,0,4096
OK
// 4096 bytes of firmware data
#XDATAMODE: 0
#XDFU: 1,1,0

// Write second chunk (4096 bytes at offset 4096)
AT#XDFUWRITE=1,4096,4096
OK
// 4096 bytes of firmware data
#XDATAMODE: 0
#XDFU: 1,1,0

// ... continue writing chunks ...

// Apply the update
AT#XDFUAPPLY=1
OK
#XDFU: 1,2,0

// Reset the modem to activate the new firmware
AT#XMODEMRESET
#XMODEMRESET: 0
OK

// Verify that the modem is running the new firmware
AT%SHORTSWVER
%SHORTSWVER: nrf91x1_2.0.4
OK
```

6.6.3 Full modem firmware update

The following example shows a complete full modem firmware update.

Important:

If the update fails after the first 8 kB block of firmware data has been programmed, the modem remains unusable until a successful update is performed. The bootloader remains intact for retries. Ensure a reliable connection between the host and nRF91M1 module, and stable power supply before starting the update.

The full modem update consists of two phases:

1. Bootloader segment - Writes the bootloader data.
2. Firmware segment - Writes the firmware data.

Note:

Firmware segment data is buffered and programmed to modem flash in 8 kB blocks. A `#XDFU: 2,1,0` notification means the data was accepted, not that it was programmed. With contiguous writes of 8 kB or less, the first block is programmed during the write that brings the total amount of firmware data to 8 kB or more. If that write fails, the modem rejected the firmware, for example, because it is an older version. In that case, the existing modem firmware is left untouched, and you can return to normal mode with the `AT#XRESET` command. If a later write or `AT#XDFUAPPLY` fails, the modem remains unusable until a full modem firmware update is completed successfully.

```
// Note the current firmware version of the modem
AT%SHORTSWVER
%SHORTSWVER: nrf91x1_2.0.0
OK

// Initialize DFU for full modem firmware
AT#XDFUINIT=2
// device reboots
Bootloader mode ready

// Phase 1: Write bootloader segment
AT#XDFUWRITE=2,0,4096
OK
// 4096 bytes of bootloader data
#XDATAMODE: 0
#XDFU: 2,1,0

// ... continue writing bootloader chunks ...

// Apply bootloader segment (switches to firmware segment mode)
AT#XDFUAPPLY=2
OK
#XDFU: 2,2,0

// Phase 2: Write firmware segments
// Warning: Data is programmed to modem flash in 8 kB blocks. A
failure after the first block is programmed leaves the modem
unusable until the update is completed successfully.
AT#XDFUWRITE=2,0,4096
OK
// 4096 bytes of firmware data
#XDATAMODE: 0
#XDFU: 2,1,0

// ... continue writing firmware chunks ...

// Apply firmware segment (triggers reboot)
```

(continues on next page)

(continued from previous page)

```
AT#XDFUAPPLY=2
OK
#XDFU: 2,2,0
// device reboots
Ready

// Verify that the modem is running the new firmware
AT%SHORTSWVER
%SHORTSWVER: nrf91x1_2.0.4
OK
```

6.6.4 MCUboot firmware update

Important:

NSIB (B0) uses monotonic counter values to prevent rollback of the MCUboot bootloader. When performing an MCUboot DFU, the host must ensure that the new MCUboot image has a strictly higher monotonic counter value than the currently running MCUboot to avoid update failure.

Updates for MCUboot images are limited to 40 slots.

The following example shows a complete MCUboot firmware update:

```
// Query the current slot for MCUboot image
AT#XBOOTINFO=1
#XBOOTINFO: 0
OK

// Download the correct MCUboot image for the inactive slot.
// The default NCS image names for MCUboot are:
// - Slot 0: signed_by_mcuboot_and_b0_mcuboot.bin
// - Slot 1: signed_by_mcuboot_and_b0_s1_image.bin

// In this example, the device is currently running slot 0, so the
host should download the signed image for slot 1, which is signed_
by_mcuboot_and_b0_s1_image.bin.

// Note the current monotonic counter value for MCUboot
AT#XBOOTINFO=0
#XBOOTINFO: 1
OK

// Initialize DFU for MCUboot firmware (total size 23456 bytes)
AT#XDFUINIT=3,23456
OK
```

(continues on next page)

(continued from previous page)

```
// Write first chunk (4096 bytes at offset 0)
AT#XDFUWRITE=3,0,4096
OK
// 4096 bytes of firmware data
#XDATAMODE: 0
#XDFU: 3,1,0

// Write second chunk (4096 bytes at offset 4096)
AT#XDFUWRITE=3,4096,4096
OK
// 4096 bytes of firmware data
#XDATAMODE: 0
#XDFU: 3,1,0

// ... continue writing chunks ...

// Apply the update
AT#XDFUAPPLY=3
OK
#XDFU: 3,2,0

// Reset to activate the new firmware
AT#XRESET
OK
Ready

// Verify that the monotonic counter value for MCUboot was updated
AT#XBOOTINFO=0
#XBOOTINFO: 2
OK

// (Optional) Verify that the MCUboot slot was switched
AT#XBOOTINFO=1
#XBOOTINFO: 1
OK
```

6.7 DFU host example

See `app/scripts/sm_dfu_host.py` for an example of how to implement a DFU host. The script demonstrates the DFU AT command flow and can be used as a starting point for custom implementations.

Install dependencies:

```
pip install -r app/scripts/requirements.txt
```

Note:

cbor2 5.8.0 does not work with the `sm_dfu_host.py` script due to a known bug introduced in that version. Versions older and newer than that work. See cbor2 bug for more information.

6.7.1 Usage

```
python sm_dfu_host.py --port <serial_port> --baudrate <baud> --file
<path-to-firmware> --type <dfu_type>
```

Where `<dfu_type>` is one of: `application`, `modem-delta`, or `modem-full`.

After the DFU transfer completes:

Note:

Applying application and delta modem firmware can take some time.

- For application updates, reset the device to activate the new firmware:

  ```
  python sm_dfu_host.py --port <serial_port> --baudrate <baud> --
  reset
  ```

- For delta modem updates, reset the modem to apply the update:

  ```
  python sm_dfu_host.py --port <serial_port> --baudrate <baud> --
  modem-reset
  ```

- Full modem updates handle the reset automatically.

To check if the device is operating in normal mode:

```
python sm_dfu_host.py --port <serial_port> --baudrate <baud> --ping
```

Note:

Ping will return an error if the device is in bootloader mode.

7 FOTA AT commands

This page describes AT commands related to Firmware Over-The-Air (FOTA) updates.

7.1 FOTA request #XFOTA

The #XFOTA command sends various types of FOTA requests, based on specific operation codes.

7.1.1 Set command

The set command performs FOTA-related operations.

Syntax

```
AT#XFOTA=<op>[,<file_url>[,<sec_tag>[,<pdn_id>]]]
```

The parameters and their defined values are the following:

<op>

- 0 - Stop FOTA (during download only). Can be used as a way to pause and resume the download, by issuing the same FOTA start command to resume.
- 1 - Start FOTA for application update.
- 2 - Start FOTA for modem delta update.
- 5 - Start FOTA for MCUboot second-stage bootloader update.
- 7 - Read modem DFU area size and firmware image offset (for modem delta update).
- 9 - Erase modem DFU area (for modem delta update).

<file_url>
String. The full HTTP or HTTPS path of the target image to download. Must be provided to the start operations. For the MCUboot bootloader update, the target image must be the signed image for the **inactive** s0/s1 slot. You can use the AT#XBOOTINFO=1 command to query which slot is active.

<sec_tag>
Integer. Indicates to the modem the credential of the security tag used for establishing a secure connection for downloading the image. It is associated with the certificate or PSK. Must be provided to the start operations when using HTTPS.

<pdn_id>
Integer. The Packet Data Network (PDN) ID that can be used instead of the default PDN for downloading. Can be provided to the start operations.

> **Note:**
>
> For historical reasons, this is the PDN ID and not the `cid` of the `+CGDCONT` command, which most other commands use. You can get the PDN ID of a `cid` with the `AT%XGETPDNID=<cid>` command.

> **Note:**
>
> Raw sockets must not use the PDN connection at the same time. See *Socket AT commands* for more information.

After the `#XFOTA: 4,0` notification indicating that the FOTA update is downloaded, it must be activated by resetting either the whole device with the `AT#XRESET` command, or in case of modem update only the modem with the `AT#XMODEMRESET` command.

> **Note:**
>
> During a delta update to the modem, the modem DFU area is automatically erased as needed as part of the update process. Erasing this area manually is optional.
>
> However, this leads to the command starting the update taking longer to complete, and also leaves the connection to the FOTA server idle while the area is being erased, which can provoke issues.

> **Note:**
>
> The firmware image is stored in external flash memory during a full modem update. The external flash is erased automatically after a new firmware activation.
>
> Activating the new modem firmware is done identically to a modem delta update, by resetting either the whole device or only the modem.

> **Important:**
>
> For the MCUboot bootloader update, NSIB (B0) uses monotonic counter values to prevent rollback. When performing an MCUboot FOTA, the host must ensure that the new MCUboot image has a strictly higher monotonic counter value than the currently running MCUboot to avoid update failure.
>
> Updates for MCUboot images are limited to 40 slots.

Response syntax

```
AT#XFOTA=7
#XFOTA: <size>,<offset>
```

The parameters and their defined values are the following:

<size>
Integer. The size of the modem DFU area in bytes.

<offset>
Integer. The offset of the firmware image in the modem DFU area. It is `0` if no image is in the modem DFU area, and `2621440` (`NRF_MODEM_DELTA_DFU_OFFSET_DIRTY`) if the modem DFU area needs to be erased before a new firmware update can be received.

Example

```
application firmware download and update
AT#XFOTA=1,"http://remote.host/fota/sm_app_update.bin"
OK
#XFOTA: 1,0,0
...
#XFOTA: 4,0
AT#XRESET
OK
Ready
#XFOTA: 5,0

read modem DFU info
AT#XFOTA=7
#XFOTA: 294912,0
OK

modem firmware download and update
AT#XFOTA=2,"http://remote.host/fota/mfw_nrf9151_update_from_2.0.2_
to_2.0.3.bin"
OK
#XFOTA: 1,0,0
...
#XFOTA: 4,0
AT#XMODEMRESET
#XFOTA: 5,0
#XMODEMRESET: 0
OK

read modem DFU info
AT#XFOTA=7
#XFOTA: 294912,2621440
```

(continues on next page)

(continued from previous page)

```
OK

erase modem DFU area for next modem delta update (optional)
AT#XFOTA=9
OK

MCUboot bootloader update
AT#XBOOTINFO=1
#XBOOTINFO: 0
OK
AT#XFOTA=5,"https://remote.host/fota/signed_by_mcuboot_and_b0_s1_
image.bin",42
OK
#XFOTA: 1,0,0
...
#XFOTA: 4,0
AT#XRESET
OK
Ready
#XFOTA: 5,0
```

Unsolicited notification

```
#XFOTA: <fota_stage>,<fota_status>[,<fota_info>]
```

The parameters and their defined values are the following:

<fota_stage>

- `0` - Init.
- `1` - Download.
- `2` - Download, erase pending (modem delta update only).
- `3` - Download, erase complete (modem delta update only).
- `4` - Downloaded, to be activated.
- `5` - Complete.

<fota_status>

- `0` - OK.
- `1` - Error.
- `2` - Cancelled.
- `3` - Reverted (application FOTA only).

<fota_info>

Integer. Its value can have different meanings based on the values returned by `<fota_stage>` and `<fota_status>`.

See the following table:

`<fota_stage>`	`<fota_status>`	`<fota_info>`
1 (namely *Download*)	`0` (namely *OK*)	Percentage of the download
1 (namely *Download*)	1 (namely *ERROR*)	Error Code
1 (namely *Download*)	2 (namely *CANCELLED*)	`0` - Downloading is cancelled before completion
5 (namely *Complete*)	1 (namely *ERROR*)	Error Code

The error codes can be the following:

- 1 - Download failed
- 2 - Update image rejected (for example modem firmware version error)
- 3 - Update image mismatch (for example `<op>` is 1 but `<file_url>` points to a modem image)

For modem FOTA, the error codes can be the following:

- `71303169` (`NRF_MODEM_DFU_RESULT_INTERNAL_ERROR`) - The modem encountered a fatal internal error during firmware update.
- `71303170` (`NRF_MODEM_DFU_RESULT_HARDWARE_ERROR`) - The modem encountered a fatal hardware error during firmware update.
- `71303171` (`NRF_MODEM_DFU_RESULT_AUTH_ERROR`) - Modem firmware update failed due to an authentication error.
- `71303172` (`NRF_MODEM_DFU_RESULT_UUID_ERROR`) - Modem firmware update failed due to UUID mismatch.
- `71303173` (`NRF_MODEM_DFU_RESULT_VOLTAGE_LOW`) - Modem firmware update not executed due to low voltage. The modem will retry the update on reboot.

7.1.2 Read command

The read command is not supported.

7.1.3 Test command

The test command tests the existence of the command and provides information about the type of its subparameters.

Syntax

```
AT#XFOTA=?
```

Response syntax

```
#XFOTA: <list of op value>,<file_url>,<sec_tag>,<pdn_id>
```

Examples

```
AT#XFOTA=?

#XFOTA: (0,1,2,3,5,7,9),<file_url>,<sec_tag>,<pdn_id>

OK
```

8 GNSS AT commands

This page describes GNSS-related AT commands.

8.1 Control GNSS #XGNSS

The #XGNSS command controls the GNSS.

8.1.1 Set command

The set command allows you to start and stop the GNSS module.

The nRF Cloud A-GNSS service can be used with the module to reduce the time it takes for the GNSS module to estimate device position.

Syntax

```
AT#XGNSS=<op>,<cloud_assistance>,<interval>[,<timeout>]
```

The parameters and their defined values are the following:

<op>

- `0` - Stop the GNSS module. In this case, no other parameter is allowed.
- `1` - Start the GNSS module.

<cloud_assistance>

- `0` - Disabled.
- `1` - Enabled.

<interval>

Integer. The GNSS fix interval in seconds. It can have one of the following values:

- `0` - Single-fix navigation mode.
- `1` - Continuous navigation mode. The fix interval is set to 1 second.
- Ranging from `10` to `65535` - Periodic navigation mode. The fix interval is set to the specified value.

<timeout>

Integer. The maximum time in seconds that the GNSS receiver is allowed to run while

trying to produce a valid Position, Velocity, and Time (PVT) estimate. Can only be specified in single-fix and periodic navigation modes, and must be omitted in continuous navigation mode. If omitted in single-fix or periodic navigation modes, the default value is 60 seconds.

- `0` - The GNSS receiver runs indefinitely until a valid PVT estimate is produced.
- Any positive integer - The GNSS receiver is turned off after the specified time is up, even if a valid PVT estimate was not produced.

In periodic navigation mode, the `<interval>` and `<timeout>` parameters are temporarily ignored during the first fix.

Note:

When `<cloud_assistance>` is disabled, no request is made to nRF Cloud for assistance data. However, if it has been previously enabled and used, such data may remain locally available and will be used if still valid.

Note:

Make sure that the GNSS antenna is configured properly. You may need to issue the `AT%XMAGPIO` and `AT%XCOEX0` AT commands to configure the Low-Noise Amplifier (LNA), depending on the hardware (HW).

If you want to use the nRF9151 DK with an onboard antenna, use the `AT%XCOEX0=1,1,1565,1586` command.

Tip:

When the LTE link is enabled, make sure to have either Power Saving Mode (PSM) or extended Discontinuous Reception (eDRX) enabled to give the GNSS receiver the time it needs to acquire fixes.

As an alternative to GNSS-based positioning, see *#XNRFCLOUDPOS* for cellular and Wi-Fi® positioning.

Unsolicited notification

`#XGNSSPOS` notification indicates that a new position fix has been acquired.

```
#XGNSSPOS: <latitude>,<longitude>,<altitude>,<accuracy>,<speed>,
<heading>,<datetime>
```

The parameters and their defined values are the following:

<latitude>
Float. The latitude in degrees.

<longitude>
Float. The longitude in degrees.

<altitude>
Float. The altitude above the WGS-84 ellipsoid in meters.

<accuracy>
Float. The accuracy (2D 1-sigma) in meters.

<speed>
Float. The horizontal speed in meters per second.

<heading>
Float. The heading of the movement of the user in degrees.

<datetime>
String. The UTC date-time.

`#XGNSS` notification indicates GNSS status.

```
#XGNSS: <gnss_service>,<gnss_status>
```

The parameters and their defined values are the following:

<gnss_service>
Integer. When it has the value `1`, it means that GNSS is supported in `%XSYSTEMMODE` and activated in `+CFUN`.

<gnss_status>

- `0` - GNSS is stopped.
- `1` - GNSS is started.
- `2` - GNSS wakes up in periodic mode.
- `3` - GNSS enters sleep because of timeout.
- `4` - GNSS enters sleep because a fix is acquired.

Example

Get GNSS fix in GNSS only mode:

```
AT%XSYSTEMMODE=0,0,1,0

OK
AT+CFUN=31

OK
AT#XGNSS=1,0,0,0

#XGNSS: 1,1

OK
```

(continues on next page)

(continued from previous page)

```
#XGNSSPOS: 35.457576,139.625090,121.473785,22.199919,0.442868,0.
000000,"2021-06-02 06:25:48"

#XGNSS: 1,4
```

Set PSM and eDRX configurations, use GNSS assistance from nRF Cloud and get GNSS fixes in continuous navigation mode:

```
AT%XSYSTEMMODE=1,0,1,0

OK
AT+CPSMS=1,,,"00000001","00000011"

OK
AT+CEDRXS=2,4,"0011"

OK
AT+CFUN=1

OK

+CEDRXP: 4,"0011","0011","0011"

AT#XNRFCLOUD=1

OK
#XNRFCLOUD: 1,0
AT#XGNSS=1,1,1

#XGNSS: 1,1

OK

#XGNSSPOS: 35.457417,139.625211,162.850952,15.621976,1.418092,0.
000000,"2021-06-02 05:21:31"

#XGNSSPOS: 35.457435,139.625348,176.104797,14.245458,1.598184,69.
148659,"2021-06-02 05:21:32"

#XGNSSPOS: 35.457417,139.625415,179.132980,13.318132,1.235241,69.
148659,"2021-06-02 05:21:33"

#XGNSSPOS: 35.457410,139.625469,181.223541,12.667312,0.803951,69.
148659,"2021-06-02 05:21:34"
AT#XGNSS=0
OK
```

(continues on next page)

(continued from previous page)

```
#XGNSS: 1,0
```

Set PSM and eDRX configurations, use GNSS assistance from nRF Cloud and get GNSS fixes in periodic navigation mode every 10 seconds:

```
AT%XSYSTEMMODE=1,0,1,0

OK
AT+CPSMS=1,,,"00000001","00000011"

OK
AT+CEDRXS=2,4,"0011"

OK
AT+CFUN=1

OK
AT#XNRFCLOUD=1

OK
#XNRFCLOUD: 1,0
AT#XGNSS=1,1,10

#XGNSS: 1,1

OK

#XGNSS: 1,2

#XGNSSPOS: 35.457243,139.625435,149.005020,28.184258,10.431827,281.
446014,"2021-06-24 04:35:52"

#XGNSS: 1,4

#XGNSS: 1,2

#XGNSSPOS: 35.457189,139.625602,176.811203,43.015198,0.601837,281.
446014,"2021-06-24 04:36:32"

#XGNSS: 1,4

#XGNSS: 1,2

#XGNSSPOS: 35.457498,139.625422,168.243591,31.753956,0.191195,281.
446014,"2021-06-24 04:37:12"
```

(continues on next page)

(continued from previous page)

```
#XGNSS: 1,4

#XGNSS: 1,2

#XGNSSPOS: 35.457524,139.624667,100.745979,25.324850,6.347160,94.
699837,"2021-06-24 04:37:52"

#XGNSS: 1,4
AT#XGNSS=0

#XGNSS: 1,0

OK
```

8.1.2 Read command

The read command allows you to check GNSS support and its status.

Syntax

```
AT#XGNSS?
```

Response syntax

```
#XGNSS: <gnss_service>,<gnss_status>
```

The parameters and their defined values are the following:

<gnss_service>
Integer. When it has the value 1, it means that GNSS is supported in %XSYSTEMMODE and activated in +CFUN.

<gnss_status>

- 0 - GNSS is stopped.
- 1 - GNSS is started.
- 2 - GNSS wakes up in periodic mode.
- 3 - GNSS enters sleep because of timeout.
- 4 - GNSS enters sleep because a fix is acquired.

Example

```
AT#XGNSS?

#XGNSS: 1,1

OK
```

8.1.3 Test command

The test command tests the existence of the command and provides information about the type of its subparameters.

Syntax

```
AT#XGNSS=?
```

Example

```
AT#XGNSS=?

#XGNSS: (0,1),(0,1),<interval>,<timeout>

OK
```

8.2 Delete GNSS data #XGNSSDEL

The #XGNSSDEL command deletes GNSS data from non-volatile memory. This command should be issued when GNSS is activated but not started yet.

Note:

This is a debug feature, and is not supposed to be used in production code.

8.2.1 Set command

The set command allows you to delete cached GNSS data. Using this command does not trigger A-GNSS nor P-GPS data request events. The execution of this command may delay the full functionality of A-GNSS and P-GPS until the next periodic data request has been received.

Syntax

```
AT#XGNSSDEL=<mask>
```

The parameters and their defined values are the following:

<mask>
Integer. A bitwise OR of the following bitmasks:

- `0x001` - Ephemerides.
- `0x002` - Almanacs (excluding leap second and ionospheric correction).
- `0x004` - Ionospheric correction parameters.
- `0x008` - Last good fix (the last position).
- `0x010` - GPS time-of-week (TOW).
- `0x020` - GPS week number.
- `0x040` - Leap second (UTC parameters).
- `0x100` - Precision estimate of GPS time-of-week (TOW).
- `0x400` - Extended Kalman Filter (EKF) state (last position/velocity/time solution).

For example, the integer value with all of the above bits enabled will be `1407` (`0x57f`).

Example

```
AT%XSYSTEMMODE=0,0,1,0
OK
AT+CFUN=31
OK
AT#XGNSSDEL=1407
OK
AT+CFUN=0
OK
```

8.2.2 Read command

The read command is not supported.

8.2.3 Test command

The test command tests the existence of the command and provides information about the type of its subparameters.

Syntax

```
AT#XGNSSDEL=?
```

Example

```
AT#XGNSSDEL=?

#XGNSSDEL: <mask>

OK
```

9 HTTP client AT commands

This page describes AT commands for the HTTP client. The HTTP client operates on sockets managed by the *Socket AT commands*. You can perform the following using the Socket AT commands:

- Create a socket with `AT#XSOCKET` or `AT#XSSOCKET`.
- Connect it with `AT#XCONNECT`.
- Set socket options with `AT#XSOCKETOPT` or `AT#XSSOCKETOPT`.
- Close with `AT#XCLOSE`.

Request idle timeout

Every active HTTP request has a sliding idle timeout of 30 seconds.

The timer resets each time data is sent or received:

- Sending request headers or body upload chunks (POST/PUT data mode).
- Receiving response headers or body bytes.
- Pulling a body chunk in manual mode (`AT#XHTTPCDATA`).

If no such activity occurs within the configured window, the request is aborted and `#XHTTPCSTAT: <handle>,-1,<total_bytes>,<connection_close>` is emitted. The timeout is enforced by a background timer that fires independently of normal socket poll events, so a server that stalls silently (no TCP RST or FIN) is also detected.

9.1 HTTP request #XHTTPCREQ

The `#XHTTPCREQ` command starts an asynchronous HTTP or HTTPS request on a socket connected with `AT#XCONNECT`.

9.1.1 Set command

The set command starts an HTTP or HTTPS request.

Syntax

```
AT#XHTTPCREQ=<handle>,<url>,<method>[,<auto_reception>[,<format>[,
<body_len>[,<header 1>[,<header 2>[...]]]]]]
```

The parameters and their defined values are the following:

<handle>
Integer. Identifies the connected socket handle returned by AT#XSOCKET or AT#XSSOCKET, and connected by AT#XCONNECT.

<url>
String. The full URL of the request, for example, `http://host/path` or `https://host/path`.

<method>

- 0 - GET.
- 1 - POST.
- 2 - PUT.
- 3 - DELETE.
- 4 - HEAD.

<auto_reception>
Integer. Optional. When omitted, automatic reception is used.

- 1 - Automatic mode (default). Response body bytes are received and forwarded to the host automatically.
- 0 - Manual mode. Response body reception is paused after the headers are parsed and #XHTTPCHEAD is emitted. The host must then retrieve body data in chunks using AT#XHTTPCDATA commands.

<format>
Integer. Optional. Controls the encoding used to deliver response body bytes to the host.

- 0 - Binary (default). Response body bytes are forwarded to the host as raw binary.
- 1 - Hex string. Response body bytes are encoded as a lower-case ASCII hex string before delivery. Each raw byte becomes two hex characters. The <length> field in #XHTTPCDATA always reports the raw byte count. The actual data delivered is 2×<length> ASCII hex characters. Applies to both automatic and manual reception modes.

<body_len>
Integer. Optional. Required as a placeholder when <header> parameters follow.

- 0 - No request body.
- Positive integer - Length of the request body in bytes. Valid for POST and PUT only. When set, the HTTP request headers are sent to the server immediately when the AT command is processed. The command then responds with OK and enters data mode. Body bytes are forwarded to the server as they are received from the host in data mode. The host must send exactly this many bytes as the request body. Data mode exits and #XDATAMODE: 0 is reported when all bytes have been processed.

<header X>
String. Optional. An additional HTTP request header. Empty parameters between headers (<header 1>, <header 2>,, <header 3>, ...) are ignored.

Note:

The HTTP client uses HTTP/1.1 and relies on the protocol default of persistent (keep-alive) connections. Multiple sequential requests can therefore be made on the same connected socket without reconnecting. To close the connection after a response, pass "Connection: close" as a custom header.

Response syntax

```
#XHTTPCREQ: <handle>
OK
```

The parameters and their defined values are the following:

<handle>
Integer. Identifies the socket.

Unsolicited notification

#XHTTPCHEAD is emitted when response headers have been parsed:

```
#XHTTPCHEAD: <handle>,<status_code>,<content_length>
```

The parameters and their defined values are the following:

<handle>
Integer. Identifies the socket.

<status_code>
Integer. The HTTP status code returned by the server.

<content_length>
Integer. The value of the Content-Length response header, or -1 when the server uses chunked transfer encoding or does not provide a content length.

#XHTTPCDATA is emitted in automatic mode for each received body chunk:

```
#XHTTPCDATA: <handle>,<offset>,<length>
<data>
```

The notification line is terminated with \r\n, the response data follows immediately, and a \r\n terminator follows the data bytes. In binary mode (<format>=0, default), the data is <length> raw bytes. In hex mode (<format>=1), the data is 2×<length> lower-case ASCII hex characters.

The parameters and their defined values are the following:

<handle>
Integer. Identifies the socket.

<offset>
Integer. The number of body bytes already delivered before this chunk.

<length>
Integer. The number of body bytes in this chunk.

Note:

When the server uses `Transfer-Encoding: chunked`, the body bytes delivered via #XHTTPCDATA include the raw chunked framing. Each chunk consists of a size line (hexadecimal length followed by `\r\n`), the chunk data, and a trailing `\r\n`. The final zero-length chunk `0\r\n\r\n` is also forwarded. The host strips this framing to recover the original body content. The `<total_bytes>` field in #XHTTPCSTAT reflects the raw wire byte count, including chunked framing, not the decoded body length.

#XHTTPCSTAT is emitted when the request completes, fails, or is cancelled:

```
#XHTTPCSTAT: <handle>,<status_code>,<total_bytes>,<connection_close>
```

The parameters and their defined values are the following:

<handle>
Integer. Identifies the socket.

<status_code>
Integer. The HTTP status code on success, or `-1` on failure, cancel, or timeout.

<total_bytes>
Integer. On successful completion, failure, or timeout, it contains the total number of response body bytes received by the HTTP client. For chunked transfer encoding this includes the raw framing bytes (chunk-size lines, `\r\n` separators, and the final `0\r\n\r\n` terminator). On cancel (`status_code=-1` from AT#XHTTPCCANCEL or AT#XCLOSE), or timeout, it contains the number of response body bytes already delivered to the host.

<connection_close>

- `0` - Keep-alive connection.
- `1` - The server includes a `Connection: close` header, indicating that the TCP connection will be closed after this response. The host must close the socket with AT#XCLOSE and open a new connection before issuing the next request.

Note:

For responses that carry no body by definition — HEAD requests, `1xx`, `204 No Content`, and `304 Not Modified` — #XHTTPCHEAD is emitted immediately followed by #XHTTPCSTAT. No #XHTTPCDATA notification is generated.

Examples

The following examples demonstrate how to perform an HTTP GET request and receive the response in automatic or manual mode on a connected socket.

HTTP GET (automatic mode):

```
AT#XHTTPCREQ=0,<url>,0
#XHTTPCREQ: 0
OK

#XHTTPCHEAD: 0,200,261

#XHTTPCDATA: 0,0,261
<261 bytes>

#XHTTPCSTAT: 0,200,261,0
```

Note:

In automatic mode, a `\r\n` terminator is appended after the data bytes of each #XHTTPCDATA notification.

HTTP GET (manual mode):

```
AT#XHTTPCREQ=0,<url>,0,0
#XHTTPCREQ: 0
OK

#XHTTPCHEAD: 0,200,261

AT#XHTTPCDATA=0
#XHTTPCDATA: 0,0,261
<261 bytes>
OK

#XHTTPCSTAT: 0,200,261,0
```

HTTP GET with hex-encoded response body (`format=1`, automatic mode):

```
AT#XHTTPCREQ=0,<url>,0,1,1
#XHTTPCREQ: 0
OK

#XHTTPCHEAD: 0,200,12

#XHTTPCDATA: 0,0,12
48656c6c6f20576f726c6421
```

(continues on next page)

(continued from previous page)

```
#XHTTPCSTAT: 0,200,12,0
```

The hex string and the line break after it represent the `2×<length>` hex characters followed by the `\r\n` terminator.

HTTP POST with JSON body and custom header:

```
AT#XHTTPCREQ=0,<url>,1,1,0,15,"Content-Type: application/json"
#XHTTPCREQ: 0
OK
{"key":"value"}
#XDATAMODE: 0

#XHTTPCHEAD: 0,200,432

#XHTTPCDATA: 0,0,432
<432 bytes>

#XHTTPCSTAT: 0,200,432,0
```

HTTP GET data in binary format with Range header (`body_len=0` is required as a placeholder when `<header>` parameters follow):

In this example the server returns `connection_close=1`, so the host must close and reopen the socket before the next request.

```
AT#XHTTPCREQ=0,<url>,0,1,0,0,"Range: bytes=0-127"
#XHTTPCREQ: 0
OK

#XHTTPCHEAD: 0,206,128

#XHTTPCDATA: 0,0,128
<128 bytes>

#XHTTPCSTAT: 0,206,128,0

AT#XHTTPCREQ=0,<url>,0,1,0,0,"Range: bytes=128-255"
#XHTTPCREQ: 0
OK

#XHTTPCHEAD: 0,206,128

#XHTTPCDATA: 0,0,128
<128 bytes>

#XHTTPCSTAT: 0,206,128,1
```

(continues on next page)

(continued from previous page)

```
AT#XCLOSE=0
#XCLOSE: 0,0
OK
```

HTTP HEAD (no body — #XHTTPCSTAT follows immediately after #XHTTPCHEAD):

```
AT#XHTTPCREQ=0,<url>,4
#XHTTPCREQ: 0
OK

#XHTTPCHEAD: 0,200,261

#XHTTPCSTAT: 0,200,0,0
```

HTTP POST with chunked response (`content_length=-1`):

```
AT#XHTTPCREQ=0,<url>,1,1,0,1024
#XHTTPCREQ: 0
OK
<1024 bytes payload>
#XDATAMODE: 0

#XHTTPCHEAD: 0,200,-1

#XHTTPCDATA: 0,0,1132
460\r\n{"args":{},"data":"<1024 bytes>","url":"..."}\r\n0\r\n\r\n

#XHTTPCSTAT: 0,200,1132,0
```

Note:

The 1132 raw bytes break down as chunked framing: `460\r\n` (chunk-size 1120 decimal in hex, 5 bytes), 1120 bytes of JSON body, `\r\n` (chunk trailer, 2 bytes), and `0\r\n\r\n` (final zero-length chunk, 5 bytes). Strip this framing to recover the 1120-byte JSON body.

9.1.2 Test command

The test command tests the existence of the command and provides information about the type of its subparameters.

Syntax

```
AT#XHTTPCREQ=?
```

Response syntax

```
#XHTTPCREQ: <handle>,<url>,<method>[,<auto_reception>[,<format>[,
<body_len>[,<header>]...]]]
```

Example

```
AT#XHTTPCREQ=?
#XHTTPCREQ: <handle>,<url>,<method>[,<auto_reception>[,<format>[,
<body_len>[,<header>]...]]]
OK
```

9.2 HTTP data pull #XHTTPCDATA

The #XHTTPCDATA command pulls the next body chunk for a manual-mode HTTP request.

9.2.1 Set command

The set command performs one non-blocking receive on the socket of a manual-mode request and forwards the available data to the host.

Syntax

```
AT#XHTTPCDATA=<handle>[,<length>]
```

The parameters and their defined values are the following:

<handle>
Integer. Identifies the socket used when starting the manual-mode request.

<length>
Integer. Optional. The maximum number of bytes to receive in this pull. When omitted, the full internal receive buffer size is used.

Response syntax

When body data is available:

```
#XHTTPCDATA: <handle>,<offset>,<length>
<length bytes of raw body data>
OK
```

The `#XHTTPCDATA:` line is terminated with `\r\n`. `<length>` raw body bytes follow immediately with no additional separator. `OK` follows on its own line after the body bytes.

When the socket buffer is temporarily empty (EAGAIN):

```
#XHTTPCDATA: <handle>,<offset>,0
OK
```

The parameters and their defined values are the following:

<handle>
Integer. Identifies the socket.

<offset>
Integer. The number of body bytes already delivered before this pull.

<length>
Integer. The number of body bytes delivered in this pull. A value of `0` means the socket buffer is currently empty.

When all body bytes have been delivered, `#XHTTPCSTAT` is sent as a URC after the final `OK`. This happens either when the server closes the connection, when the `Content-Length` bytes have all been forwarded, or when the chunked transfer `0\r\n\r\n` terminator is received.

Note:

When the server uses `Transfer-Encoding: chunked` (`content_length=-1` in `#XHTTPCHEAD`), the bytes returned by `#XHTTPCDATA` include raw chunked framing. See the note under `#XHTTPCDATA` in the `AT#XHTTPCREQ` section.

Note:

Retry the pull after a brief delay when `#XHTTPCDATA: <handle>,<offset>,0` is received. Use `AT#XHTTPCCANCEL=<handle>` to cancel if no more data is needed.

Example

```
AT#XHTTPCDATA=0,256
#XHTTPCDATA: 0,0,256
<256 bytes>
OK

AT#XHTTPCDATA=0,256
#XHTTPCDATA: 0,256,256
<256 bytes>
OK

#XHTTPCSTAT: 0,200,512,0
```

9.2.2 Test command

The test command tests the existence of the command and provides information about the type of its subparameters.

Syntax

```
AT#XHTTPCDATA=?
```

Response syntax

```
#XHTTPCDATA: <handle>[,<length>]
```

Example

```
AT#XHTTPCDATA=?
#XHTTPCDATA: <handle>[,<length>]
OK
```

9.3 HTTP request cancel #XHTTPCCANCEL

The #XHTTPCCANCEL command cancels an active HTTP request.

9.3.1 Set command

The set command cancels an active request.

Syntax

```
AT#XHTTPCCANCEL=<handle>
```

The parameters and their defined values are the following:

<handle>
Integer. Identifies the socket of the request to cancel.

An unsolicited `#XHTTPCSTAT: <handle>,-1,<total_bytes>,<connection_close>` notification is emitted after cancellation, where `<total_bytes>` is the number of response body bytes already delivered to the host.

Note:

Closing the socket with AT#XCLOSE while a request is active also cancels the request and emits `#XHTTPCSTAT: <handle>,-1,<total_bytes>,<connection_close>` before the socket is freed.

Example

```
AT#XHTTPCCANCEL=0
OK
#XHTTPCSTAT: 0,-1,0,0
```

9.3.2 Test command

The test command tests the existence of the command and provides information about the type of its subparameters.

Syntax

```
AT#XHTTPCCANCEL=?
```

Response syntax

```
#XHTTPCCANCEL: <handle>
```

Example

```
AT#XHTTPCCANCEL=?
#XHTTPCCANCEL: <handle>
OK
```

10 ICMP AT commands

This page describes ICMP-related AT commands.

10.1 ICMP echo request #XPING

The `#XPING` command sends an ICMP Echo Request, also known as *Ping*.

Only one ping request can be active at a time.

10.1.1 Set command

The set command allows you to send an ICMP Echo Request.

Syntax

```
AT#XPING=<url>,<length>,<timeout>[,<count>[,<interval>[,<pdn>]]]
```

The parameters and their defined values are the following:

<url>
: String. The hostname, the IPv4, or the IPv6 address of the target host.

<length>
: Integer. The length of the ICMP payload, in bytes. The value range is `0` to `1472` for IPv4 and `0` to `1452` for IPv6. These limits come from the MTU of 1500 bytes minus the IP and ICMP headers.

<timeout>
: Integer. The time to wait for each reply, in milliseconds. The value range is `0` to `4294967295`.

<count>
: Integer. The number of echo requests to send. The default value is `1`. The value range is `1` to `65535`.

<interval>
: Integer. The time to wait for sending the next echo request, in milliseconds. The default value is `1000`. The value range is `0` to `4294967295`.

<pdn>
: Integer. Represents `cid` in the `+CGDCONT` command. The default value is `0`.

Note:

Other sockets cannot use the same PDN connection. See *Socket AT commands* for more information.

Unsolicited notification

```
#XPING: <response time> seconds
```

The parameters and their defined values are the following:

<response time>
Float. The elapsed time, in seconds, between the echo requests and the echo replies.

Example

```
AT#XPING="5.189.130.26",45,5000,5,1000
OK
#XPING: 0.386 seconds
#XPING: 0.341 seconds
#XPING: 0.353 seconds
#XPING: 0.313 seconds
#XPING: 0.313 seconds
#XPING: average 0.341 seconds
AT#XGETADDRINFO="ipv6.google.com"
#XGETADDRINFO: "2404:6800:4006:80e::200e"
OK
AT#XPING="ipv6.google.com",45,5000,5,1000
OK
#XPING: 0.286 seconds
#XPING: 0.077 seconds
#XPING: 0.110 seconds
#XPING: 0.037 seconds
#XPING: 0.106 seconds
#XPING: average 0.123 seconds
AT#XPING="5.189.130.26",45,5000,5,1000,1
OK
#XPING: 1.612 seconds
#XPING: 0.349 seconds
#XPING: 0.334 seconds
#XPING: 0.278 seconds
#XPING: 0.278 seconds
#XPING: average 0.570 seconds
```

10.1.2 Read command

The read command is not supported.

10.1.3 Test command

The test command is not supported.

11 MQTT client AT commands

This page describes the AT commands used to operate the MQTT client.

Note:

The MQTT client holds no internal state information for publish and subscript. It will not retransmit packets or guarantee packet delivery according to QoS classes. This deviates from MQTT v3.1.1.

11.1 MQTT event notification #XMQTTEVT

MQTT client AT commands report protocol events as unsolicited notifications:

11.1.1 Unsolicited notification

```
#XMQTTEVT: <evt_type>,<result>
```

The parameters and their defined values are the following:

<evt_type>
Integer. The MQTT event type:

- `0` - Connection acknowledgment (CONNACK).
- `1` - Disconnection (DISCONNECT). The MQTT client is disconnected from the MQTT broker when the nRF91M1 module sends this event. If the host receives this event without a preceding `AT#XMQTTCON=0` command, the host must issue `AT#XMQTTCON=0` before connecting again.
- 2 - Incoming publish (PUBLISH). A subscribed topic received a message. The nRF91M1 module delivers the payload in a separate `#XMQTTMSG` notification. See *Incoming message notification #XMQTTMSG*.
- 3 - Publish acknowledgment (PUBACK), QoS 1.
- 4 - Publish received (PUBREC), QoS 2.
- 5 - Publish release (PUBREL), QoS 2.
- 6 - Publish complete (PUBCOMP), QoS 2.
- 7 - Subscribe acknowledgment (SUBACK).

- 8 - Unsubscribe acknowledgment (UNSUBACK).
- 9 - Ping response (PINGRESP). The nRF91M1 module can send this event at any time while connected, when the keep alive interval expires and the client pings the broker.

<result>

- 0 - Success.
- *Negative value* - Error code indicating the reason for the failure.

The commands describe which <evt_type> values each command can trigger. All notifications use the syntax documented in this section.

11.2 MQTT configure #XMQTTCFG

The #XMQTTCFG command allows you to configure the MQTT client before connecting to a broker.

11.2.1 Set command

The set command allows you to configure the MQTT client.

Syntax

```
AT#XMQTTCFG=<client_id>[,<keep_alive>[,<clean_session>]]
```

The parameters and their defined values are the following:

<client_id>
String. The MQTT Client ID. If this command is not issued, nRF91M1 module uses the default value of sm_default_client_id.

<keep_alive>
Integer. The maximum Keep Alive time in seconds for MQTT. The default Keep Alive time is 60 seconds.

<clean_session>

- 0 - Connect to a MQTT broker using a persistent session.
- 1 - Connect to a MQTT broker using a clean session.

The default is using a persistent session.

Examples

```
AT#XMQTTCFG="MyMQTT-Client-ID",300,1
OK
```

11.2.2 Read command

The read command shows MQTT client configuration information.

Syntax

```
AT#XMQTTCFG?
```

Response syntax

```
#XMQTTCFG: <client_id>,<keep_alive>,<clean_session>
```

The parameters and their defined values are the following:

<client_id>
String. The MQTT Client ID.

<keep_alive>
Integer. The maximum Keep Alive time in seconds for MQTT.

<clean_session>

- `0` - Connect to a MQTT broker using a persistent session.
- `1` - Connect to a MQTT broker using a clean session.

Examples

```
AT#XMQTTCFG?
#XMQTTCFG: "MyMQTT-Client-ID",60,0
OK
```

11.2.3 Test command

The test command tests the existence of the command and provides information about the type of its subparameters.

Syntax

```
AT#XMQTTCFG=?
```

Response syntax

```
#XMQTTCFG: <client_id>,<keep_alive>,<clean_session>
```

Examples

```
AT#XMQTTCFG=?
#XMQTTCFG: <client_id>,,<keep_alive>,<clean_session>
OK
```

11.3 MQTT connect #XMQTTCON

The `#XMQTTCON` command allows you to connect to and disconnect from the MQTT broker.

This command can send `#XMQTTEVT` with the following `<evt_type>` values:

- `0` - After a successful connect request (`<op>` `1` or `2`).
- `1` - After a disconnect request (`<op>` `0`), when the broker closes the connection, or when the client disconnects for other reasons.
- `9` - While connected, when the broker responds to a keep alive ping.

See *MQTT event notification #XMQTTEVT* for the notification syntax and parameter descriptions.

11.3.1 Set command

The set command allows you to connect to and disconnect from the MQTT broker.

Note:

The `#XMQTTCON` command uses default PDN connection with ID `0`. Raw sockets must not use the PDN connection at the same time. See *Socket AT commands* for more information.

Syntax

```
AT#XMQTTCON=<op>[,<username>,<password>,<url>,<port>[,<sec_tag>]]
```

The parameters and their defined values are the following:

<op>

- `0` - Disconnect from the MQTT broker.
- `1` - Connect to the MQTT broker using IP protocol family version 4.
- `2` - Connect to the MQTT broker using IP protocol family version 6.

<username>
String. The MQTT client username.

<password>
String. The MQTT client password in cleartext.

<url>
String. The MQTT broker hostname.

<port>
Integer. An unsigned 16-bit integer (0 - 65535) indicating the MQTT broker port.

<sec_tag>
Integer. The credential of the security tag used for establishing a secure connection.

Examples

```
AT#XMQTTCFG="MyMQTT-Client-ID",300,1
OK

AT#XMQTTCON=1,"","","mqtt.server.com",1883
OK
#XMQTTEVT: 0,0

Keep alive expires and broker responds to our ping:
#XMQTTEVT: 9,0
```

```
AT#XMQTTCON=0
OK
#XMQTTEVT: 1,0
```

When the MQTT broker closes the connection:

```
#XMQTTEVT: 1,0

AT#XMQTTCON=0
OK
```

11.3.2 Read command

The read command shows MQTT client information.

Syntax

```
AT#XMQTTCON?
```

Response syntax

```
#XMQTTCON: <status>[,<client_id>,<url>,<port>[,<sec_tag>]]
```

The parameters and their defined values are the following:

<status>

- `0` - MQTT is not connected.
- `1` - MQTT is connected.

<client_id>
String. The MQTT client ID.

<url>
String. The MQTT broker hostname. Present only when `<status>` is `1`.

<port>
Integer. An unsigned 16-bit integer (0 - 65535) indicating the MQTT broker port. Present only when `<status>` is `1`.

<sec_tag>
Integer. The credential of the security tag used for establishing a secure connection. Present only when `<status>` is `1`.

Examples

```
AT#XMQTTCON?
#XMQTTCON: 1,"","","mqtt.server.com",1883
OK
```

11.3.3 Test command

The test command tests the existence of the command and provides information about the type of its subparameters.

Syntax

```
#XMQTTCON=?
```

Response syntax

```
#XMQTTCON: (list of op),<username>,<password>,<url>,<port>,<sec_tag>
```

Examples

```
AT#XMQTTCON=?
#XMQTTCON: (0,1,2),<username>,<password>,<url>,<port>,<sec_tag>
OK
```

11.4 Incoming message notification #XMQTTMSG

When the MQTT client has successfully subscribed to a topic using *MQTT subscribe #XMQTTSUB* and a message is published on that topic, the nRF91M1 module sends #XMQTTMSG unsolicited notification.

11.4.1 Unsolicited notification

```
#XMQTTMSG: <topic_length>,<message_length>
<topic_received>
<message>
```

The parameters and their defined values are the following:

<topic_length>
Integer. The length of the <topic_received> field.

<message_length>
Integer. The length of the <message> field.

<topic_received>
String. The topic that received the message.

<message>
Raw payload bytes. The message received from the topic, forwarded without conversion or escaping. Read exactly <message_length> bytes, as the payload can contain NUL bytes and line breaks.

11.5 MQTT subscribe #XMQTTSUB

The #XMQTTSUB command allows you to subscribe to an MQTT topic.

On successful subscription, the nRF91M1 module sends #XMQTTEVT with <evt_type> 7 (SUBACK).

When a message is published on a subscribed topic, the nRF91M1 module sends #XMQTTMSG and #XMQTTEVT with <evt_type> 2 (PUBLISH). For incoming QoS 2 messages, the nRF91M1 module also sends #XMQTTEVT with <evt_type> 5 (PUBREL).

See *MQTT event notification #XMQTTEVT* and *Incoming message notification #XMQTTMSG* for the notification syntax and parameter descriptions.

11.5.1 Set command

The set command allows you to subscribe to an MQTT topic.

Syntax

```
AT#XMQTTSUB=<topic>,<qos>
```

The parameters and their defined values are the following:

<topic>
String. The topic to subscribe to.

<qos>
The MQTT Quality of Service type.

- `0` - Lowest Quality of Service. No acknowledgment of the reception is needed for the published message.
- `1` - Medium Quality of Service. If the acknowledgment of the reception is expected for the published message, publishing duplicate messages is permitted.
- `2` - Highest Quality of Service. The acknowledgment of the reception is expected and the message should be published only once.

Examples

```
AT#XMQTTSUB="nrf91/sm/mqtt/topic0",0
OK
#XMQTTEVT: 7,0

Message with QoS0 is received:
#XMQTTMSG: 21,7
nrf91/sm/mqtt/topic0
message
#XMQTTEVT: 2,0
```

```
AT#XMQTTSUB="nrf91/sm/mqtt/topic1",1
OK
#XMQTTEVT: 7,0

Message with QoS1 is received:
#XMQTTMSG: 21,7
nrf91/sm/mqtt/topic1
message

#XMQTTEVT: 2,0
```

```
AT#XMQTTSUB="nrf91/sm/mqtt/topic2",2
OK
#XMQTTEVT: 7,0

Message with QoS2 is received:
#XMQTTMSG: 21,7
nrf91/sm/mqtt/topic2
message

#XMQTTEVT: 2,0
```

(continues on next page)

(continued from previous page)

```
#XMQTTEVT: 5,0
```

11.5.2 Read command

The read command is not supported.

11.5.3 Test command

The test command is not supported.

11.6 MQTT unsubscribe #XMQTTUNSUB

The #XMQTTUNSUB command allows you to unsubscribe from an MQTT topic.

On successful unsubscription, the nRF91M1 module sends #XMQTTEVT with <evt_type> 8 (UNSUBACK).

See *MQTT event notification #XMQTTEVT* for the notification syntax and parameter descriptions.

11.6.1 Set command

The set command allows you to unsubscribe from an MQTT topic.

Syntax

```
AT#XMQTTUNSUB=<topic>
```

The parameters and their defined values are the following:

<topic>
String. The topic to unsubscribe from.

Examples

```
AT#XMQTTUNSUB="nrf91/sm/mqtt/topic0"
OK
#XMQTTEVT: 8,0
```

11.6.2 Read command

The read command is not supported.

11.6.3 Test command

The test command is not supported.

11.7 MQTT publish #XMQTTPUB

The #XMQTTPUB command allows you to publish messages on MQTT topics.

Depending on the QoS level, this command can send #XMQTTEVT with the following <evt_type> values:

- 3 - QoS 1 (PUBACK).
- 4 and 6 - QoS 2 (PUBREC, then PUBCOMP).

QoS 0 does not produce an #XMQTTEVT notification.

See *MQTT event notification #XMQTTEVT* for the notification syntax and parameter descriptions.

11.7.1 Set command

The set command allows you to publish messages on MQTT topics.

Syntax

```
AT#XMQTTPUB=<topic>[,<msg>[,<qos>[,<retain>[,<data_len>]]]]
```

The parameters and their defined values are the following:

<topic>
String. The topic on which data is published.

<msg>
String. The payload on the topic being published. If the payload is empty (for example, ""), nRF91M1 module enters *Data mode*.

<qos>

- 0 - Lowest Quality of Service (default value). No acknowledgment of the reception is needed for the published message.
- 1 - Medium Quality of Service. If the acknowledgment of the reception is expected for the published message, publishing duplicate messages is permitted.
- 2 - Highest Quality of Service. The acknowledgment of the reception is expected and the message should be published only once.

<retain>
Integer. Its default value is 0. When 1, it indicates that the broker should store the message persistently.

<data_len>
Integer. Optional, only used when <msg> is empty (data mode). Sets the number of bytes of payload to publish in data mode. When the required number of bytes are received, the payload is published and the data mode is exited.

The termination command +++ is not used in this case. The value must not exceed 4096 bytes, as the payload must fit within the data mode buffer to be published as a single message.

The value `0` is equivalent to omitting the parameter. Specifying a non-zero <data_len> together with a non-empty <msg> results in an error.

Examples

```
AT#XMQTTPUB="nrf91/sm/mqtt/topic0","Test message with QoS 0",0,0
OK
```

```
AT#XMQTTPUB="nrf91/sm/mqtt/topic0"
OK
{"msg":"Test Json publish"}+++
#XDATAMODE: 0
```

```
AT#XMQTTPUB="nrf91/sm/mqtt/topic0","",0,0,23
OK
Test message, 23 bytes.
#XDATAMODE: 0
```

```
AT#XMQTTPUB="nrf91/sm/mqtt/topic1","Test message with QoS 1",1,0
OK
#XMQTTEVT: 3,0
```

```
AT#XMQTTPUB="nrf91/sm/mqtt/topic2","",2,0
OK
Test message with QoS 2+++
#XDATAMODE: 0
#XMQTTEVT: 4,0
#XMQTTEVT: 6,0
```

11.7.2 Read command

The read command is not supported.

11.7.3 Test command

The test command is not supported.

12 nRF Cloud AT commands

The page describes the following nRF Cloud-related AT commands:

- nRF Cloud connection (see *nRF Cloud access #XNRFCLOUD*)
- nRF Cloud location (see *nRF Cloud location #XNRFCLOUDPOS*)
- nRF Cloud FOTA (see *nRF Cloud FOTA #XNRFCLOUDFOTA*)
- nRF Cloud observability (see *Observability #XNRFCLOUDOBS** and commands starting with `#XNRFCLOUDOBS`)

Note:

Before using nRF Cloud AT commands, you must onboard and claim the device in nRF Cloud. See *nRF Provisioning AT commands* for information about device provisioning and claiming.

Observability #XNRFCLOUDOBS*

The `#XNRFCLOUDOBS*` commands control the Memfault data that the device collects (metrics, events, logs, and, in builds that include it, a coredump) and its upload to nRF Cloud over the CoAP transport. See *nRF Cloud observability* for how the data is collected and how it ends up in nRF Cloud.

To use the `#XNRFCLOUDOBS*` commands that access the network (`#XNRFCLOUDOBSUPLOAD` and `#XNRFCLOUDOBSFORWARD`), the device must be connected to nRF Cloud. See *nRF Cloud access #XNRFCLOUD*.

Upload is host-driven. Automatic uploads are disabled by default. The host enables them with `AT#XNRFCLOUDOBSAUTO=1` or initiates uploads on demand with `AT#XNRFCLOUDOBSUPLOAD`.

The `<project_key>` parameter, used by several commands, is a string. It is a 32-character Memfault project key. When it is present and not empty, it overrides the server-side project-key routing, sending the data to the specified Memfault project. For more information about Memfault project keys, see Memfault Project Keys. Find your project key in Memfault Project Settings.

12.1 nRF Cloud access #XNRFCLOUD

The `#XNRFCLOUD` command controls the access to the nRF Cloud service.

Note:

To use #XNRFCLOUD, the following preconditions apply:

- You must first onboard the device to nRF Cloud, using the device-specific UUID as the device ID. See nRF Cloud Preconnect onboarding for more information.
- The device must have access to nRF Cloud through the LTE network.

12.1.1 Set command

The set command allows you to access the nRF Cloud service.

Note:

The #XNRFCLOUD command uses default PDN connection with ID 0. Raw sockets must not use the PDN connection at the same time. See *Socket AT commands* for more information.

Syntax

```
AT#XNRFCLOUD=<op>[,<send_location>]
```

The parameters and their defined values are the following:

<op>

- 0 - Disconnect from the nRF Cloud service.
- 1 - Connect to the nRF Cloud service.
- 2 - Send a message in the JSON format to the nRF Cloud service.

When <op> is 2, nRF91M1 module enters *Data mode*.

<send_location>

Used only when the value of <op> is 1.

- 0 - The device location is not sent to nRF Cloud. This is the default behavior if the parameter is omitted.
- 1 - The device location is sent to nRF Cloud.

Note:

The location is sent to the nRF Cloud whenever a fix is produced by the GNSS module. You must use the *#XGNSS* AT command to start GNSS either in single-fix or periodic navigation mode. The interval between fixes must be at least 5 seconds.

Unsolicited notification

```
#XNRFCLOUD: <ready>,<send_location>
```

The parameters and their defined values are the following:

<ready>
Integer. Indicates whether the connection to nRF Cloud is established or not.

<send_location>
Integer. Indicates whether the device location will be sent to nRF Cloud or not.

Example

```
// Connect to nRF Cloud without sending location.
AT#XNRFCLOUD=1

OK

#XNRFCLOUD: 1,0
// Send a message to nRF Cloud.
AT#XNRFCLOUD=2

OK
{"msg":"Hello, nRF Cloud"}+++

#XDATAMODE: 0
// Disconnect from nRF Cloud.
AT#XNRFCLOUD=0

OK

#XNRFCLOUD: 0,0
// Connect to nRF Cloud and send location.
AT#XNRFCLOUD=1,1

OK

#XNRFCLOUD: 1,1
AT#XNRFCLOUD=0

OK

#XNRFCLOUD: 0,1
```

12.1.2 Read command

The read command checks whether the connection to nRF Cloud is established or not.

Syntax

```
AT#XNRFCLOUD?
```

Response syntax

```
#XNRFCLOUD: <ready>,<send_location>,<sec_tag>,<device_id>
```

The parameters and their defined values are the following:

<ready>
Integer. Indicates whether the connection to nRF Cloud is established or not.

<send_location>
Integer. Indicates whether the device location will be sent to nRF Cloud or not.

<sec_tag>
Integer. The `sec_tag` used for accessing nRF Cloud.

<device_id>
String. The device ID used for accessing nRF Cloud.

Example

```
AT#XNRFCLOUD?

#XNRFCLOUD: 1,0,16842753,"50503041-3633-4261-803d-1e2b8f70111a"

OK
```

12.1.3 Test command

The test command tests the existence of the command and provides information about the type of its subparameters.

Syntax

```
AT#XNRFCLOUD=?
```

Example

```
AT#XNRFCLOUD=?

#XNRFCLOUD: (0,1,2),<send_location>

OK
```

12.2 nRF Cloud location #XNRFCLOUDPOS

The #XNRFCLOUDPOS command sends a request to nRF Cloud to determine the device's location. The request uses information from the cellular network, Wi-Fi® access points, or both.

Note:

To use #XNRFCLOUDPOS, the device must be connected to nRF Cloud using *#XNRFCLOUD*.

12.2.1 Set command

The set command allows sending a location request to nRF Cloud.

Syntax

```
AT#XNRFCLOUDPOS=<cell_count>,<wifi_pos>[,<MAC 1>[,<RSSI 1>],<MAC 2>
[,<RSSI 2>][,<MAC 3>[...]]]
```

The parameters and their defined values are the following:

<cell_count>
Integer. The number of cells to include in the location request. The value range is 0 to 15. For cellular positioning, a recommended value is 4. 0 means that no cellular network information will be included in the location request. The nRF91M1 module uses the AT%NCELLMEAS command to retrieve the cellular network information, and depending on the value of <cell_count>, the command might be executed multiple times.

Note:

Since the nRF91M1 module uses the AT%NCELLMEAS command internally, the host must not use the AT%NCELLMEAS command during #XNRFCLOUDPOS command execution. You may still use AT%NCELLMEAS command before or after #XNRFCLOUDPOS command execution for your own purposes. You will also see %NCELLMEAS notifications, which you can ignore, during the #XNRFCLOUDPOS command execution.

<wifi_pos>

- 0 - Do not include Wi-Fi access point information in the location request.
- 1 - Use Wi-Fi access point information. The access points must be given as additional parameters to the command. The minimum number of access points to provide is two (NRF_CLOUD_LOCATION_WIFI_AP_CNT_MIN), and the maximum is limited by the maximum size of the AT command, which is 8190 bytes, including the terminator character.

<MAC x>
String. The MAC address of a Wi-Fi access point. Must be formatted as

`%02x:%02x:%02x:%02x:%02x:%02x` (`WIFI_MAC_ADDR_TEMPLATE`).

<RSSI x>
Integer. Optional. The signal strength of a Wi-Fi access point in dBm, between `-128` and `0`. If provided, it must follow the MAC address parameter of the access point. Providing the RSSI parameters helps improve the accuracy of the Wi-Fi location.

Unsolicited notification

```
#XNRFCLOUDPOS: <status>[,<type>,<latitude>,<longitude>,<uncertainty>
]
```

The notification is sent when the location request completes.

The parameters and their defined values are the following:

<status>

- `0` - Successful request. Other parameters are also present.
- `-1` - Location request failed.
- `<positive integer>` - Requesting location from the cloud failed with cloud error as defined in `nrf_cloud_error` values.

<type>
The service used to fulfill the location request.

- `0` (`LOCATION_TYPE_SINGLE_CELL`) - Single-cell cellular location.
- `1` (`LOCATION_TYPE_MULTI_CELL`) - Multi-cell cellular location.
- `2` (`LOCATION_TYPE_WIFI`) - Wi-Fi location.

<latitude>
Float. The latitude in degrees.

<longitude>
Float. The longitude in degrees.

<uncertainty>
Integer. The radius of the uncertainty circle around the location in meters, also known as Horizontal Positioning Error (HPE).

Example

```
AT%XSYSTEMMODE=1,0,0,0

OK
AT+CFUN=1

OK
AT#XNRFCLOUD=1
```

(continues on next page)

(continued from previous page)

```
OK

#XNRFCLOUD: 1,0
AT#XNRFCLOUDPOS=1,0

OK

#XNRFCLOUDPOS: 0,0,35.455833,139.626111,1094

AT#XNRFCLOUDPOS=5,0

OK

%NCELLMEAS: 0,"0199F10A","44020","107E",65535,3750,5,49,27,107504,
3750,251,33,4,0,475,107,26,14,25,475,58,26,17,25,475,277,24,9,25,
475,51,18,1,25

%NCELLMEAS: 0,"01234567","44020","0340",50,175456,3400,34,5,24,
1775066,1,0,"00143FAE","44020","0140",65535,0,6200,47,40,14,1775066,
0,0

%NCELLMEAS: 0,"00987654","44020","0240",50,1754746,5500,44,4,4,
1463457,1,0,"002F4344","44020","0140",65535,0,6200,47,40,14,1775066,
0,0,"001C0502","44013","5A00",65535,0,6400,130,29,18,1775124,0,0,
"00136107","44013","5A00",65535,0,3600,202,26,13,234533,0,0

#XNRFCLOUDPOS: 0,1,35.455833,139.626111,1094
AT#XNRFCLOUDPOS=0,1,"40:9b:cd:c1:5a:40","00:90:fe:eb:4f:42"

OK

#XNRFCLOUDPOS: 0,2,35.457335,139.624443,60
AT#XNRFCLOUDPOS=0,1,"40:9b:cd:c1:5a:40",-40,"00:90:fe:eb:4f:42",-69

OK

#XNRFCLOUDPOS: 0,2,35.457346,139.624449,20
```

12.2.2 Read command

The read command is not supported.

12.2.3 Test command

The test command is not supported.

12.3 nRF Cloud FOTA #XNRFCLOUDFOTA

The #XNRFCLOUDFOTA command checks for and downloads an application or modem firmware update using Memfault release management, delivered over the same nRF Cloud CoAP transport as #XNRFCLOUDOBS*.

An application update is staged the same way as AT#XFOTA=1, that is, the host activates it with the AT#XRESET AT command. A modem update is staged the same way as AT#XFOTA=2, that is, the host activates it with the AT#XMODEMRESET AT command. Both share their FOTA session with #XFOTA, but progress and completion are reported over the #XNRFCLOUDFOTA notification instead, using the same <fota_stage>, <fota_status> and <fota_info> tuple as #XFOTA (see *FOTA AT commands* for download and activation semantics). Only one FOTA session can be ongoing at a time, regardless of which command initiated it.

Note:

Unlike AT#XFOTA, #XNRFCLOUDFOTA does not support MCUboot bootloader updates. This is expected to be a rare use case. The support is planned for a future release.

Note:

<op>=2 uses a dedicated Memfault project key for modem firmware, obtained from the General section of the modem firmware project's Project Settings, which is separate from the application project.

Use the <project_key> parameter to set it at runtime.

When neither is set, the request targets the default project where the device was claimed.

12.3.1 Set command

The set command starts a FOTA check, and the download if requested and an update is available.

Note:

The #XNRFCLOUDFOTA command uses default PDN connection with ID 0. Raw sockets must not use the PDN connection at the same time. See *Socket AT commands* for more information.

Syntax

```
AT#XNRFCLOUDFOTA=<op>[,<project_key>]
```

The parameters and their defined values are the following:

<op>

- `0` - Cancel an ongoing download. This is effective only after a download has started, that is, after the first `#XNRFCLOUDFOTA` progress notification.
- `1` - Check for and download an application update.
- `2` - Check for and download a modem firmware update. If an interrupted download is detected, it is continued.
- `4` - Check for an application update, without downloading it.
- `6` - Check for a modem firmware update, without downloading it.

<project_key>

String. Project key used for the FOTA operation.

For `<op>=1` and `<op>=4`, it is the application Memfault project key, and for `<op>=2` and `<op>=6` it is the modem Memfault project key. If this option is not provided, the request targets the default project where the device was claimed.

The command returns `OK` immediately and the check runs asynchronously. When it completes, an unsolicited notification is sent.

Unsolicited notification

```
#XNRFCLOUDFOTA: <fota_stage>,<fota_status>[,<fota_info>]
```

The `<fota_stage>`, `<fota_status>`, and `<fota_info>` parameters use the same values as the `#XFOTA` notification:

<fota_stage>

- `0` - Init.
- `1` - Download.
- `2` - Download, erase pending (modem delta update only).
- `3` - Download, erase complete (modem delta update only).
- `4` - Downloaded, to be activated.
- `5` - Complete.

<fota_status>

- `0` - OK.
- `1` - Error.
- `2` - Cancelled.
- `3` - Reverted (application FOTA only).

<fota_info>

Integer. Its value can have different meanings based on the values returned by `<fota_stage>` and `<fota_status>`.

See the following table:

<fota_stage>	<fota_status>	<fota_info>
1 (namely *Download*)	0 (namely *OK*)	Percentage of the download
1 (namely *Download*)	1 (namely *ERROR*)	Error Code
1 (namely *Download*)	2 (namely *CANCELLED*)	0 - Downloading is cancelled before completion
5 (namely *Complete*)	1 (namely *ERROR*)	Error Code

The error codes can be the following:

- 1 - Download failed
- 2 - Update image rejected (for example modem firmware version error)
- 3 - Update image mismatch (for example <op> is 1 but <file_url> points to a modem image)

For modem FOTA, the error codes can be the following:

- 71303169 (NRF_MODEM_DFU_RESULT_INTERNAL_ERROR) - The modem encountered a fatal internal error during firmware update.
- 71303170 (NRF_MODEM_DFU_RESULT_HARDWARE_ERROR) - The modem encountered a fatal hardware error during firmware update.
- 71303171 (NRF_MODEM_DFU_RESULT_AUTH_ERROR) - Modem firmware update failed due to an authentication error.
- 71303172 (NRF_MODEM_DFU_RESULT_UUID_ERROR) - Modem firmware update failed due to UUID mismatch.
- 71303173 (NRF_MODEM_DFU_RESULT_VOLTAGE_LOW) - Modem firmware update not executed due to low voltage. The modem will retry the update on reboot.

When the FOTA check finishes without starting a download, <fota_stage> is 0 (*Init*) and <fota_info> indicates the result. This form is used by #XNRFCLOUDFOTA only (not #XFOTA) and specified as follows:

<fota_stage>	<fota_status>	<fota_info>
0 (namely *Init*)	0 (namely *OK*)	0 - No update is available
0 (namely *Init*)	0 (namely *OK*)	**1 - An update is available (sent only for** <op>=4 and <op>=6
0 (namely *Init*)	1 (namely *ERROR*)	Error code

Examples

```
AT#XNRFCLOUDFOTA=1

OK

#XNRFCLOUDFOTA: 1,0,45

#XNRFCLOUDFOTA: 1,0,100

#XNRFCLOUDFOTA: 4,0
AT#XRESET
```

The following example checks whether an application update is available, without downloading it:

```
AT#XNRFCLOUDFOTA=4

OK

#XNRFCLOUDFOTA: 0,0,1
```

12.3.2 Read command

The read command is not supported.

12.3.3 Test command

The test command returns the supported syntax.

Syntax

```
AT#XNRFCLOUDFOTA=?
```

Response

```
#XNRFCLOUDFOTA: (0,1,2,4,6)[,<project_key>]
```

12.4 On-demand upload #XNRFCLOUDOBSUPLOAD

The `#XNRFCLOUDOBSUPLOAD` command uploads the buffered observability data to nRF Cloud.

The captured logs are collected before the upload so that they are included.

12.4.1 Set command

The set command uploads the buffered data.

Syntax

```
AT#XNRFCLOUDOBSUPLOAD[=<project_key>]
```

The command returns `OK` immediately and the upload runs asynchronously. When it completes, an unsolicited notification is sent.

Unsolicited notification

```
#XNRFCLOUDOBSUPLOAD: <result>[,<bytes>]
```

The parameters and their defined values are the following:

<result>

- `0` - Success. The `<bytes>` parameter follows and indicates the number of bytes uploaded, which is `0` when there was nothing buffered.
- `-1` - Failure. The error code is shown in the log.

Example

```
AT#XNRFCLOUDOBSUPLOAD

OK

#XNRFCLOUDOBSUPLOAD: 0,108
```

12.4.2 Read command

The read command is not supported.

12.4.3 Test command

The test command returns the supported syntax.

Syntax

```
AT#XNRFCLOUDOBSUPLOAD=?
```

Response

```
#XNRFCLOUDOBSUPLOAD: <project_key>
```

12.5 Metrics heartbeat #XNRFCLOUDOBSHEARTBEAT

The #XNRFCLOUDOBSHEARTBEAT command collects and finalizes a metrics heartbeat.

The data is buffered on the device until it is uploaded, so the command does not require a connection to nRF Cloud.

The heartbeat carries the LTE metrics that the modem reports, such as the modem firmware version, the network operator, RSRP, SNR, the current band, and the transmitted and received data in kilobytes.

The heartbeat also carries the cell ID, the tracking area code, the LTE mode, the PSM active time and periodic TAU, and the eDRX interval and paging time window.

The heartbeat additionally carries connection statistics that are tracked between heartbeats, including the time to connect to the network, the number of connection losses, and the time the LTE link has been active. Values not reported by the modem, such as when it is deactivated, are excluded from the heartbeat.

12.5.1 Set command

The set command collects a heartbeat.

Syntax

```
AT#XNRFCLOUDOBSHEARTBEAT
```

Example

```
AT#XNRFCLOUDOBSHEARTBEAT

OK
```

12.5.2 Read command

The read command is not supported.

12.5.3 Test command

The test command is not supported.

12.6 Chunk forwarding #XNRFCLOUDOBSFORWARD

The `#XNRFCLOUDOBSFORWARD` command forwards a Memfault chunk produced by the host to nRF Cloud, using the nRF91M1 module connection.

Note:

nRF Cloud attributes the chunk to the authenticated device, so the observability data of the host is reported under the device serial of the nRF91M1 module device by default. To report it under a device serial of the host instead, build the host firmware with `MEMFAULT_EVENT_INCLUDE_DEVICE_SERIAL` set to `1` in its `memfault_platform_config.h`. Every chunk then carries the device serial of the host, which Memfault uses to attribute the data.

12.6.1 Set command

The set command forwards a chunk.

Syntax

```
AT#XNRFCLOUDOBSFORWARD=<base64_chunk>[,<project_key>]
```

The parameters and their defined values are the following:

<base64_chunk>
String. The base64-encoded Memfault chunk to forward.

The command returns `OK` immediately and the chunk is posted asynchronously. When the post completes, an unsolicited notification is sent.

Unsolicited notification

```
#XNRFCLOUDOBSFORWARD: <result>
```

The parameters and their defined values are the following:

<result>

- `0` - Success.
- `-1` - Failure. The chunk is not buffered, so the host must send it again to retry.

Example

Forward a chunk of the host, sent to a specific Memfault project:

```
AT#XNRFCLOUDOBSFORWARD="CAKnAgIDAQpqdGVzdHNlcmlhbA==","<project_key>
"
```

(continues on next page)

(continued from previous page)

```
OK

#XNRFCLOUDOBSFORWARD: 0
```

12.6.2 Read command

The read command is not supported.

12.6.3 Test command

The test command returns the supported syntax.

Syntax

```
AT#XNRFCLOUDOBSFORWARD=?
```

Response

```
#XNRFCLOUDOBSFORWARD: <base64_chunk>,<project_key>
```

12.7 Core dump upload #XNRFCLOUDOBSCOREDUMP

The `#XNRFCLOUDOBSCOREDUMP` command controls whether a stored core dump is included in an upload.

When disabled, `#XNRFCLOUDOBSUPLOAD` and the automatic upload send the buffered events, logs, and CDRs but leave the core dump in storage, so it can be uploaded later by enabling this again. Core dump upload is enabled by default, and the setting is persisted.

Note:

Core dump storage holds a single core dump. While a core dump is stored, subsequent crashes do not capture a new core dump until the stored one has been uploaded. Keeping the upload disabled prevents new core dumps from being captured.

12.7.1 Set command

The set command enables or disables the core dump upload.

Syntax

```
AT#XNRFCLOUDOBSCOREDUMP=<enable>
```

The parameters and their defined values are the following:

<enable>

- `0` - Exclude the stored core dump from uploads.
- `1` - Include the stored core dump in uploads.

12.7.2 Read command

The read command returns the current setting.

Syntax

```
AT#XNRFCLOUDOBSCOREDUMP?
```

Response

```
#XNRFCLOUDOBSCOREDUMP: <enable>
```

12.7.3 Test command

The test command returns the supported syntax.

Syntax

```
AT#XNRFCLOUDOBSCOREDUMP=?
```

Response

```
#XNRFCLOUDOBSCOREDUMP: (0,1)
```

12.8 Automatic upload #XNRFCLOUDOBSAUTO

The `#XNRFCLOUDOBSAUTO` command configures the automatic upload of the buffered observability data.

The configuration is persistent, so it survives a reboot.

12.8.1 Set command

The set command enables or disables the automatic upload and configures its interval.

Syntax

```
AT#XNRFCLOUDOBSAUTO=<enable>[,<interval_seconds>][,<project_key>]
```

The parameters and their defined values are the following:

<enable>

- `0` - Disable the automatic upload.
- `1` - Enable the automatic upload.

<interval_seconds>
Integer. The interval between two uploads, from `60` to `86400` seconds. When omitted, the stored interval is kept.

The initial value is 3600 seconds.

<project_key>
String. When omitted, the stored project key is kept, and an empty string clears it.

The first upload runs when the interval expires, not when the automatic upload is enabled. An upload that falls while there is no connection to nRF Cloud is skipped, and the next one is scheduled as usual.

The automatic upload is silent and does not send any unsolicited notifications.

Response

The command returns `OK`, also when the configuration could not be stored, in which case only its persistence is lost.

12.8.2 Read command

The read command returns the configuration of the automatic upload.

Syntax

```
AT#XNRFCLOUDOBSAUTO?
```

Response

```
#XNRFCLOUDOBSAUTO: <enable>,<interval_seconds>,<project_key>
```

Example

```
AT#XNRFCLOUDOBSAUTO=1,600

OK
```

(continues on next page)

(continued from previous page)

```
AT#XNRFCLOUDOBSAUTO?

#XNRFCLOUDOBSAUTO: 1,600,""

OK
```

12.8.3 Test command

The test command returns the supported syntax.

Syntax

```
AT#XNRFCLOUDOBSAUTO=?
```

Response

```
#XNRFCLOUDOBSAUTO: (0,1),(60-86400),<project_key>
```

13 nRF Provisioning AT commands

This page describes AT commands for the nRF Device Provisioning service. The provisioning client connects to the nRF Cloud Provisioning Service over CoAP or DTLS and applies device configuration commands (credentials, settings, and firmware updates) issued from the cloud.

To use the provisioning service with the nRF91M1 module, you must first claim the device using its attestation token:

1. Retrieve the attestation token with the `AT%ATTESTTOKEN` command.
2. Enter the token as described in the nRF Cloud Provisioning Service documentation.
3. Run the `AT#XNRFPROV` command to trigger provisioning, as described in *Trigger provisioning #XNRFPROV*. During the session, the host must respond to the `#XNRFPROV: 1` and `#XNRFPROV: 2` notifications. The host takes the modem offline to write credentials, and then brings it back online to reconnect to the provisioning server. See the *unsolicited notifications* section for details.

13.1 Trigger provisioning #XNRFPROV

The `#XNRFPROV` command triggers an immediate provisioning attempt.

13.1.1 Set command

The set command starts a provisioning session immediately, independently of the configured schedule interval.

Syntax

```
AT#XNRFPROV
```

Response syntax

```
OK
```

Unsolicited notifications

One of the following #XNRFPROV notifications is emitted when the provisioning session completes, an event occurs, or host action is required:

```
#XNRFPROV: <status>
```

The parameters and their defined values are the following:

<status>

- 0 - Provisioning successful, or no pending commands on the server. The two cases are not distinguished.
- 1 - Host action required: deactivate LTE. The host must take the modem offline (for example AT+CFUN=4) so that credentials can be written safely.

  The provisioning client waits until the modem reports offline functional mode before continuing or times out after 120 seconds.
- 2 - Host action required: activate LTE. The host must bring the modem back online (for example AT+CFUN=1) to reconnect to the provisioning server.

  The provisioning client waits until the modem reports LTE registration before continuing or times out after 120 seconds.
- -1 - Provisioning failed. The host can retry provisioning by triggering the #XNRFPROV command again.
- -2 - Device not claimed on nRF Cloud. The device must be claimed using its attestation token. Refer to the nRF Cloud Provisioning Service documentation for more information.
- -3 - Wrong root CA certificate. Provision the correct nRF Cloud root CA certificate.
- -4 - No valid datetime reference. The modem must have a valid time before provisioning can proceed.
- -5 - Too many commands received from the server. The maximum number of records is 5.
- -6 - Fatal error: the provisioning client encountered an irrecoverable error.

Example

The provisioning client requests the host to manage the LTE connection during the session. The host must respond to the #XNRFPROV: 1 and #XNRFPROV: 2 notifications accordingly.

```
AT#XNRFPROV
OK

#XNRFPROV: 1

AT+CFUN=4
OK
```

(continues on next page)

(continued from previous page)

```
#XNRFPROV: 2

AT+CFUN=1
OK

#XNRFPROV: 0
```

13.1.2 Test command

The test command returns the command syntax.

Syntax

```
AT#XNRFPROV=?
```

Response syntax

```
#XNRFPROV
```

14 PPP AT commands

This page describes AT commands related to the Point-to-Point Protocol (PPP).

PPP is enabled in nRF91M1 module by compiling it with the appropriate configuration files, depending on your use case (with or without CMUX). See the Configuration files section for more information.

14.1 Enter data state +CGDATA

The AT+CGDATA command enters data state and starts PPP, as specified in 3GPP TS 27.007 (section 10.1.12).

14.1.1 Set command

The set command starts PPP on the current channel, optionally specifying the layer 2 protocol and PDN connection.

Syntax

```
AT+CGDATA[=<L2P>[,<cid>]]
```

The parameters and their defined values are the following:

<L2P>
: String. The layer 2 protocol to use. The only supported value is `"PPP"`. If omitted, `"PPP"` is assumed.

<cid>
: Integer. The PDN connection to use. Its default value is `0`, which represents the default PDN connection.

Response syntax

```
CONNECT
```

Examples

With default parameters:

```
AT+CGDATA

CONNECT
```

Request PPP to use specified PDN connection:

```
AT+CGDATA="PPP",1

CONNECT
```

14.1.2 Test command

The test command lists the supported layer 2 protocols.

Syntax

```
AT+CGDATA=?
```

Response syntax

```
+CGDATA: (<L2P_values>)
```

Example

```
AT+CGDATA=?

+CGDATA: ("PPP")

OK
```

14.2 Control PPP #XPPP

Note:

AT#XPPP is a compatibility command for Serial Modem v1.x.x and is not recommended for new designs. It depends on the channel assignment configured by the AT#XCMUX command. Use the standard AT+CGDATA command instead.

14.2.1 Set command

The set command request activation or deactivation of PPP link, and optionally define the PDN connection used for PPP. PPP link is established on the second channel (DLC channel

2) when CMUX is used. If the PPP link is preferred on the first channel (DLC channel 1), you must use the AT#XCMUX=2 command to switch the AT command channel to DLC channel 2 before starting PPP.

If PPP is started without CMUX, the current UART switches to PPP mode.

Note:

When a PPP start has been issued, the PPP connection is automatically activated and deactivated when the PDN connection requested for PPP is established and lost, respectively. This will continue until a PPP stop is issued by either the user by the AT#XPPP=0 command or by the remote peer disconnecting the PPP using LCP termination.

Note:

When PPP is started without CMUX, the current UART cannot be used for AT commands until PPP is stopped by LCP termination.

Syntax

```
AT#XPPP=<op>[,<cid>]
```

The parameters and their defined values are the following:

<op>

- 0 - Stop PPP.
- 1 - Start PPP.

<cid>

Integer. The PDN connection to be used for PPP. It represents cid in the +CGDCONT command. Its default value is 0, which represents the default PDN connection.

Note:

Other sockets cannot use the same PDN connection. See *Socket AT commands* for more information.

Unsolicited notification

```
#XPPP: <running>,<peer_connected>,<cid>
```

The parameters and their defined values are the following:

<running>

- 0 - PPP is stopped.

- `1` - PPP is running.

<peer_connected>

- `0` - No peer is connected.
- `1` - A peer is connected.

<cid>
Integer. The PDN connection used for PPP.

When you activate a PDN connection used for PPP, the `#XPPP: 1,0,<cid>` notification is sent and the PPP process starts and takes ownership of the associated serial channel. If you use the CMUX , the PPP connection starts at the alternative DLC channel, which can be controlled by the `AT#XCMUX` command. Without CMUX, the PPP starts on the current UART, and AT commands cannot be used until PPP is stopped by the LCP termination message. Similarly, no more AT notifications are sent over the same UART and all URC messages are lost until PPP is stopped.

Note:

PPP auto-start uses `+CGEV:` URCs inside nRF91M1 module. These URCs are not forwarded to the host unless you enable them with `AT+CGEREP=1`. The host receives PPP status through the `#XPPP` notification.

14.2.2 PPP Link termination

PPP link termination behaves differently in various scenarios.

When the PPP is started on the UART without CMUX, the PPP link termination always returns the channel back to AT command mode.

When the PPP is started over CMUX, the behavior depends on the termination reason and channel configuration:

Table 1: PPP link termination reason

Reason	Description
PDN connection lost when PPP is running on DLC channel 2	The PPP link is terminated using the LCP Terminate-Request message. The PPP module keeps waiting for the PDN connection to be re-established to restart PPP automatically. The Remote peer might wait for LCP Config-Requests to re-establish the link.
PDN connection lost when PPP is running on DLC channel 1	The PPP link is terminated using the LCP Terminate-Request message. The PPP module stops without trying to restart. DLC channel 1 is returned to AT command mode. Traffic on DLC channel 2 is ignored.
Peer disconnection using the LCP Terminate-Request	The PPP module stops PPP and releases the CMUX channel. It does not attempt to restart.
PPP stopped by AT command	The PPP module stops PPP and releases the CMUX channel. It does not attempt to restart.

As the `AT#XPPP=1` command is designed to restart the PPP connection automatically when the DLC channel used for PPP is the second channel (DLC channel 2). This is to allow the PPP connection to recover from temporary network losses without your intervention while still allowing you to stop PPP when needed. This behavior differs from the standard ATD*99# command, which is not implemented on nRF91M1 module.

14.2.3 Examples

The following examples assume CMUX is already established so that the AT channel is usable while PPP is running on a separate CMUX channel.

PPP with default PDN connection:

```
// Start PPP.
AT#XPPP=1

OK

AT+CFUN=1

OK

// PPP is started and waits for a peer to connect.
#XPPP: 1,0,0

// Peer connects to |SM|'s PPP.
#XPPP: 1,1,0

// Peer disconnects.
#XPPP: 1,0,0

// |SM| stops PPP when a peer disconnects.
#XPPP: 0,0,0

AT+CFUN=4

OK
```

PPP with non-default PDN connection:

```
// Exemplary PDN connection creation.
// Note: APN depends on operator and additional APNs may not be
supported by the operator.
AT+CGDCONT=1,"IP","internet2"

OK

// Start PPP with the created PDN connection.
AT#XPPP=1,1
```

(continues on next page)

(continued from previous page)

```
OK

AT+CFUN=1

OK

// Activate the created PDN connection.
AT+CGACT=1,1

// PPP is automatically started when the PDN connection set for PPP
has been activated.
#XPPP: 1,0,1

// Peer connects to |SM|'s PPP.
#XPPP: 1,1,1
```

Connection recovery for network loss. This requires the PPP on the peer side to keep retrying or waiting for LCP Config-Requests.

```
// Simulate connection loss by activating the flight mode.
AT+CFUN=4

OK

// PPP is automatically stopped when the PDN connection has been
deactivated.
#XPPP: 0,0,0

// Reactivate the connection.
AT+CFUN=1

OK

// PPP is automatically restarted when the PDN connection has been
reactivated.
#XPPP: 1,0,0

// Peer connects to |SM|'s PPP.
#XPPP: 1,1,0
```

14.2.4 Read command

The read command allows you to get the status of PPP.

Syntax

```
AT#XPPP?
```

Response syntax

```
#XPPP: <running>,<peer_connected>,<cid>
```

The parameters and their defined values are the following:

<running>

- `0` - PPP is stopped.
- `1` - PPP is running.

<peer_connected>

- `0` - No peer is connected.
- `1` - A peer is connected.

<cid>
Integer. The PDN connection used for PPP.

When you activate a PDN connection used for PPP, the `#XPPP: 1,0,<cid>` notification is sent and the PPP process starts and takes ownership of the associated serial channel. If you use the CMUX , the PPP connection starts at the alternative DLC channel, which can be controlled by the `AT#XCMUX` command. Without CMUX, the PPP starts on the current UART, and AT commands cannot be used until PPP is stopped by the LCP termination message. Similarly, no more AT notifications are sent over the same UART and all URC messages are lost until PPP is stopped.

Note:

PPP auto-start uses `+CGEV:` URCs inside nRF91M1 module. These URCs are not forwarded to the host unless you enable them with `AT+CGEREP=1`. The host receives PPP status through the `#XPPP` notification.

15 SMS AT commands

This page describes SMS-related AT commands.

Note:

nRF91M1 module's SMS implementation is based on the SMS library, which uses the following AT commands: `AT+CMGS`, `AT+CNMI`, `AT+CNMA`. These modem AT commands should not be used in conjunction with the `#XSMS` command by the host in order to avoid unexpected behavior. The implementation always acknowledges incoming `+CMT` and `+CDS` notifications with `AT+CNMA` command.

15.1 SMS support #XSMS

The `#XSMS` command supports functionalities for sending and receiving SMS messages.

15.1.1 Set command

The set command allows you to start or stop SMS, as well as send SMS text. Only GSM-7 encoding is supported, 8-bit and UCS2 encodings are not supported.

Syntax

```
AT#XSMS=<op>[,<number>,<text>]
```

The parameters and their defined values are the following:

<op>

- `0` - Stop SMS.
- `1` - Start SMS, ready to receive.
- `2` - Send SMS.

<number>
String. The SMS recipient's phone number, including the country code (for example +81xxxxxxx).

<text>
String. The SMS text to be sent.

Unsolicited notification

This is the notification syntax when an SMS message is received:

```
#XSMS: <datetime>,<number>,<text>
```

The parameters and their defined values are the following:

<datetime>
String. The time when the SMS is received. It has a format of `YY-MM-DD HH:MM:SS UTC[+|-]HH:MM`.

<number>
String. The SMS sender's phone number.

<text>
String. The SMS text that has been received.

When receiving concatenated SMS messages, there will be only one notification.

Example

```
at#xsms=1

OK
at#xsms=2,"+8190xxxxxxxx","SM test"

OK

#XSMS: "21-05-24 11:58:22 UTC+09:30","090xxxxxxxx","Tested OK"

at#xsms=2,"+8190xxxxxxxx","0123456789abcdefghijklmnopqrstuvwxyzABCDE
FGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJK
LMNOPQRSTUVWXYZ0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQ
RSTUVWXYZ"

OK

#XSMS: "21-05-24 13:29:47 UTC-09:00","090xxxxxxxx","0123456789abcdef
ghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijkl
mnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnopqr
stuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

15.1.2 Read command

The read command is not supported.

15.1.3 Test command

The test command tests the existence of the command and provides information about the type of its subparameters.

Syntax

```
AT#XSMS=?
```

Response syntax

```
#XSMS: <list of op value>,<number>,<text>
```

Examples

```
at#xsms=?
#XSMS: (0,1,2),<number>,<text>
OK
```

16 Socket AT commands

This page describes socket-related AT commands. The application can open up to 8 sockets.

For more information on the networking services, see the Zephyr Network APIs.

Note:

A socket cannot be created on a PDN connection that is already in use by a raw socket. A raw socket cannot be created on a PDN connection that is already in use by another socket. The reason is that a raw socket cannot keep its data separate from the data of another IP socket when both are operating on the same PDN. When a raw socket is active, it intercepts all downlink data intended for other sockets on the same PDN, which disrupts normal socket operations. Raw sockets are used internally by the following AT commands: `AT#XPPP` and `AT#XPING`.

16.1 Socket #XSOCKET

The `#XSOCKET` command allows you to open a socket and to check the socket handle.

16.1.1 Set command

The set command allows you to open a socket.

Syntax

```
AT#XSOCKET=<family>,<type>,<role>[,<cid>]
```

The parameters and their defined values are the following:

<family>

- 1 - IP protocol family version 4.
- 2 - IP protocol family version 6.
- 3 - Raw packet family. The `<type>` parameter must be 3 and the `<role>` parameter must be `0`.

<type>

- 1 - Set `SOCK_STREAM` for the stream socket type using the TCP protocol.

- 2 - Set SOCK_DGRAM for the datagram socket type using the UDP protocol.
- 3 - Set SOCK_RAW for the raw socket type using a generic packet protocol. The <family> parameter must be 3 and the <role> parameter must be 0.

<role>

- 0 - Client.
- 1 - Server.

<cid>

Integer. Indicates the used PDN connection. It represents cid in the +CGDCONT command. Its default value is 0.

Response syntax

```
#XSOCKET: <handle>,<type>,<protocol>
```

The parameters and their defined values are the following:

<handle>

Integer. Socket handle (ID) that can be interpreted as follows:

- Positive or 0 - The socket opened successfully.
- Negative - The socket failed to open.

<type>

- 1 - Set SOCK_STREAM for the stream socket type using the TCP protocol.
- 2 - Set SOCK_DGRAM for the datagram socket type using the UDP protocol.
- 3 - Set SOCK_RAW for the raw socket type using a generic IP protocol.

<protocol>

- 0 - IPPROTO_IP.
- 6 - IPPROTO_TCP.
- 17 - IPPROTO_UDP.

Examples

```
AT#XSOCKET=1,1,0
#XSOCKET: 0,1,6
OK
AT#XSOCKET=1,2,0
#XSOCKET: 1,2,17
OK
AT#XSOCKET=2,1,0
#XSOCKET: 2,1,6
OK
AT#XSOCKET=1,3,0
```

(continues on next page)

(continued from previous page)

```
#XSOCKET: 3,3,0
OK
```

16.1.2 Read command

The read command allows you to list all sockets that have been opened.

Syntax

```
AT#XSOCKET?
```

Response syntax

```
#XSOCKET: <handle>,<family>,<role>,<type>,<cid>
```

The parameters and their defined values are the following:

<handle>
Integer. When positive or `0`, the socket is valid.

<family>
Present only in the response to a request to open the socket.

- 1 - IP protocol family version 4.
- 2 - IP protocol family version 6.
- 3 - Packet family.

<role>

- `0` - Client.
- 1 - Server.

<type>

- 1 - Set `SOCK_STREAM` for the stream socket type using the TCP protocol.
- 2 - Set `SOCK_DGRAM` for the datagram socket type using the UDP protocol.
- 3 - Set `SOCK_RAW` for the raw socket type using a generic packet protocol.

<cid>
Integer. Indicates the used PDN connection. It represents `cid` in the `+CGDCONT` command.

Example

```
AT#XSOCKET?
#XSOCKET: 0,1,0,1,0
#XSOCKET: 1,2,0,1,0
#XSOCKET: 4,1,0,2,1
OK
```

16.1.3 Test command

The test command tests the existence of the command and provides information about the type of its subparameters.

Syntax

```
AT#XSOCKET=?
```

Response syntax

```
#XSOCKET: <handle>,<list of ops>,<list of types>,<list of roles>,
<cid>
```

Example

```
AT#XSOCKET=?
#XSOCKET: <handle>,(1,2),(1,2,3),(0,1),<cid>
OK
```

16.2 Secure socket #XSSOCKET

The #XSSOCKET command allows you to open a secure socket and to check the socket handle.

Note:

TLS and DTLS servers are currently not supported.

Note:

A secure socket cannot be created on a PDN connection that is already in use by a raw socket. See *Socket AT commands* for more information.

16.2.1 Set command

The set command allows you to open a secure socket.

Syntax

```
AT#XSSOCKET=<family>,<type>,<role>,<sec_tag>[,<peer_verify>[,<cid>]]
```

The parameters and their defined values are the following:

<family>

- 1 - IP protocol family version 4.
- 2 - IP protocol family version 6.

<type>

- 1 - Set SOCK_STREAM for the stream socket type using the TLS 1.2 protocol.
- 2 - Set SOCK_DGRAM for the datagram socket type using the DTLS 1.2 protocol.

<role>

- 0 - Client.
- 1 - Server.

<sec_tag>

Integer. Indicates to the modem the credential of the security tag to be used for establishing a secure connection. It is associated with a credential, that is, a certificate or PSK. The credential must be stored on the modem side beforehand.

> **Note:**
>
> When <role> has a value of 1, <sec_tag> is not supported.

<peer_verify>

- 0 - None (default for server role).
- 1 - Optional.
- 2 - Required (default for client role).

<cid>

Integer. Indicates the used PDN connection. It represents cid in the +CGDCONT command. Its default value is 0.

Response syntax

```
#XSSOCKET: <handle>,<type>,<protocol>
```

The parameters and their defined values are the following:

<handle>
Integer. Socket handle (ID) that can be interpreted as follows:

- Positive or `0` - The socket opened successfully.
- Negative - The socket failed to open.

<type>

- 1 - `SOCK_STREAM` for the stream socket type using the TLS 1.2 protocol.
- 2 - `SOCK_DGRAM` for the datagram socket type using the DTLS 1.2 protocol.

<protocol>

- 258 - IPPROTO_TLS_1_2.
- 273 - IPPROTO_DTLS_1_2.

Examples

```
AT#XSSOCKET=1,1,0,16842753,2
#XSSOCKET: 0,1,258
OK
AT#XSSOCKET=1,2,0,16842753
#XSSOCKET: 1,2,273
OK
```

16.2.2 Read command

The read command allows you to check the secure socket handle.

Syntax

```
AT#XSSOCKET?
```

Response syntax

```
#XSSOCKET: <handle>,<family>,<role>,<type>,<sec_tag>,<cid>
```

The parameters and their defined values are the following:

<handle>
Integer. When positive or `0`, the socket is valid.

<family>

- 1 - IP protocol family version 4.
- 2 - IP protocol family version 6.

<role>

- `0` - Client.
- `1` - Server.

<type>

- `1` - `SOCK_STREAM` for the stream socket type using the TLS 1.2 protocol.
- `2` - `SOCK_DGRAM` for the datagram socket type using the DTLS 1.2 protocol.

<sec_tag>
Integer. Indicates to the modem the credential of the security tag to be used for establishing a secure connection.

<cid>
Integer. Indicates the used PDN connection. It represents `cid` in the `+CGDCONT` command.

Example

```
AT#XSSOCKET?
#XSSOCKET: 0,1,0,1,16842753,0
OK
```

16.2.3 Test command

The test command tests the existence of the command and provides information about the type of its subparameters.

Syntax

```
AT#XSSOCKET=?
```

Response syntax

```
#XSSOCKET: <handle>,<list of ops>,<list of types>,<list of roles>,
<sec_tag>,<peer_verify>,<cid>
```

Example

```
AT#XSSOCKET=?
#XSSOCKET: <handle>,(1,2),(1,2),(0,1),<sec_tag>,<peer_verify>,<cid>
OK
```

16.3 Close socket #XCLOSE

The `#XCLOSE` command allows you to close one or all sockets.

16.3.1 Set command

The set command allows you to close a specific socket or all open sockets.

Syntax

```
AT#XCLOSE[=<handle>]
```

The parameters and their defined values are the following:

<handle>
Integer. Optional. Specifies the socket handle to close. This is the handle value returned from #XSOCKET or #XSSOCKET commands. If omitted, all open sockets will be closed.

Response syntax

```
#XCLOSE: <handle>,<result>
```

The parameters and their defined values are the following:

<handle>
Integer. The handle of the closed socket.

<result>
Integer. The result of closing the socket. When `0`, the socket was closed successfully.

When closing all sockets (no handle parameter provided), multiple responses will be sent, one for each socket that was closed.

Examples

Close a specific socket:

```
AT#XCLOSE=0
#XCLOSE: 0,0
OK
```

Close all open sockets:

```
AT#XCLOSE
#XCLOSE: 0,0
#XCLOSE: 1,0
#XCLOSE: 2,0
OK
```

16.3.2 Read command

The read command is not supported.

16.3.3 Test command

The test command is not supported.

16.4 Socket options #XSOCKETOPT

The #XSOCKETOPT command allows you to get and set socket options.

16.4.1 Set command

The set command allows you to get and set socket options.

Syntax

```
AT#XSOCKETOPT=<handle>,<op>,<name>[,<value>]
```

The parameters and their defined values are the following:

<handle>
Integer. Specifies the socket handle returned from #XSOCKET or #XSSOCKET commands.

<op>

- 0 - Get.
- 1 - Set.

<name>

- 2 - AT_SO_REUSEADDR (set-only).
  - <value> is an integer that indicates whether the reuse of local addresses is enabled. It is 0 for disabled or 1 for enabled.
- 20 - AT_SO_RCVTIMEO.
  - <value> is an integer that indicates the receive timeout in seconds.
- 21 - AT_SO_SNDTIMEO.
  - <value> is an integer that indicates the send timeout in seconds.
- 30 - AT_SO_SILENCE_ALL.
  - <value> is an integer that indicates whether ICMP echo replies for IPv4 and IPv6 are disabled. It is 0 for allowing ICMP echo replies or 1 for disabling them.
- 31 - AT_SO_IP_ECHO_REPLY.
  - <value> is an integer that indicates whether ICMP echo replies for IPv4 are enabled. It is 0 for disabled or 1 for enabled.
- 32 - AT_SO_IPV6_ECHO_REPLY.
  - <value> is an integer that indicates whether ICMP echo replies for IPv6 are enabled. It is 0 for disabled or 1 for enabled.

- 34 - AT_SO_KEEPOPEN.
  - <value> is an integer that indicates whether socket is kept open when the modem enters the flight mode (+CFUN=4). It is 0 for disabled or 1 for enabled.
- 40 - AT_SO_BINDTOPDN (set-only).
  - <value> is an integer that indicates the packet data network ID to bind to.
- 55 - AT_SO_TCP_SRV_SESSTIMEO.
  - <value> is an integer that indicates the TCP server session inactivity timeout for a socket. It accepts values from the range 0 to 135, where 0 is no timeout and 135 is 2 hours, 15 minutes.
- 61 - AT_SO_RAI (set-only). Release Assistance Indication (RAI).
  - <value> The option accepts an integer, indicating the type of RAI. Accepted values for the option are:
    - 1 - RAI_NO_DATA. Indicates that the application does not intend to send more data. This socket option applies immediately and lets the modem exit connected mode more quickly.
    - 2 - RAI_LAST. Indicates that the application does not intend to send more data after the next call to send() or sendto(). This lets the modem exit connected mode more quickly after sending the data.
    - 3 - RAI_ONE_RESP. Indicates that the application is expecting to receive just one data packet after the next call to send() or sendto(). This lets the modem exit connected mode more quickly after having received the data.
    - 4 - RAI_ONGOING. Indicates that the application is expecting to receive just one data packet after the next call to send() or sendto(). This lets the modem exit connected mode more quickly after having received the data.
    - 5 - RAI_WAIT_MORE. Indicates that the socket is in active use by a server application. This lets the modem stay in connected mode longer.
- 62 - AT_SO_IPV6_DELAYED_ADDR_REFRESH.
  - <value> is an integer that indicates whether delayed IPv6 address refresh is enabled. It is 0 for disabled or 1 for enabled.

See nRF socket options for explanation of the supported options.

Examples

```
AT#XSOCKETOPT=0,1,20,30
OK
```

```
AT#XSOCKETOPT=0,0,20
#XSOCKETOPT: 0,30
OK
```

16.4.2 Read command

The read command is not supported.

16.4.3 Test command

The test command tests the existence of the command and provides information about the type of its subparameters.

Syntax

```
AT#XSOCKETOPT=?
```

Response syntax

```
#XSOCKETOPT: <handle>,<list of ops>,<name>,<value>
```

Example

```
AT#XSOCKETOPT=?
#XSOCKETOPT: <handle>,(0,1),<name>,<value>
OK
```

16.5 Secure socket options #XSSOCKETOPT

The #XSSOCKETOPT command allows you to get and set socket options for secure sockets.

16.5.1 Set command

The set command allows you to get and set socket options for secure sockets.

Syntax

```
AT#XSSOCKETOPT=<handle>,<op>,<name>[,<value>]
```

The parameters and their defined values are the following:

<handle>
Integer. Specifies the socket handle returned from #XSSOCKET command.

<op>

- 0 - Get.
- 1 - Set.

<name>

- 2 - AT_TLS_HOSTNAME.
  - <value> is a string that indicates the hostname to check against during TLS handshakes. It can be NULL to disable hostname verification.
- 4 - AT_TLS_CIPHERSUITE_USED (get-only). The TLS cipher suite is chosen during the TLS handshake.
- 5 - AT_TLS_PEER_VERIFY.
  - <value> is an integer that indicates the peer verification level. It is 0 for none, 1 for optional, or 2 for required.
- 12 - AT_TLS_SESSION_CACHE.
  - <value> is an integer that indicates whether to use TLS session caching. It is 0 for disabled or 1 for enabled.
- 13 - AT_TLS_SESSION_CACHE_PURGE (set-only). Indicates that the TLS session cache must be deleted.
  - <value> can be any integer value.
- 14 - AT_TLS_DTLS_CID (set-only).
  - <value> is an integer that indicates the DTLS connection identifier setting. It can be one of the following values:
    * 0 - TLS_DTLS_CID_DISABLED.
    * 1 - TLS_DTLS_CID_SUPPORTED.
    * 2 - TLS_DTLS_CID_ENABLED.

  See NRF_SO_SEC_DTLS_CID for more details regarding the allowed values.
- 15 - AT_TLS_DTLS_CID_STATUS (get-only). It is the DTLS connection identifier status. It can be retrieved after the DTLS handshake. See NRF_SO_SEC_DTLS_CID_STATUS for more details regarding the returned values.
- 18 - AT_TLS_DTLS_HANDSHAKE_TIMEO.
  - <value> is an integer that indicates the DTLS handshake timeout in seconds. It can be one of the following values: 1, 3, 7, 15, 31, 63, 123.
- 19 - AT_TLS_DTLS_CONN_SAVE (set-only). Write-only socket option to save DTLS connection.
  - <value> can be any integer value, which will be ignored.

  After this option is successfully called, you must call AT_TLS_DTLS_CONN_LOAD before continuing to communicate on the socket.

- 20 - AT_TLS_DTLS_CONN_LOAD (set-only). Write-only socket option to load DTLS connection.
  - <value> can be any integer value, which will be ignored.
- 22 - AT_TLS_DTLS_FRAG_EXT.
  - <value> is an integer that indicates the use of the DTLS fragmentation extension specified in RFC 6066. It can be one of the following values:
    * 0 - DTLS_FRAG_EXT_DISABLED.
    * 1 - DTLS_FRAG_EXT_512_ENABLED.
    * 2 - DTLS_FRAG_EXT_1024_ENABLED.

See nRF socket options for explanation of the supported options.

Example

```
AT#XSSOCKETOPT=0,1,5,2
OK
```

16.5.2 Read command

The read command is not supported.

16.5.3 Test command

The test command tests the existence of the command and provides information about the type of its subparameters.

Syntax

```
AT#XSSOCKETOPT=?
```

Response syntax

```
#XSSOCKETOPT: <handle>,<list of ops>,<name>,<value>
```

Example

```
AT#XSSOCKETOPT=?
#XSSOCKETOPT: <handle>,(0,1),<name>,<value>
OK
```

16.6 Socket binding #XBIND

The #XBIND command allows you to bind a socket with a local port.

You can use this command with TCP and UDP, and it is needed for incoming UDP data, where the remote end targets a particular port.

16.6.1 Set command

The set command allows you to bind a socket with a local port.

Syntax

```
AT#XBIND=<handle>,<port>
```

The parameters and their defined values are the following:

<handle>
Integer. Specifies the socket handle returned from #XSOCKET or #XSSOCKET commands.

<port>
Integer. An unsigned 16-bit integer (0 - 65535) representing the specific port to use for binding the socket. 0 is a special value that indicates that the modem should choose the port automatically.

Example

```
AT#XBIND=0,1234
OK
```

16.6.2 Read command

The read command is not supported.

16.6.3 Test command

The test command is not supported.

16.7 Connection #XCONNECT

The #XCONNECT command allows you to connect to a server and to check the connection status.

This command is for TCP and UDP client sockets.

16.7.1 Set command

The set command allows you to connect to a TCP or UDP server.

Syntax

```
AT#XCONNECT=<handle>,<url>,<port>
```

The parameters and their defined values are the following:

<handle>
Integer. Specifies the socket handle returned from #XSOCKET or #XSSOCKET commands.

<url>
String. The hostname or the IP address of the server. The maximum supported size of the hostname is 128 bytes. When using IP addresses, it supports both IPv4 and IPv6.

<port>
Integer. An unsigned 16-bit integer (0 - 65535) representing the port of the TCP or UDP service on the remote server.

Response syntax

```
#XCONNECT: <handle>,<status>
```

The parameters and their defined values are the following:

<handle>
Integer. The socket handle.

<status>

- `0` - Disconnected.
- `1` - Connected.

Examples

```
AT#XCONNECT=0,"test.server.com",1234
#XCONNECT: 0,1
OK
```

```
AT#XCONNECT=1,"192.168.0.1",1234
#XCONNECT: 1,1
OK
```

```
AT#XCONNECT=2,"2a02:c207:2051:8976::1",4567
#XCONNECT: 2,1
OK
```

16.7.2 Read command

The read command is not supported.

16.7.3 Test command

The test command is not supported.

16.8 Send data #XSEND

The #XSEND command allows you to send data over TCP, UDP, and raw sockets.

16.8.1 Set command

The set command allows you to send data over the connection.

Syntax

```
AT#XSEND=<handle>,<mode>,<flags>,<data> when ``<mode>`` is ``0`` or
``1``

AT#XSEND=<handle>,<mode>,<flags>[,<data_len>] when ``<mode>`` is
``2``
```

The parameters and their defined values are the following:

<handle>
Integer. Specifies the socket handle returned from #XSOCKET or #XSSOCKET commands.

<mode>

- 0 - String mode. Data is provided directly in the command as the <data> parameter.
- 1 - Hex string mode. Data is provided as a hexadecimal string in the <data> parameter.
- 2 - Data mode. nRF91M1 module enters *Data mode* for data input.

<flags>
Sending behavior based on the BSD socket definition.

- 0 - No flags set. The request is complete when the data is pushed to the modem buffer.
- 512 - Blocks send operation until the request is acknowledged by the network. The request will not return until the data is pushed to the network, or acknowledged by the network (for TCP), or the timeout given by the AT_SO_SNDTIMEO socket option is reached. Valid timeout values are 1 to 600 seconds.
- 8192 - Send unsolicited #XSENDNTF notification when the request is acknowledged by the network. Unsolicited notification will be sent when the data is

pushed to the network, or acknowledged by the network (for TCP), or the timeout given by the `AT_SO_SNDTIMEO` socket option is reached. Valid timeout values are 1 to 600 seconds. Further sends for the socket are blocked until the unsolicited notification is received.

This is only supported by the following modem firmware:

- mfw_nrf9151-ntn

<data>
String. Required when `<mode>` is `0` (string mode) or `1` (hex string mode). For string mode (`0`), it is a string that contains the data to be sent. For hex string mode (`1`), it is a hexadecimal string representation of the data to be sent. The maximum payload size in hexadecimal string mode is up to 2800 characters (1400 bytes). Not used when `<mode>` is 2 (data mode).

<data_len>
Integer. Optional, only used when `<mode>` is 2 (data mode). Sets the number of bytes to send in data mode. When the required number of bytes are sent, the data mode is exited.

The termination command +++ is not used in this case.

Note:

UDP packets that exceed the Maximum Transmission Unit (MTU) of any network segment along their path might be dropped or fragmented, increasing the risk of packet loss. Ethernet networks have an MTU of 1500 bytes, which allows a maximum UDP payload of 1472 bytes for IPv4 and 1452 bytes for IPv6. With DTLS sockets, the usable payload size is further reduced due to encryption overhead.

The cellular network MTU can be queried with the `AT+CGCONTRDP` command, but some networks might still drop packets smaller than the reported MTU.

A UDP payload size of 1200 bytes is commonly recommended, especially for IPv6, as it ensures the total packet size remains below the IPv6 minimum MTU of 1280 bytes after accounting for headers and DTLS overhead. Keeping UDP packet sizes well below the theoretical maximum increases the likelihood of successful transmission. You can even use 1024 bytes as a safe size for UDP packets.

Response syntax

```
#XSEND: <handle>,<result_type>,<size>
```

The parameters and their defined values are the following:

<handle>
Integer. The socket handle.

<result_type>

- `0` - Indicates that there are no further notifications.

- `1` - Indicates that an unsolicited notification will be sent when the network acknowledged send is completed.

<size>
Integer. The actual number of bytes that has been sent.

Unsolicited notification

For network acknowledged sends (when the `8192` flag is used), an unsolicited notification is sent when the send operation is completed.

This is only supported by the following modem firmware:

- mfw_nrf9151-ntn

```
#XSENDNTF: <handle>,<status>,<size>
```

The parameters and their defined values are the following:

<handle>
Integer. The socket handle.

<status>
The status of the send acknowledged by the network.

- `0` - Success.
- `-1` - Failure.

<size>
Integer. The size of the data sent.

Example

```
AT#XSEND=0,0,0,"Test TCP"
#XSEND: 0,0,8
OK

AT#XSEND=0,1,0,"48656C6C6F"
#XSEND: 0,0,5
OK

AT#XSEND=1,0,8192,"Test notification"
#XSEND: 1,1,17
OK
#XSENDNTF: 1,0,17

AT#XSEND=1,2,8192
OK
Test datamode with flags+++
#XDATAMODE: 0
#XSENDNTF: 1,0,24
```

16.8.2 Read command

The read command is not supported.

16.8.3 Test command

The test command is not supported.

16.9 Receive data #XRECV

The #XRECV command allows you to receive data over TCP, UDP, and raw sockets.

16.9.1 Set command

The set command allows you to receive data over the connection.

Syntax

```
AT#XRECV=<handle>,<mode>,<flags>,<timeout>[,<data_len>]
```

The parameters and their defined values are the following:

<handle>
Integer. Specifies the socket handle returned from #XSOCKET or #XSSOCKET commands.

<mode>

- `0` - Binary mode. Data is received as binary data.
- `1` - Hex string mode. Data is received as a hexadecimal string representation.

<flags>
Receiving behavior based on the BSD socket definition.

- `0` - No flags set.
- 2 - Read data without removing it from the socket input queue.
- 64 - Override the operation to non-blocking.
- 256 (TCP only) - Block until the full amount of data can be returned.

<timeout>
Integer. The timeout value in seconds. When `0`, it means no timeout, and it makes this request block indefinitely.

<data_len>
Integer. Optional. The maximum number of bytes to receive. The maximum value is 2048 bytes, which is also the default value when the parameter is omitted.

Response syntax

```
#XRECV: <handle>,<mode>,<size>
<data>
```

The parameters and their defined values are the following:

<handle>
Integer. The socket handle.

<mode>
Integer. The receive mode used.

<size>
Integer. The actual number of bytes received. In case of hex string mode, it represents the number of bytes before conversion to hexadecimal format.

<data>
String. The data being received.

Example

```
AT#XRECV=0,0,0,10
#XRECV: 0,0,7
Test OK
OK

AT#XRECV=0,1,0,10
#XRECV: 0,1,5
74205a6f63
OK
```

16.9.2 Read command

The read command is not supported.

16.9.3 Test command

The test command is not supported.

16.10 UDP send data #XSENDTO

The `#XSENDTO` command allows you to send data over UDP.

16.10.1 Set command

The set command allows you to send data over UDP.

Syntax

```
AT#XSENDTO=<handle>,<mode>,<flags>,<url>,<port>,<data> when ``<mode>
`` is ``0`` or ``1``

AT#XSENDTO=<handle>,<mode>,<flags>,<url>,<port>[,<data_len>] when ``
<mode>`` is ``2``
```

The parameters and their defined values are the following:

<handle>
Integer. Specifies the socket handle returned from #XSOCKET or #XSSOCKET commands.

<mode>

- `0` - String mode. Data is provided directly in the command as the `<data>` parameter.
- `1` - Hex string mode. Data is provided as a hexadecimal string in the `<data>` parameter.
- 2 - Data mode. nRF91M1 module enters *Data mode* for data input.

<flags>
Sending behavior based on the BSD socket definition.

- `0` - No flags set. The request is complete when the data is pushed to the modem buffer.
- `512` - Blocks send operation until the request is acknowledged by network. The request will not return until the data is pushed to the network, or acknowledged by network (for TCP), or the timeout given by the AT_SO_SNDTIMEO socket option, is reached. Valid timeout values are 1 to 600 seconds.
- `8192` - Send unsolicited `#XSENDNTF` notification when the request is acknowledged by network. Unsolicited notification will be sent when the data is pushed to the network, or acknowledged by network (for TCP), or the timeout given by the AT_SO_SNDTIMEO socket option, is reached. Valid timeout values are 1 to 600 seconds. Further sends for the socket are blocked until the unsolicited notification is received.

  This is only supported by the following modem firmware:

  - mfw_nrf9151-ntn

<url>
String. The hostname or the IP address of the remote peer. The maximum size of the hostname is 128 bytes. When using IP addresses, it supports both IPv4 and IPv6.

<port>
Integer. An unsigned 16-bit integer (0 - 65535) representing the port of the UDP service on the remote peer.

<data>
String. Required when `<mode>` is `0` (string mode) or `1` (hex string mode). For string mode (`0`), it is a string that contains the data to be sent. For hex string mode (`1`), it is a

hexadecimal string representation of the data to be sent. The maximum payload size in hexadecimal string mode is up to 2800 characters (1400 bytes). For large packets, it is recommended to use data mode (2) since AT parser's memory limits the maximum size of data that can be sent in string or hex string modes. Not used when <mode> is 2 (data mode).

<data_len>
Integer. Optional, only used when <mode> is 2 (data mode). Sets the number of bytes to send in data mode. When the required number of bytes are sent, the data mode is exited.

The termination command +++ is not used in this case.

Note:

UDP packets that exceed the Maximum Transmission Unit (MTU) of any network segment along their path may be dropped or fragmented, increasing the risk of packet loss. Ethernet networks have an MTU of 1500 bytes, which allows a maximum UDP payload of 1472 bytes for IPv4 and 1452 bytes for IPv6. With DTLS sockets, the usable payload size is further reduced due to encryption overhead.

The cellular network MTU can be queried with the AT+CGCONTRDP command, but some networks may still drop packets smaller than the reported MTU.

A UDP payload size of 1200 bytes is commonly recommended, especially for IPv6, as it ensures the total packet size remains below the IPv6 minimum MTU of 1280 bytes, after accounting for headers and DTLS overhead. Keeping UDP packet sizes well below the theoretical maximum increases the likelihood of successful transmission. Even 1024 bytes could be used as a safe size for UDP packets.

Note:

With DTLS connections, the connection cannot be established with the #XSENDTO command. Instead, it must be established using the #XCONNECT command. This is a limitation in the nRF91 modem firmware.

Response syntax

```
#XSENDTO: <handle>,<result_type>,<size>
```

The parameters and their defined values are the following:

<handle>
Integer. The socket handle.

<result_type>

- 0 - Indicates that there are no further notifications.
- 1 - Indicates that an unsolicited notification will be sent when the network acknowledged send is completed.

<size>
Integer. The actual number of bytes that have been sent.

Unsolicited notification

For network acknowledged sends (when the 8192 flag is used), an unsolicited notification is sent when the send operation is completed.

This is only supported by the following modem firmware:

- mfw_nrf9151-ntn

```
#XSENDNTF: <handle>,<status>,<size>
```

The parameters and their defined values are the following:

<handle>
Integer. The socket handle.

<status>
The status of the send acknowledged by the network.

- `0` - Success.
- `-1` - Failure.

<size>
Integer. The size of the data sent.

Example

```
AT#XSENDTO=0,0,0,"test.server.com",1234,"Test UDP"
#XSENDTO: 0,0,8
OK

AT#XSENDTO=0,1,0,"test.server.com",1234,"48656C6C6F"
#XSENDTO: 0,0,5
OK

AT#XSENDTO=0,0,8192,"test.server.com",1234,"Test notification"
#XSENDTO: 0,1,17
OK
#XSENDNTF: 0,0,17
```

16.10.2 Read command

The read command is not supported.

16.10.3 Test command

The test command is not supported.

16.11 UDP receive data #XRECVFROM

The #XRECVFROM command allows you to receive data over UDP.

16.11.1 Set command

The set command allows you to receive data over UDP.

Syntax

```
AT#XRECVFROM=<handle>,<mode>,<flags>,<timeout>[,<data_len>]
```

The parameters and their defined values are the following:

<handle>
Integer. Specifies the socket handle returned from #XSOCKET or #XSSOCKET commands.

<mode>

- 0 - Binary mode. Data is received as binary data.
- 1 - Hex string mode. Data is received as a hexadecimal string representation.

<flags>
Receiving behavior based on the BSD socket definition.

- 0 - No flags set.
- 2 - Read data without removing it from the socket input queue.
- 64 - Override the operation to non-blocking.

<timeout>
Integer. The timeout value in seconds. When 0, it means no timeout, and it makes this request block indefinitely.

<data_len>
Integer. Optional. The maximum number of bytes to receive. The maximum value is 2048 bytes, which is also the default value when the parameter is omitted.

Response syntax

```
#XRECVFROM: <handle>,<mode>,<size>,"<ip_addr>",<port>
<data>
```

The parameters and their defined values are the following:

<handle>
Integer. The socket handle.

<mode>
Integer. The receive mode used.

<size>
Integer. The actual number of bytes received. In the case of hex string mode, it represents the number of bytes before conversion to hexadecimal format.

<ip_addr>
String. The IPv4 or IPv6 address of the remote peer.

<port>
Integer. The UDP port of the remote peer.

<data>
String. The data being received.

Example

```
AT#XRECVFROM=0,0,0,10
#XRECVFROM: 0,0,7,"192.168.1.100",24210
Test OK
OK

AT#XRECVFROM=0,1,0,10
#XRECVFROM: 0,1,7,"192.168.1.100",24210
54657374205a4d
OK
```

16.11.2 Read command

The read command is not supported.

16.11.3 Test command

The test command is not supported.

16.12 Asynchronous socket polling #XAPOLL

The #XAPOLL command allows you to receive Unsolicited Result Code (URC) notifications for poll events on sockets.

16.12.1 Set command

The set command allows you to activate or deactivate asynchronous polling for sockets.

Syntax

```
AT#XAPOLL=[<handle>],<op>,[<events>]
```

The parameters and their defined values are the following:

<handle>
Integer. Optional. Sets the socket handle to poll. Handles are sent in the AT#XSOCKET or AT#XSSOCKET responses. Handles can also be obtained using the AT#XSOCKET? or AT#XSSOCKET? command. If the handle is omitted, the operation applies to all open sockets and to any new sockets that are created.

<op>

- 0 - Stop asynchronous polling.
- 1 - Start asynchronous polling.

<events>
Integer. Optional. Interpreted as a bit field representing the events to poll for. Permanent error and closure events (POLLERR, POLLHUP, and POLLNVAL) are always polled. The value can be any combination of the following values summed up:

- 0 - Poll the default events.
- 1 - Read events (POLLIN) are polled, in addition to the default events.
- 4 - Write events (POLLOUT) are polled, in addition to the default events.

Response syntax

When the asynchronous socket events are enabled, nRF91M1 module sends events as URC notifications.

- For POLLIN events, the URC notification is sent only for the first incoming data on the socket. AT#XRECV or AT#XRECVFROM command will re-enable the URC notification for the next incoming data.
- For POLLOUT events, the URC notification is sent only for the first time when the socket is ready for writing. AT#XSEND or AT#XSENDTO command will re-enable the URC notification for the next time when the socket is ready for writing.
- For POLLERR, POLLHUP, and POLLNVAL events, the URC notification is sent only once for each socket. No further URC notifications will be sent for the same socket.

Note:

When closing the socket with #XCLOSE, no closure event will be sent.

```
#XAPOLL: <handle>,<revents>
```

The parameters and their defined values are the following:

<handle>
Integer. The handle of the socket that has events.

<revents>
Integer. Interpreted as a bit field representing the returned events as a combination of POLLIN (1), POLLOUT (4), POLLERR (8), POLLHUP (16), and POLLNVAL (32) summed up. Hexadecimal representation is avoided to support AT command parsers that do not support hexadecimal values.

Example

```
// Start asynchronous polling for all sockets with POLLIN and
POLLOUT events.
AT#XAPOLL=,1,5

OK

// Create a TCP socket and connect to the test server.
AT#XSOCKET=1,1,0

#XSOCKET: 0,1,6

OK

AT#XCONNECT=0,"test.server.com",1234

#XCONNECT: 0,1

OK

#XAPOLL: 0,4

// Send data to the test server, which will echo it back.
AT#XSEND=0,0,0,"echo"

#XSEND: 0,0,4

OK

#XAPOLL: 0,4

// Test server sends the data back and closes the connection. POLLIN
and POLLHUP events are received.
#XAPOLL: 0,17

AT#XRECV=0,0,0,1

#XRECV: 0,0,4
echo
OK

// Close the TCP socket.
AT#XCLOSE=0

#XCLOSE: 0,0

OK

// Create a UDP socket and send network acknowledge data to another
```

(continues on next page)

(continued from previous page)

```
test server, which does not echo.
AT#XSOCKET=2,2,0

#XSOCKET: 0,2,17

OK

#XAPOLL: 0,4

AT#XCONNECT=0,"no_echo.test.server.com",1234

#XCONNECT: 0,1

OK

// Send data to the test server with network acknowledged send flag.
AT#XSEND=0,0,8192,"no echo"

#XSEND: 0,1,7

OK

// Unsolicited notification for network acknowledged send.
#XSENDNTF: 0,0,7

#XAPOLL: 0,4

// Close the UDP socket.
AT#XCLOSE=0

#XCLOSE: 0,0

OK

// Stop asynchronous polling for all sockets.
AT#XAPOLL=,0

OK
```

16.12.2 Read command

The read command lists the socket handles with the events that are being polled.

Syntax

```
AT#XAPOLL?
```

Response syntax

```
#XAPOLL: <handle>,<events>
```

The parameters and their defined values are the following:

<handle>
Integer. The handle of the socket that is being polled.

<events>
Integer. Interpreted as a bit field representing the events that are being polled. Permanent error and closure events (`POLLERR`, `POLLHUP`, and `POLLNVAL`) are always polled. The value can be any combination of the following values:

- `0` - Poll the default events.
- `1` - Poll read events (`POLLIN`) in addition to the default events.
- `4` - Poll write events (`POLLOUT`) in addition to the default events.

Example

```
// Start asynchronous polling for all sockets with POLLIN event.
AT#XAPOLL=,1,1

OK

// Create socket 0.
AT#XSOCKET=1,1,0

#XSOCKET: 0,1,6

OK

// Create socket 1 to show that poll events can be configured per
socket.
AT#XSOCKET=1,1,0

#XSOCKET: 1,1,6

OK

// Activate only POLLOUT event polling for socket 1.
AT#XAPOLL=1,1,4

OK

// Create socket 2 to show that POLLIN for all sockets is still in
effect.
AT#XSOCKET=1,1,0
```

(continues on next page)

(continued from previous page)

```
#XSOCKET: 2,1,6

OK

// Read the current poll settings.
AT#XAPOLL?

#XAPOLL: 0,1

#XAPOLL: 1,4

#XAPOLL: 2,1

OK

// Stop asynchronous polling for all sockets.
AT#XAPOLL=,0

OK

// Close all sockets.
AT#XCLOSE

#XCLOSE: 0,0

#XCLOSE: 1,0

#XCLOSE: 2,0

OK
```

16.12.3 Test command

The test command provides information about the command and its parameters.

Syntax

```
AT#XAPOLL=?
```

Response syntax

```
#XAPOLL: <handle>,(stop/start),(events)
```

Example

```
AT#XAPOLL=?

#XAPOLL: <handle>,(0,1),(0,1,4,5)

OK
```

16.13 Configure socket receive #XRECVCFG

The socket receive configuration command allows you to configure the following aspects of a socket:

- Automatic data reception
- Automatic data reception in hex format

16.13.1 Set command

The set command allows you to configure the socket receive configuration of all sockets or a specific socket.

Syntax

```
AT#XRECVCFG=[<handle>],<auto_reception_flags>[,<hex_format>]
```

The parameters and their defined values are the following:

<handle>
Integer. Identifies the socket handle. If omitted, the command applies to all opened sockets, whether already open or opened in the future.

<auto_reception_flags>
Integer. Specifies the automatic reception flags. Can be a combination of the following values summed up:

- `0` - No automatic data reception.
- `1` - Automatic data reception in AT-command mode.
- `2` - Automatic data reception in data mode.

<hex_format>
Integer. Specifies the hex format for automatically received data. Applies only when automatic data reception is enabled.

- `0` - Data is received in binary format (default).
- `1` - Data is received in hex string format (supported only in AT-command mode).

Response syntax

When in the AT-command mode and the automatic data reception is enabled, nRF91M1 module sends #XRECV and #XRECVFROM responses as URC notifications when data is received. #XRECV is used for TCP, raw, and connected UDP sockets, while #XRECVFROM is used for unconnected UDP sockets.

```
#XRECV: <handle>,<mode>,<size>
<data>
```

The parameters and their defined values are the following:

<handle>
Integer. The socket handle.

<mode>
Integer. The receive mode used.

<size>
Integer. The actual number of bytes received. In case of hex string mode, it represents the number of bytes before conversion to hexadecimal format.

<data>
String. The data being received.

```
#XRECVFROM: <handle>,<mode>,<size>,"<ip_addr>",<port>
<data>
```

The parameters and their defined values are the following:

<handle>
Integer. The socket handle.

<mode>
Integer. The receive mode used.

<size>
Integer. The actual number of bytes received. In the case of hex string mode, it represents the number of bytes before conversion to hexadecimal format.

<ip_addr>
String. The IPv4 or IPv6 address of the remote peer.

<port>
Integer. The UDP port of the remote peer.

<data>
String. The data being received.

Note:

<CR><LF> is sent after <data>. This differs from the standard #XRECV and #XRECVFROM responses where <CR><LF> is sent before OK.

When in data mode and the automatic data reception is enabled, nRF91M1 module sends the received data as is, without any additional headers or formatting.

Example

```
// Enable automatic data reception in AT-command mode in binary
format for all sockets.
AT#XRECVCFG=,1,0

OK

AT#XSOCKET=1,1,0

#XSOCKET: 0,1,6

OK

AT#XCONNECT=0,"test.server.com",1234

#XCONNECT: 0,1

OK

// Send data to the test server, which will echo it back.
AT#XSEND=0,0,0,"Test"

#XSEND: 0,0,4

OK

// Data is automatically received.
#XRECV: 0,0,4
Test

// Enable automatic data reception in AT-command mode in hex string
format for socket 0.
AT#XRECVCFG=0,1,1

OK

// Send data to the test server, which will echo it back.
AT#XSEND=0,0,0,"Test hex"

#XSEND: 0,0,8

OK

// Data is automatically received in hex string format.
#XRECV: 0,1,8
```

(continues on next page)

(continued from previous page)

```
5465737420686578

// Enable automatic reception of data in data mode.
AT#XRECVCFG=0,2,0

OK

// Enter data mode and send data to the test server, which will echo
it back.
AT#XSEND=0,2,0

OK
DATA TEST
DATA TEST
+++
#XDATAMODE: 0

// Create a new UDP socket. Data reception in AT-command mode in
binary format is enabled for it.
AT#XSOCKET=1,2,0

#XSOCKET: 1,2,17

OK

// Send data with unconnected UDP socket to the test server, which
will echo it back with a delay.
AT#XSENDTO=1,0,0,"test.server.com",1235,"Delayed UDP data"

#XSENDTO: 1,0,16

OK

// Enter data mode with socket 0.
AT#XSEND=0,2,0

OK

// Long operations during data mode.

// Exiting the data mode allows the delayed UDP data to be received.
+++
#XDATAMODE: 0

// Unconnected UDP socket automatically receives the delayed data
with RECVFROM.
#XRECVFROM: 1,0,16,"111.112.113.114",1235
Delayed UDP data
```

(continues on next page)

(continued from previous page)

```
// Disable automatic data reception for all sockets.
AT#XRECVCFG=,0

OK

AT#XCLOSE

#XCLOSE: 0,0

#XCLOSE: 1,0

OK
```

16.13.2 Read command

The read command allows you to check the receive configuration settings of sockets.

Syntax

```
AT#XRECVCFG?
```

Response syntax

```
#XRECVCFG: <handle>,<auto_reception_flags>,<hex_format>
```

The parameters and their defined values are the following:

<handle>
Integer. Identifies the socket handle.

<auto_reception_flags>
Integer. Specifies the automatic reception flags. A combination of the following values summed up:

- `0` - No automatic data reception.
- `1` - Automatic data reception in AT-command mode.
- `2` - Automatic data reception in data mode.

<hex_format>
Integer. Specifies the hex format for automatically received data.

- `0` - Data is received in binary format.
- `1` - Data is received in hex string format (supported only in AT-command mode).

Example

```
AT#XRECVCFG=,1,0

OK
AT#XSOCKET=1,1,0

#XSOCKET: 0,1,6

OK
AT#XSOCKET=1,1,0

#XSOCKET: 1,1,6

OK
// Enable automatic data reception in data mode for socket 1.
AT#XRECVCFG=1,2

OK
AT#XRECVCFG?

#XRECVCFG: 0,1,0

#XRECVCFG: 1,2,0

OK
// Disable automatic reception for all sockets.
AT#XRECVCFG=,0

OK
AT#XRECVCFG?

OK
```

16.13.3 Test command

The test command provides information about the command and its parameters.

Syntax

```
AT#XRECVCFG=?
```

Response syntax

```
#XRECVCFG: <handle>,(0,1,2,3),(0,1)
```

Example

```
AT#XRECVCFG=?

#XRECVCFG: <handle>,(0,1,2,3),(0,1)

OK
```

16.14 Resolve hostname #XGETADDRINFO

The `#XGETADDRINFO` command allows you to resolve hostnames to IPv4 and IPv6 addresses.

16.14.1 Set command

The set command allows you to resolve hostnames to IPv4 and IPv6 addresses.

Syntax

```
AT#XGETADDRINFO=<hostname>[,<address_family>]
```

The parameters and their defined values are the following:

<hostname>
String. The hostname to resolve.

<address_family>
Integer. Optional. Gives a hint for DNS query on address family.

- `0` - Unspecified address family.
- `1` - IPv4 address family.
- `2` - IPv6 address family.

If not specified, there will be no hint given for the DNS query.

Response syntax

```
#XGETADDRINFO: "<ip_addresses>"
```

The parameters and their defined values are the following:

<ip_addresses>
String. The IPv4 or IPv6 address of the resolved hostname.

Example

```
AT#XGETADDRINFO="google.com"
#XGETADDRINFO: "142.251.42.142"
OK
AT#XGETADDRINFO="google.com",0
#XGETADDRINFO: "172.217.31.142"
OK
AT#XGETADDRINFO="google.com",1
#XGETADDRINFO: "142.251.42.142"
OK
AT#XGETADDRINFO="ipv6.google.com",2
#XGETADDRINFO: "2404:6800:4004:824::200e"
OK
```

16.14.2 Read command

The read command is not supported.

16.14.3 Test command

The test command is not supported.

16.15 Socket listen #XLISTEN

The #XLISTEN command allows you to put a TCP server socket in listening mode to accept incoming connections.

The socket must be a stream socket (TCP) that has been bound to a non-zero local port using the #XBIND command before it can be put in listening mode.

Note:

The nRF91 modem does not support TLS server sockets, so you cannot use the #XLISTEN command with TLS sockets created with the #XSOCKET command.

16.15.1 Set command

The set command allows you to put a socket in listening mode.

Syntax

```
AT#XLISTEN=<handle>
```

The parameters and their defined values are the following:

<handle>
Integer. Specifies the socket handle returned from #XSOCKET command. The socket

must be a stream socket (TCP) that has been bound to a non-zero local port with #XBIND.

Example

```
AT#XLISTEN=0
OK
```

16.15.2 Read command

The read command allows you to list all sockets that are in listening mode.

Syntax

```
AT#XLISTEN?
```

Response syntax

```
#XLISTEN: <handle>,<cid>,<local_port>
```

The parameters and their defined values are the following:

<handle>
Integer. The socket handle.

<cid>
Integer. The PDN connection ID.

<local_port>
Integer. An unsigned 16-bit integer (0 - 65535) representing the local port the socket is bound to and listening on.

Example

```
AT#XLISTEN?
#XLISTEN: 0,0,1000

OK
```

16.15.3 Test command

The test command tests the existence of the command and provides information about the type of its subparameters.

Syntax

```
AT#XLISTEN=?
```

Response syntax

```
#XLISTEN: <handle>
```

Example

```
AT#XLISTEN=?
#XLISTEN: <handle>
OK
```

16.16 Socket accept #XACCEPT

The #XACCEPT command allows you to accept an incoming connection on a listening socket.

This command is used with TCP server sockets that have been put in listening mode with the #XLISTEN command. When a connection is accepted, a new socket is created to handle the connection, and the original listening socket continues to listen for additional connections.

Note:

The listening socket is set to non-blocking mode when the #XLISTEN command is executed. If there are no incoming connections when the #XACCEPT command is executed, it returns an error. You can use the #XAPOLL command to receive asynchronous notifications (POLLIN) for incoming connections on the listening socket.

16.16.1 Set command

The set command allows you to accept an incoming connection on a listening socket.

Syntax

```
AT#XACCEPT=<handle>
```

The parameters and their defined values are the following:

<handle>
Integer. Specifies the socket handle that was used for the AT#XLISTEN command.

Response syntax

```
#XACCEPT: <handle>,<cid>,"<peer_addr>",<peer_port>
```

The parameters and their defined values are the following:

<handle>
Integer. The new socket handle created for the accepted connection. You can use this new socket to send and receive data with the connected client.

<cid>
Integer. The PDN connection ID.

<peer_addr>
String. The IP address of the remote peer that connected. Supports both IPv4 and IPv6 addresses.

<peer_port>
Integer. An unsigned 16-bit integer (0 - 65535) representing the port number of the remote peer.

Example

```
AT#XSOCKET=2,1,0
#XSOCKET: 0,1,6

OK

// Enable asynchronous polling for all the sockets with POLLIN
event.
AT#XAPOLL=,1,1
OK

AT#XBIND=0,1000
OK

AT#XLISTEN=0
OK

// POLLIN for listening socket is received when a client connects.
#XAPOLL: 0,1

// Accept the incoming connection.
AT#XACCEPT=0
#XACCEPT: 1,0,"1111:2222:3333:4444::1",1234

OK

// Once the connection is accepted, data can be received and sent
with the new socket handle.
```

(continues on next page)

(continued from previous page)

```
#XAPOLL: 1,1

AT#XRECV=1,0,0,1
#XRECV: 1,0,17
Hello from client
OK

AT#XSEND=1,0,0,"Hello from server"
#XSEND: 1,0,17

OK

#XAPOLL: 1,1

AT#XRECV=1,0,0,1
#XRECV: 1,0,15
Bye from client
OK

// Client closes the connection. POLLIN event is always received
with POLLHUP, if POLLIN is enabled for polling.
#XAPOLL: 1,17
AT#XRECV=1,0,0,1
OK

AT#XCLOSE=1
#XCLOSE: 1,0

OK

AT#XCLOSE
#XCLOSE: 0,0

OK
```

16.16.2 Read command

The read command is not supported.

16.16.3 Test command

The test command tests the existence of the command and provides information about the type of its subparameters.

Syntax

```
AT#XACCEPT=?
```

Response syntax

```
#XACCEPT: <handle>
```

Example

```
AT#XACCEPT=?
#XACCEPT: <handle>
OK
```

17 Trace AT commands

This page describes the AT commands for controlling the shared UART trace backend.

The Zephyr application log backend (AT#XLOG) and the modem trace backend (AT#XTRACE) share a single UART. They are mutually exclusive: enabling one while the other is active returns an error. During boot, the UART outputs logs from B0, MCUboot, and the application. After initialization, the UART is suspended when no backend is active and resumed when at least one backend is enabled.

The trace UART is configured at 1000000 baud rate to support the high data rate required for the modem traces. Trace data is available through UART1.

17.1 Application log AT#XLOG

The AT#XLOG command enables or disables the Zephyr application log backend and the shared UART.

Note:

Regardless of the logging mode, nRF Cloud Observability collects information, warning, and error level logs (with sensitive payloads redacted) on crash, even when the UART is silent. You can only view these logs once uploaded to nRF Cloud. See *Observability #XNRFCLOUDOBS**.

17.1.1 Set command

The set command enables or disables the Zephyr application log backend.

Syntax

```
AT#XLOG=<mode>
```

The parameters and their defined values are the following:

<mode>

- 0 - Suspend the UART and disable the application log backend, no logs are shown. Logs are still collected by nRF Cloud Observability.

- 1 - Resume the UART and enable the application log backend. AT commands, responses, and URCs are logged as strings at information log level, with sensitive payloads redacted.
- 2 - Resume the UART and enable the application log backend. AT commands, responses, and URCs are logged as raw hex dumps at debug level only. These hex dumps are not collected by nRF Cloud Observability.

Note:

Returns ERROR if AT#XTRACE=1 has been issued.

17.1.2 Read command

The read command returns the current state of the application log backend.

Syntax

```
AT#XLOG?
```

Response syntax

```
#XLOG: <mode>
```

The parameters and their defined values are the following:

<mode>
The current state.

- 0 - Disabled.
- 1 - Logging with sensitive AT command payloads redacted.
- 2 - Logging with sensitive AT command and response payloads logged as DBG hex dump only.

17.1.3 Test command

The test command returns the supported parameter range.

Syntax

```
AT#XLOG=?
```

Response syntax

```
#XLOG: (0,1,2)
```

Example

```
AT#XLOG=1
OK

AT#XLOG?
#XLOG: 1
OK

AT#XLOG=0
OK
```

17.2 Modem trace AT#XTRACE

The AT#XTRACE command enables or disables the modem trace backend on the shared UART.

When enabled, the UART is resumed and the modem is instructed to generate full-level traces (AT%XMODEMTRACE=1,2). When disabled, the modem is instructed to stop generating traces (AT%XMODEMTRACE=0) and the UART is suspended.

17.2.1 Set command

The set command enables or disables the modem trace backend.

Syntax

```
AT#XTRACE=<mode>
```

The parameters and their defined values are the following:

<mode>

- 0 - Disable modem traces and suspend the UART.
- 1 - Resume the UART and enable the modem trace backend.

Note:

Returns ERROR if AT#XLOG=1 has been issued.

17.2.2 Read command

The read command returns the current state of the modem trace backend.

Syntax

```
AT#XTRACE?
```

Response syntax

```
#XTRACE: <mode>
```

The parameters and their defined values are the following:

<mode>
The current state.

- `0` - Disabled.
- `1` - Enabled.

17.2.3 Test command

The test command returns the supported parameter range.

Syntax

```
AT#XTRACE=?
```

Response syntax

```
#XTRACE: (0,1)
```

Example

```
AT#XTRACE=1
OK

AT#XTRACE?
#XTRACE: 1
OK

AT#XTRACE=0
OK
```

18 Data mode

The nRF91M1 module can run in the two operation modes defined by the AT command set: AT-command mode and data mode.

When running in data mode, the application does the following:

- It considers all the data received from the MCU over the UART bus as arbitrary data to be streamed through the LTE network by various service modules.
- It buffers the URCs received from modem and threads and sends them to the MCU only after exiting data mode.
- For the socket that is in data mode with automatic data reception, the data streamed from a remote service is considered binary data to be sent to the MCU over the UART.

18.1 Overview

You can manually switch between AT-command mode and data mode. However, the nRF91M1 module data mode is applied automatically when any of the following AT commands are issued:

- Socket `AT#XSEND` and `AT#XSENDTO`
- MQTT publish `AT#XMQTTPUB`
- nRF Cloud send message `AT#XNRFCLOUD`
- DFU write `AT#XDFUWRITE`

18.1.1 Entering data mode

The nRF91M1 module enters data mode when an AT command to send data out does not carry the payload. See the following examples:

- `AT#XSEND=0,2,0` makes nRF91M1 module enter data mode to receive arbitrary data to transmit for socket 0.
- `AT#XSEND=0,2,0,<data_len>` makes nRF91M1 module enter data mode to receive arbitrary data of the specified length to transmit for socket 0.
- `AT#XSEND=0,0,0,"data"` makes nRF91M1 module transmit data in normal AT Command mode for socket 0.

Note:

If the data contains either , or " as characters, it can only be sent in data mode. A typical use case is to send JSON messages.

Other examples:

- `AT#XMQTTPUB=<topic>,"",<qos>,<retain>`
- `AT#XNRFCLOUD=2`
- `AT#XDFUWRITE=0,0,4096`

The nRF91M1 module sends an *OK* response when it successfully enters data mode.

18.1.2 Sending data in data mode

Any arbitrary data received from the MCU is sent to LTE network *as-is*.

Note:

If sending fails while in data mode, the nRF91M1 module drops any further data received from the UART until data mode is exited, as described in the *Exiting data mode* section. When data mode was entered with `<data_len>`, no termination string is sent. The MCU receives `#XDATAMODE` at the expected byte boundary instead.

When the data mode was entered without `<data_len>`, the nRF91M1 module also sends the termination string +++ to the MCU to signal the error.

18.1.3 Exiting data mode

To exit the data mode without the specification of `<data_len>`, the MCU sends the termination command +++ over UART.

The pattern string could be sent alone or as an affix to the data. The pattern string must be sent in full.

If `<data_len>` is specified in the AT command and the specified data length is reached, the nRF91M1 module exits data mode. The termination command is not used in this case.

If a send operation fails while in data mode, such as when the socket in data mode encounters an error, the same exit conditions apply. The MCU must send the termination command if `<data_len>` was not specified. If `<data_len>` was specified, the MCU must send the remaining bytes required to reach the specified length.

When exiting the data mode, the nRF91M1 module sends the `#XDATAMODE` unsolicited notification.

After exiting the data mode, the nRF91M1 module returns to the AT command mode.

18.1.4 Triggering the transmission

The nRF91M1 module buffers all the arbitrary data received from the UART bus before initiating the transmission.

The transmission of the buffered data to the LTE network is triggered in the following scenarios:

- Time limit when the defined inactivity timer times out.
- Reception of the termination string.
- Filling of the data mode buffer.

If there is no time limit configured, the minimum required value applies. For more information, see the *Data mode control #XDATACTRL* command.

18.1.5 Flow control in data mode

When nRF91M1 module fills its UART receive buffers, it disables UART reception. If `hw-flow-control` is enabled for the UART, hardware flow control is imposed. Without hardware flow control, the nRF91M1 module will drop incoming data while the UART reception is disabled. nRF91M1 module reenables UART reception when the data has been moved to the data mode buffer. If the data mode buffer fills, the data are transmitted to the LTE network.

Note:

There is no unsolicited notification defined for this event. UART hardware flow control is responsible for imposing and revoking flow control.

Note:

The whole buffer is sent in a single operation. When transmitting UDP packets, only one complete packet must reside in the data mode buffer at any time.

The data mode buffer size is 4096 bytes.

18.2 Data mode control #XDATACTRL

The `#XDATACTRL` command allows you to configure the time limit used to trigger data transmissions. It can be applied only after entering data mode.

When the time limit is configured, small-size packets will be sent only after the timeout.

18.2.1 Set command

The set command allows you to configure the time limit for the data mode.

Syntax

```
AT#XDATACTRL=<time_limit>
```

The parameters and their defined values are the following:

<time_limit>
Integer. The timeout value in milliseconds. The default value is the minimum required value, based on the configured UART baud rate.

This value must be long enough to allow for a DMA transmission of an UART receive (RX) buffer (532 bytes).

18.2.2 Read command

The read command allows you to check the current time limit configuration and the minimum value required, based on the configured UART baud rate.

Syntax

```
AT#XDATACTRL?
```

Response syntax

```
#XDATACTRL: <current_time_limit>,<minimal_time_limit>
```

18.2.3 Test command

The test command tests the existence of the command and provides information about the type of its subparameters.

Syntax

```
AT#XDATACTRL=?
```

Response syntax

```
#XDATACTRL=<time_limit>
```

18.3 Exit data mode notification #XDATAMODE

When the application receives the termination command +++ in data mode, it sends the #XDATAMODE unsolicited notification.

18.3.1 Unsolicited notification

```
#XDATAMODE: <status>
```

The parameters and their defined values are the following:

<status>

- `0` - Success.
- `-1` - Failure.

Example

```
AT#XSEND=0,2,0

OK
Test datamode
+++
#XDATAMODE: 0
```

Appendix A: UART configuration

The nRF91M1 module is a pre-programmed nRF9151 SiP that runs the Serial Modem application out of the box.

A.1 UART interface

The nRF91M1 module is controlled by a host processor through a UART interface. The host sends AT commands and receives responses and notifications over this UART. When CMUX is enabled, the same UART also carries PPP data frames for network communication.

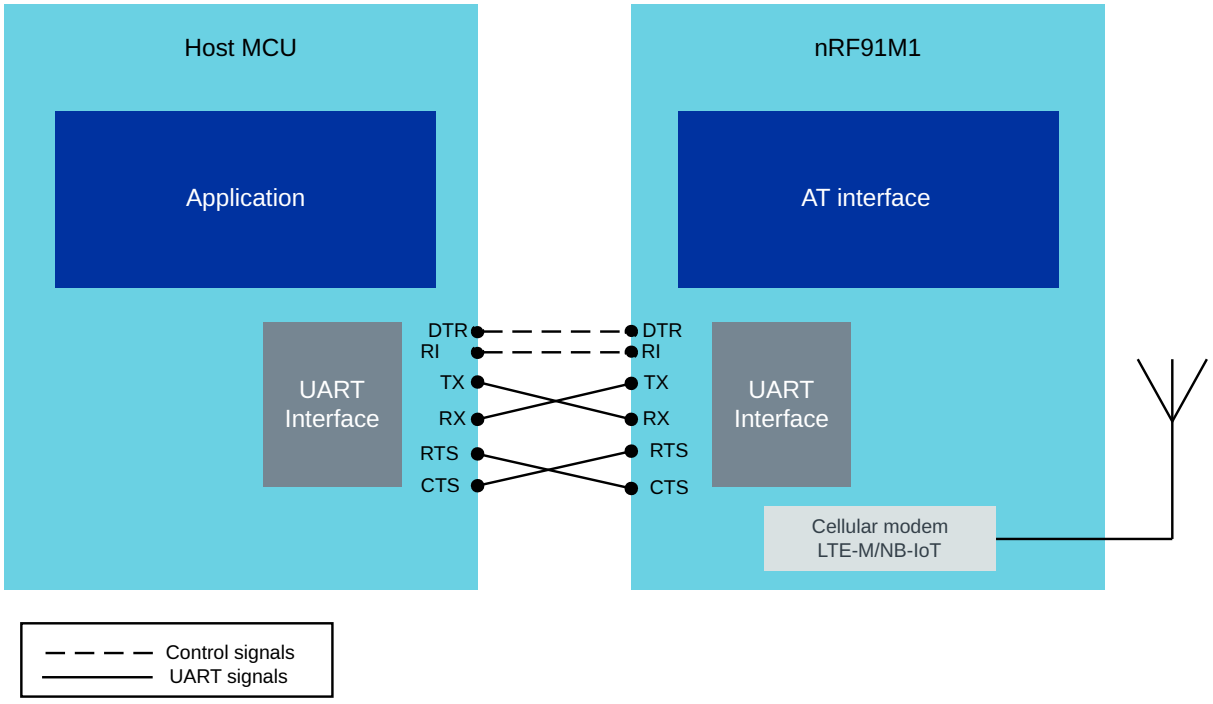


Fig. 1: UART interface setup

The following UART signals are used:

Signal	Direction	Purpose
TX/RX	Bidirectional (crossed)	Data transmission.
RTS/CTS	Bidirectional (crossed)	Hardware flow control.
DTR	Host to nRF91M1	Data Terminal Ready. Host signals UART is powered on.
RI	nRF91M1 to Host	Ring Indicator. The nRF91M1 wakes host for incoming data.

The Data Terminal Ready (DTR) and Ring Indicator (RI) signals are the control signals for UART power management, while TX/RX/RTS/CTS are the standard UART signals.

Important:

The nRF91M1 module has hardware flow control (RTS and CTS) permanently enabled. Using the nRF91M1 module without connecting RTS and CTS to the host is experimental and strongly not recommended. See *Operating without hardware flow control* for more information.

A.2 nRF91M1 module pin assignment

The following table shows how to connect the UART pins to the corresponding pins on the nRF91M1 module (nRF9151 LGA pin assignments):

Signal	nRF91M1 module pin
UART0 TX	45
UART0 RX	44 (pull-up)
UART0 RTS	74
UART0 CTS	75 (pull-up)
UART0 DTR	50 (active low, pull-up)
UART0 RI	49 (active low)
UART1 TX	48
UART1 RX	47 (pull-up, not connected to the host)

AT UART:

- UART instance: UART0
- Baud rate: 115200
- Hardware flow control: Enabled

Log / Trace UART:

- UART instance: UART1
- Baud rate: 1000000
- Hardware flow control: Disabled

Important:

Operating the nRF91M1 module without hardware flow control is strongly not recommended. See *Operating without hardware flow control* for more information.

A.2.1 Power and reset pins

The following pins supply, power on, and reset the nRF91M1 module:

Signal	nRF91M1 module pin
ENABLE	10 (active high)
nRESET	9 (active low, external pull-up not allowed)
VDD	14
VDD_GPIO	65

The nRF91M1 module firmware has no watchdog. The host must therefore be able to reset the nRF91M1 module to a known state, either by power cycling it with **ENABLE** or by using **nRESET**.

Note:

The pins listed here are not the only pins that can be useful with the nRF91M1 module. For the complete pin list, see the nRF9151 LGA pin assignments. For hardware integration guidance, see the nRF9151 Hardware Integration page.

A.3 Controlling the UART power state

The UART power state of the nRF91M1 module is controlled by the **DTR** signal. When DTR is asserted by the host, the UART is enabled and ready for communication. When DTR is deasserted, the UART is powered down to save power.

When the UART is powered down, it releases its high-speed clocks, which are then automatically stopped by the system's power management. This achieves the lowest possible power consumption, as the UART is one of the primary power consumers in nRF91M1 module. Power consumption can be reduced from hundreds of microamps to just a few microamps when the UART is powered down.

UART power management is completely independent of the modem power management (CFUN, PSM, eDRX, and so on). By actively managing the UART power state through DTR control, the nRF91M1 module achieves ultra-low power consumption of approximately 2 µA when the cellular modem enters Power Saving Mode (PSM) and the UART is powered down. This is a reduction of over 99% compared to the 400-450 µA consumed when the UART remains active during idle periods. The nRF91M1 module retains all the RAM content while in any of the power saving modes, so the application can resume from where it left off when the UART is powered up. Full network context is maintained, including socket connections, TLS/DTLS sessions, and so on. LTE connection is automatically resumed when there is outgoing data to be sent. No other GPIO control is required to utilize the PSM or eDRX modes.

A.3.1 Incoming data wake-up

When the UART is powered down and there is incoming data, the nRF91M1 module asserts the **RI** pin to notify the host. The host must handle the RI signal by powering up its UART device and asserting the DTR line, signaling to the nRF91M1 module that the UART is powered on and ready to receive data. The RI signal remains asserted until the UART is activated and ready to transmit data.

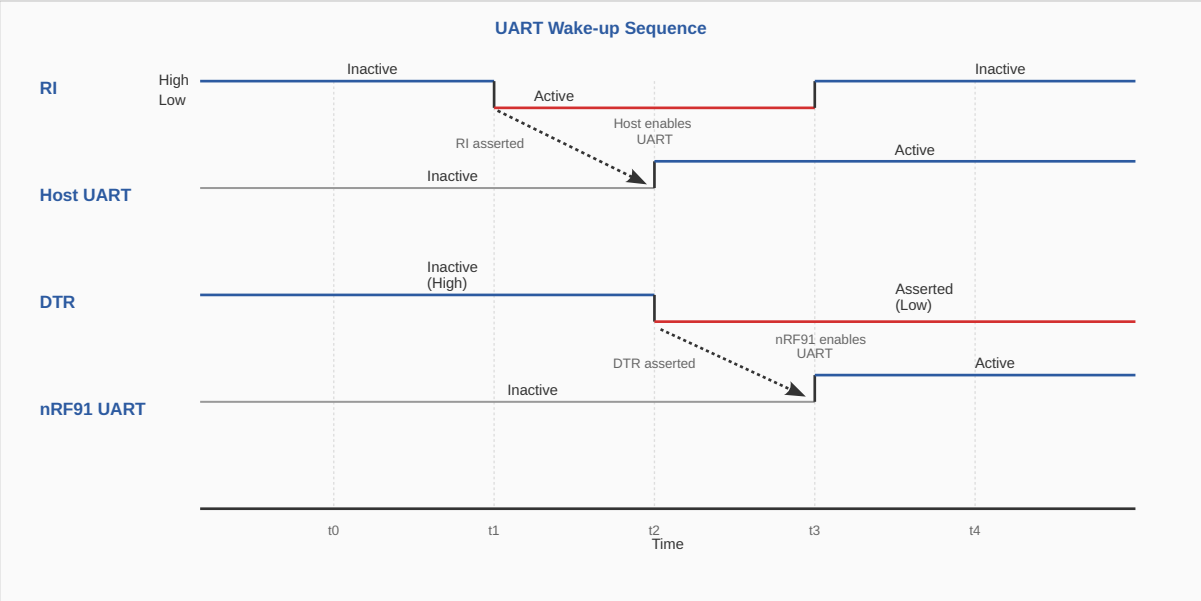


Fig. 2: Sequence diagram showing the wake-up sequence when the UART is powered down

This sequence diagram illustrates the wake-up sequence when the UART is powered down. The RI signal triggers the host to power up its UART and assert DTR. Once the host UART is ready and DTR is asserted, the nRF91M1 module enables its UART interface. The RI signal remains active throughout this process and only returns to inactive once the nRF91M1 module UART is active. This coordinated wake-up ensures both ends of the communication link are ready before data transmission begins.

With hardware flow control - The host can observe its CTS pin (connected to nRF91M1 module's RTS) to know when nRF91M1 module is ready to receive data. When nRF91M1 module starts driving its RTS pin, the host knows that nRF91M1 module is ready to receive. The host must also have a pull-up on its CTS pin.

Without hardware flow control- The host must monitor the RI signal, which is deasserted when the nRF91M1 module UART becomes ready to receive data. Alternatively, the host can implement a sufficient timeout between asserting DTR and initiating its first TX operation. See *Operating without hardware flow control* for how to configure nRF91M1 module to transmit without hardware flow control.

A.3.2 Host-initiated wake-up

When the host processor needs to send data, it first powers up the UART device, then asserts the DTR signal to indicate that the UART is active. The modem detects the DTR assertion and powers up its UART interface, allowing communication to resume.

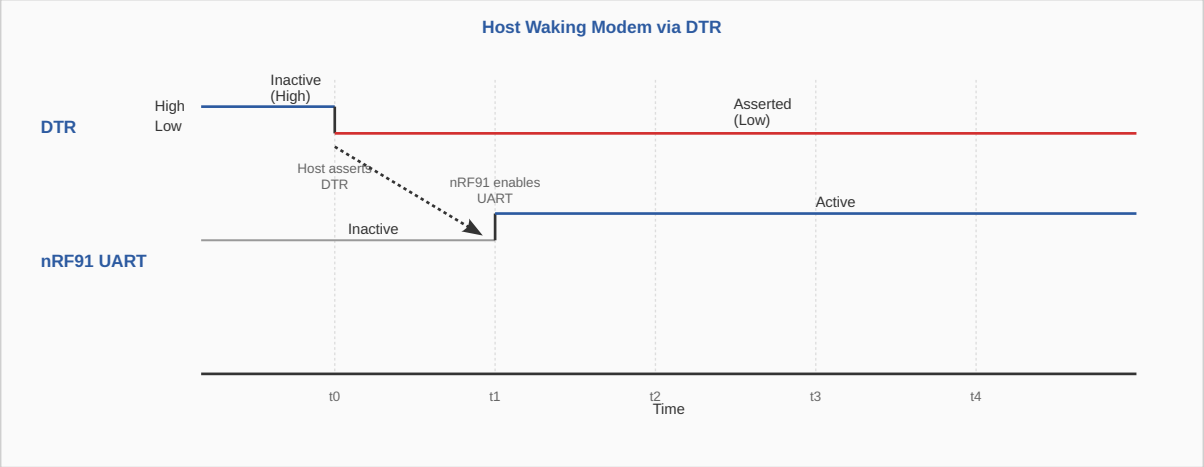


Fig. 3: Sequence diagram showing host-initiated wake-up of the modem

A.3.3 Boot-up scenario

To boot-up the nRF91M1 module, the host asserts **ENABLE**, **DTR**, and enables its UART (driving nRF91M1 module **CTS** low). The nRF91M1 module firmware initializes within approximately 2000 ms, during which it activates its UART and asserts **RTS** to signal readiness. nRF91M1 module then transmits a `"Ready"` message on **TX**, confirming that the link is operational.

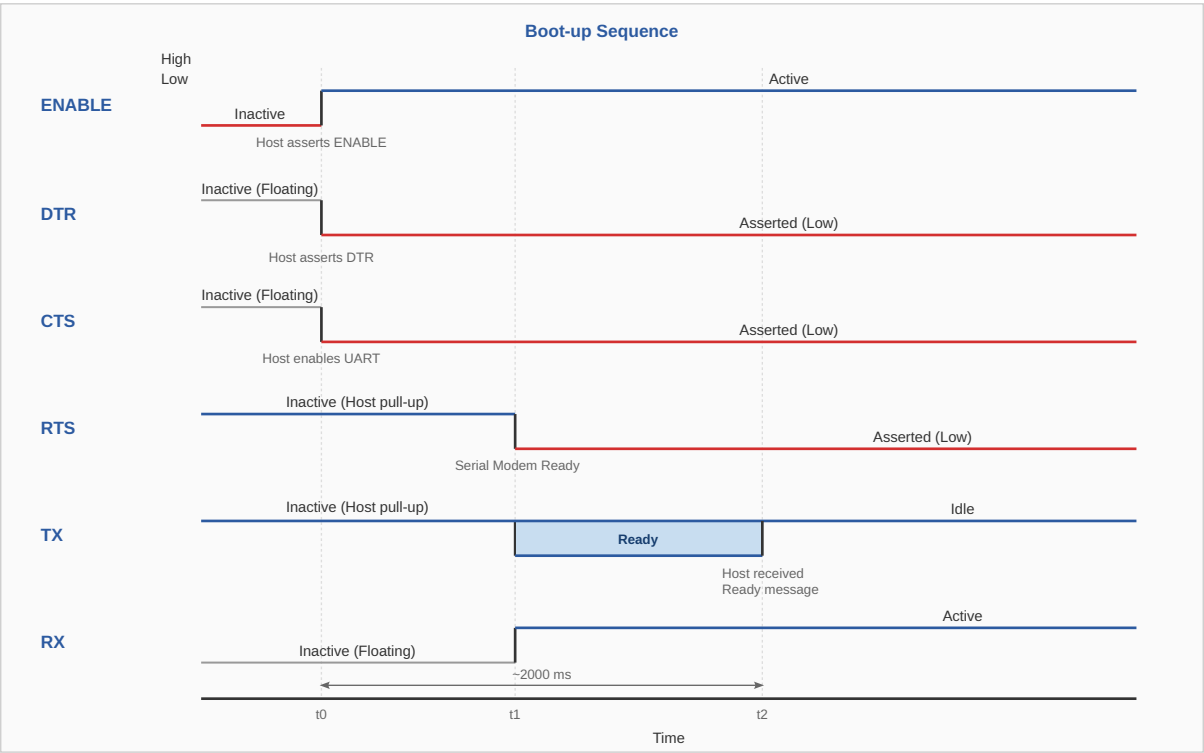


Fig. 4: Signal sequence during boot-up

The **ENABLE** pin (pin **10** on the nRF91M1 module) is a high-impedance control input for the internal power management unit (PMU) of the nRF91M1 module. Driving it to a logic high powers on the nRF91M1 module. Driving it to a logic low brings the device into an extremely low-current state. Any internal state not retained in non□volatile memory is

lost, and a full boot-up sequence is required when **ENABLE** is reasserted. See nRF9151 Hardware Design Guidelines — ENABLE for more information about the ENABLE pin.

A.3.4 Reset scenario

To reset the nRF91M1 module to a known state, do one of the following:

- Drive **nRESET** low for at least 5 µs.
- Power cycle the nRF91M1 module with **ENABLE**. Keep **ENABLE** low for at least 1 s.

After the reset, the startup sequence is the same as in the *Boot-up scenario*.

A.3.5 Zephyr hosts

If the host runs Zephyr, you can automate the UART power management with Zephyr's cellular modem driver.

See Serial Modem UART Zephyr cellular modem driver auto power for more information.

A.4 Operating without hardware flow control

Important:

The use of the nRF91M1 module without hardware flow control (RTS/CTS) is experimental and strongly not recommended. Hardware flow control ensures reliable communication and prevents timing-related issues. Without hardware flow control, there is a risk of data loss in UART communication, especially in high-throughput scenarios.

The nRF91M1 module has hardware flow control permanently enabled and it cannot be disabled. To operate without hardware flow control, do the following:

- **Wire the CTS pin to GND** — Wiring CTS to GND holds the line low, allowing the nRF91M1 module to transmit. Without this, the nRF91M1 module sees CTS as high (not clear to send) and does not transmit.
  - **Increased power consumption** — The nRF91M1 module has an internal pull-up on the CTS pin that cannot be removed. With CTS wired to GND, the pull-up draws current and increases the power consumption while the UART is active.
- **Leave RTS unconnected** — The RTS pin is not used and can be left floating.

Key requirements:

- The host must enable its own UART before asserting DTR.
- To detect when the nRF91M1 module is ready to receive, monitor the RI signal (de-asserted when ready) or use a sufficient timeout after DTR assertion.
- See *Incoming data wake-up* for detailed timing information.
- In boot-up scenarios, the host must wait for the `"Ready"` message from the nRF91M1 module before sending any data.

Buffer size:

The UART receive buffer of the nRF91M1 module is fixed to 532 bytes and cannot be configured.

A.5 Operating with the nRF9151 DK

You can flash the nRF91M1 firmware to the nrf9151dk board for development and testing purposes. On the DK, the SiP is an nRF9151, and the UART pins follow the pin mapping of the nRF91M1 as listed in the following table:

Signal	nrf9151dk pin
UART0 TX	P0.27
UART0 RX	P0.26 (pull-up)
UART0 RTS	P0.14
UART0 CTS	P0.15 (pull-up)
UART0 DTR	P0.31 (active low, pull-up)
UART0 RI	P0.30 (active low)
UART1 TX	P0.29

By default on the nrf9151dk board, UART0 is routed to VCOM0 on the interface chip, and UART1 is routed to VCOM1 on the interface chip. This allows you to use the nrf9151dk board with a PC host for development and testing.

A.5.1 PC host

Important:

When you use the nrf9151dk board with a PC host, you must wire the **DTR** pin to **GND** to power on UART0. Currently, you must use a physical jumper wire, but in the future, the Board Configurator app can be used.

When you use a PC serial terminal application (for example, PuTTY, Tera Term, minicom) to send AT commands, configure the terminal as follows:

- UART instance: VCOM0
- Baud rate: 115200
- Hardware flow control: RTS/CTS enabled
- Line terminator: CRLF
- AT command terminator: CR

Ensure that the return key sends the **CR** (Carriage Return) character as the line terminator. Incoming serial output uses both the **CR** and the **LF** (Line Feed) characters. These settings are the default configuration for the PuTTY, Tera Term, minicom, and picocom serial terminal applications.

UART1 (VCOM1) carries the log and trace output at 1000000 baud, without hardware flow control.

See *AT command syntax* for the AT command syntax.

A.5.2 External MCU host

When you use the nrf9151dk board with an external MCU host, you must disable VCOM0 in the Board Configurator app to release the UART pins for external use.

A.5.3 Cellular PPP modem

For information on how the nRF91M1 module works as a cellular PPP modem, see *Cellular PPP modem*. For information on how to configure a Linux host with the nrf9151dk board running nRF91M1 firmware, see Serial Modem PPP Linux host. For a Zephyr host, see Serial Modem PPP Zephyr host.

A.6 UART security considerations

UART is a plain serial link with no built-in encryption or authentication, so data that the host and the nRF91M1 module exchange over the UART is therefore not protected. You can reduce the exposure by keeping the UART lines hidden inside the physical security boundary of your product. When you need stronger confidentiality, terminate secure protocols on the host instead of in the modem.

The following data is sent over the UART in cleartext:

- Application data sent and received over modem-terminated TLS or DTLS sockets.
- AT commands, responses, and notifications exchanged with the nRF91M1 module that may contain sensitive information.

Choose one of the following security models based on the physical security of your product.

Aspect	Model A: Trusted UART	Model B: Untrusted UART
When to use	The UART lines are inside the physical security boundary of the product.	The UART lines are accessible outside the physical security boundary of the product, or the product requires stronger confidentiality.
Credential storage	Credentials are stored in the modem with `AT%CMNG`. Provision them in a controlled environment, such as the production line.	Credentials are stored on the host. No credentials for internet services are written to the modem.
Secure channel	Secure protocols, such as TLS, DTLS, MQTT, and HTTP, are terminated in the modem and controlled with AT commands.	All internet protocols, such as TLS, DTLS, MQTT, HTTP, and CoAP, are terminated on the host. The host uses *PPP* over *CMUX* and the modem only forwards IP packets.
Data on the UART	Application data and AT commands are in cleartext.	Application data is encrypted by the host. AT commands are still in cleartext. Commands that configure the modem, such as the SIM PIN, PDN authentication credentials, and SMS content, can carry sensitive data. Apply these settings in a controlled environment where possible. Commands that are needed in the field remain visible on the UART.

Note:

Firmware integrity does not depend on the UART. Firmware updates are signature-verified on the device, regardless of the security model.

Appendix B: Logging and tracing

The nRF91M1 module outputs application logs and modem traces on UART1. See *nRF91M1 module pin assignment* for the UART1 pins.

B.1 Application logging

The nRF91M1 module outputs B0, MCUboot, and application logs over UART1 during boot. After the application is initialized, the log backend is disabled, and the UART is suspended. Use AT#XLOG=1 or AT#XLOG=2 to enable application logs and resume the UART. This avoids UART power overhead when logs are not needed during normal operation. See *Trace AT commands* for the full command reference.

Nordic Semiconductor support might ask you for application logs when investigating an issue.

Note:

The negative error codes that are visible in logs are *errno* codes defined in nrf_errno.h.

B.1.1 Log levels

The nRF91M1 module uses the following log levels, from the most severe to the most verbose:

Level	Usage
Error (ERR)	An operation failed and the nRF91M1 module cannot complete it.
Warning (WRN)	Something unexpected happened, but the nRF91M1 module can continue.
Information (INF)	Normal operation. Includes the AT commands, responses, and notifications as text, with sensitive payloads redacted.
Debug (DBG)	Detailed internal information for debugging. Includes the AT commands, responses, and notifications as raw hex dumps.

You select what the nRF91M1 module outputs on the log UART at runtime with the AT#XLOG command:

AT#XLOG mode	Log UART	AT traffic logged as
0	Suspended	Not output
1	Active	Information level text, with sensitive payloads redacted
2	Active	Debug level hex dump

Use mode 1 for normal troubleshooting. Use mode 2 only when you need the exact bytes of the AT traffic, for example when the redaction hides the information that you need. Mode 2 outputs sensitive payloads, such as credentials, so do not share these logs publicly.

B.2 Modem traces

The modem outputs modem traces that you can use to analyze the behavior of the modem. Collect the modem traces for the following purposes:

- Inspecting the IP-level traffic during development.
- Analyzing network issues.
- Investigating modem crashes and suspected modem failures with the crash dump.

Nordic Semiconductor support might ask you for modem traces when investigating an issue.

Enable the modem traces before the problem occurs, because they cannot be collected afterwards. If the nRF91M1 module reports a modem fault with the *#XMODEM: FAULT* notification, or if you suspect another modem failure, repeat the scenario with the modem traces enabled.

B.2.1 Modem traces through UART

The nRF91M1 module outputs modem traces over UART1. The UART is shared with the application logs. The baud rate of the shared UART is set to 1000000 to support the high data rate required for modem traces.

Configuration

The application log output over UART1 and the modem trace backend (AT#XTRACE) are both included in the nRF91M1 module.

After the application is initialized, the UART is suspended. Use AT#XLOG=1 to activate application logs and AT#XTRACE=1 to activate modem traces. See *Trace AT commands* for the full command reference.

Note:

Attempting to enable one backend while the other is active returns ERROR. Always disable the active backend before enabling the other.

Note:

When measuring power consumption, make sure both backends are disabled (`AT#XLOG=0` and `AT#XTRACE=0`) so the UART stays suspended. An active trace or log backend adds approximately 700 uA overhead.

Collecting traces

The modem traces cannot be enabled while the application logs are enabled, so send `AT#XLOG=0` first.

To collect modem traces using the shared UART backend:

1. Capture the trace data from `UART1` at 1000000 baud rate, for example, with the Cellular Monitor app. See the Cellular Monitor app documentation for how to capture and save the trace.
2. Enable modem trace collection with the AT command UART:

```
AT#XTRACE=1
OK
```

3. Activate the modem to start generating trace data:

```
AT+CFUN=1
OK
```

4. Wait for the modem to register on the network and collect the traces.
5. When done, disable the modem trace backend:

```
AT#XTRACE=0
OK
```

B.2.2 Analyzing traces

You can analyze the captured trace file in the Cellular Monitor app to inspect AT commands, network events, and IP-level details of the communication between the modem and the cellular network.

If you suspect a modem failure, or if the modem crashes and the trace collection was enabled, you can send the trace file to Nordic Semiconductor support for further analysis.

B.3 nRF Cloud observability

The nRF91M1 module collects diagnostic data about its operation. The host can ask the nRF91M1 module to upload the data to nRF Cloud, so you can use it to investigate crashes of devices in the field.

The nRF91M1 module collects the following data:

- Events, such as the reset reason after a restart and a modem fault with its reason and program counter.
- Metrics, which the nRF91M1 module collects into a heartbeat every hour. The metrics include the LTE metrics and connection statistics. See *#XNRFCLOUDOBSHEARTBEAT* for more information.
- Logs, which are the most recent information, warning, and error messages, with sensitive payloads redacted. The nRF91M1 module captures logs at these levels even when `AT#XLOG=0` disables UART logging. With `AT#XLOG=2`, the nRF91M1 module logs AT traffic only as debug-level hex dumps and does not include it in the observability logs. After a crash, the logs from before the crash are part of the core dump.
- A core dump, which the nRF91M1 module stores in flash when it crashes.

The data does not leave the device on its own. The host controls when it is sent, and the data reaches the cloud as follows:

1. The nRF91M1 module records the data on the device. When the nRF91M1 module crashes, it stores a core dump in flash and restarts. The core dump stays in flash over the restart until it is uploaded. The nRF91M1 module keeps the other data in RAM, so the data is lost on reset or power loss.
2. The host connects the nRF91M1 module to nRF Cloud with the *#XNRFCLOUD* command. You must onboard and claim the device first. See *nRF Provisioning AT commands*.
3. The host uploads the buffered data with one of the following commands:
   - *#XNRFCLOUDOBSUPLOAD* uploads the data when you issue it.
   - *#XNRFCLOUDOBSAUTO* uploads the data periodically. It is disabled by default.

   The upload includes the buffered events, metrics, and logs. A stored core dump is included only when core dump upload is enabled with *#XNRFCLOUDOBSCOREDUMP*, which is the default.
4. nRF Cloud receives the data. By default, the data goes to the project that nRF Cloud associates with your device. To send the data to a specific project, specify the project key of that project in the upload command. You can analyze the events, metrics, logs, and core dumps in nRF Cloud.

Note:

The core dump storage holds one core dump. While a core dump is stored, subsequent crashes do not capture a new one until the stored one has been uploaded.

The core dump covers the crashes of the Serial Modem. The nRF91M1 module records a modem fault only as an event. The modem fault also drops the network connection, so the host cannot upload the event until the nRF91M1 module is connected to nRF Cloud again. If the host resets the nRF91M1 module before the upload, the event is lost. To investigate a modem fault, collect the modem traces as described in *Modem traces*.

See *Observability #XNRFCLOUDOBS** for the full command reference.

Appendix C: Cellular PPP modem

C.1 Overview

nRF91M1 module can provide a Point-to-Point Protocol (PPP) link over a UART so that a host can use the nRF91M1 module as a cellular dial-up modem. This is typically used in one of the following ways:

- **Linux host** - The host runs a standard PPP daemon (`pppd`) and establishes a PPP network interface (`ppp0`). Optionally, you can use CMUX to keep an AT command channel available while PPP is running.
- **MCU host (controlling chip)** - The host runs embedded RTOS, for example, Zephyr, and uses its cellular modem driver. The host application uses the IP stack on the RTOS, while nRF91M1 module provides the modem control, CMUX, and PPP transport.

PPP and CMUX are controlled through AT commands. For details, see *PPP AT commands* and *CMUX AT commands*.

C.2 PPP and CMUX

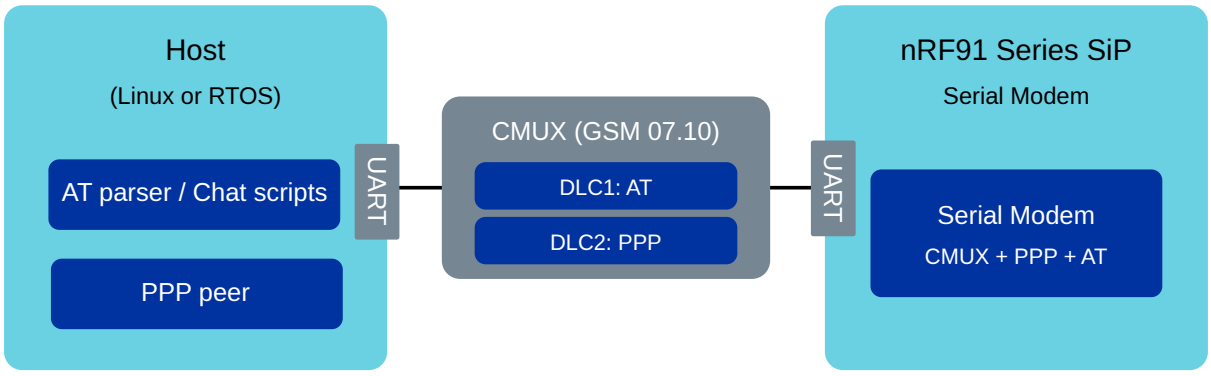


Fig. 1: CMUX multiplexes AT and PPP over one UART using separate DLC channels.

nRF91M1 module supports PPP both with or without CMUX (GSM 07.10 multiplexing). This allows you to choose the setup that best fits your use case. The recommended setup for most modem use cases is to use CMUX, as it keeps AT control available while PPP is running.

C.2.1 PPP without CMUX

Without CMUX, PPP runs on the current UART. Send `AT+CFUN=1` to activate the modem, then `AT+CGDATA` to start PPP. The modem responds with `CONNECT` and the UART imme-

diately enters PPP data mode. You cannot use the UART for AT commands until PPP is terminated by LCP negotiation. No AT notifications are delivered over the UART while PPP is active.

Example

Switching UART from AT mode to PPP mode and back:

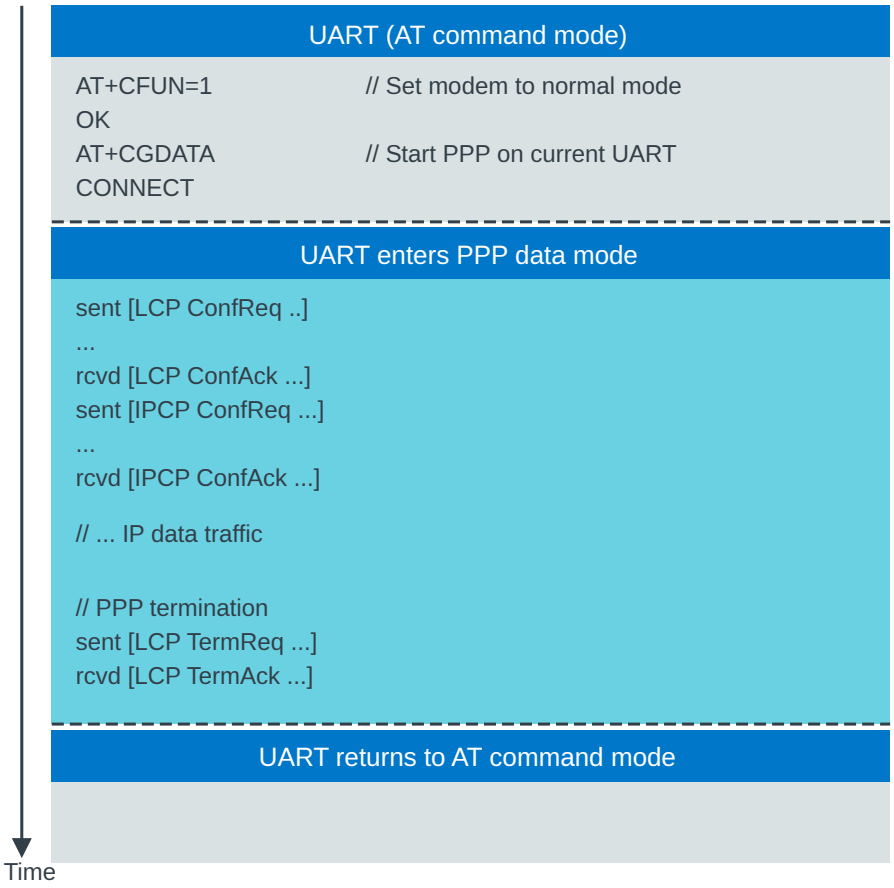


Fig. 2: PPP session without CMUX.

C.2.2 PPP with CMUX

With CMUX, AT traffic and PPP traffic can share a single UART by using separate CMUX channels.

Start CMUX with `AT+CMUX=0`, then issue `AT+CGDATA` on the chosen DLC channel to start PPP on that channel. The following example uses DLC channel 1 for PPP and DLC channel 2 for AT commands, with `AT#XCMUXURC=2` directing URCs to DLC channel 2.

Example

Send `AT#XCMUXURC=2` to route URCs to DLC channel 2, start CMUX with `AT+CMUX=0`, then start PPP on DLC channel 1 using `AT+CGDATA`:

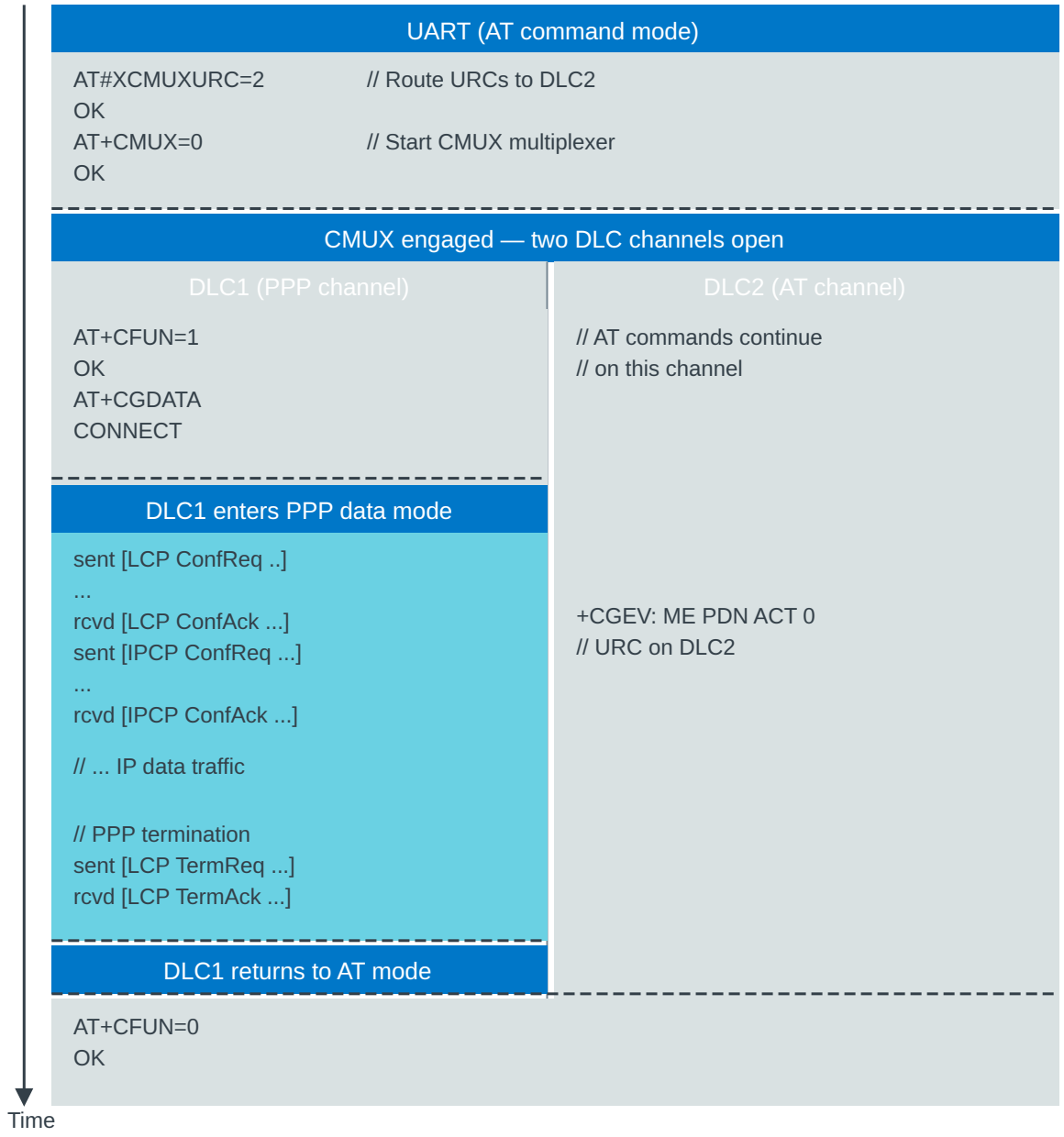


Fig. 3: PPP on DLC1, AT commands and URCs on DLC2.

Note:

The `AT#XCMUXURC=2` is optional, but it is recommended to route URCs to a static channel. When the URC channel is not selected, URCs are sent to the first open AT channel, which is not in data mode.

Note:

The sequence diagrams in this section show `+CEREG:` and `+CGEV:` URCs on the AT channel for illustration. By default, nRF91M1 module does not forward these URCs to the host.

C.3 Operational notes

- The nRF91M1 module automatically activates the default PDP context. Manually configuring the APN is typically not required.
- The nRF91M1 module automatically restarts a network scan when connectivity is lost. Manual dial-up scripts are typically not required for LTE connection recovery.
- When you use the proprietary `AT#XCMUX=1` and `AT#XPPP=1` commands, CMUX channel allocation is static (by default, allocating DLC channel 1 to PPP and DLC channel 2 to AT). See *CMUX AT commands* and *PPP AT commands* for details.
- If you enable packet domain URC forwarding with `AT+CGEREP=1`, you might see a packet domain event (`+CGEV: IPV6 FAIL 0`). This event indicates a failure to obtain an IPv6 address. This is normal and you can ignore it, because the modem does not require an IPv6 address when you use PPP. The host's IP stack handles IPv6 addressing.

Appendix D: Release notes

For detailed list on changes for a specific nRF91M1 module application firmware release, refer to the following:

D.1 Release notes for Serial Modem v2.0.0

This is the first official application firmware release for the nRF91M1 module.

It is based on the nRF Connect SDK v3.4.1 release. It is also based on the Memfault SDK v1.45.0 release, which is updated from the v1.40.1 in the nRF Connect SDK v3.4.1 release.