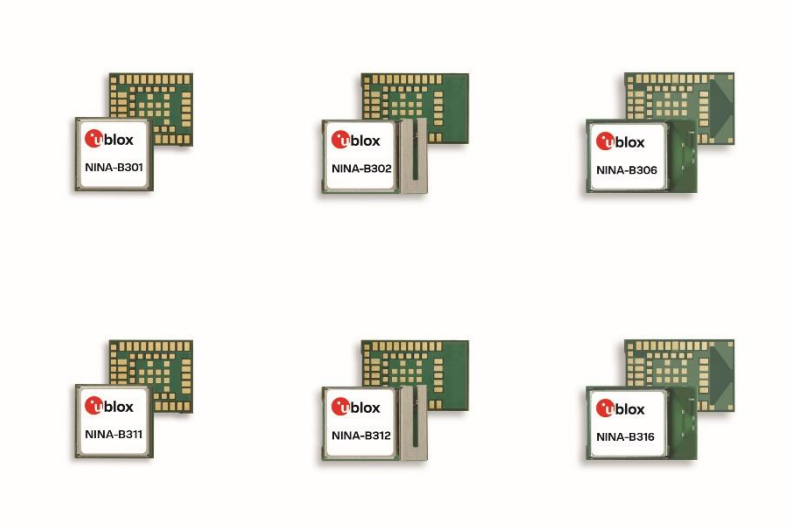
Bluetooth Mesh with u-connect software

Bluetooth Mesh

Application Note



Abstract

This application note describes how to use Bluetooth Mesh-enabled u-blox modules in a Bluetooth Mesh network.

www.u-blox.com
UBX-19025268 - R01

Document Information

Title	**Bluetooth Mesh with u-connect software**	
Subtitle	Bluetooth Mesh	
Document type	Application Note	
Document number	UBX-19025268	
Revision and date	R01	11-Jul-2019
Disclosure Restriction		

This document applies to the following products:

Product name	Type number	u-connectXpress Software version	PCN reference
NINA-B311	-	Mesh Experimental v0.7	N/A
NINA-B312	-	Mesh Experimental v0.7	N/A
NINA-B316	-	Mesh Experimental v0.7	N/A

Product name	Type number	u-connectScript Software version	PCN reference
NINA-B311	-	Mesh Experimental v0.7	N/A
NINA-B312	-	Mesh Experimental v0.7	N/A
NINA-B316	-	Mesh Experimental v0.7	N/A

Contents

1 Introduction

1.1 General information about Bluetooth Mesh

Bluetooth Mesh is a specification for forming mesh networks developed by the Bluetooth SIG (http://www.bluetooth.org). See reference [3] for additional information. It was developed to support a number of use cases for large scale networks. The nodes can communicate using one-to-one, one-to-many, and many-to-many communication. Each node in a Bluetooth Mesh network has one of the defined node types as shown in Figure 1.

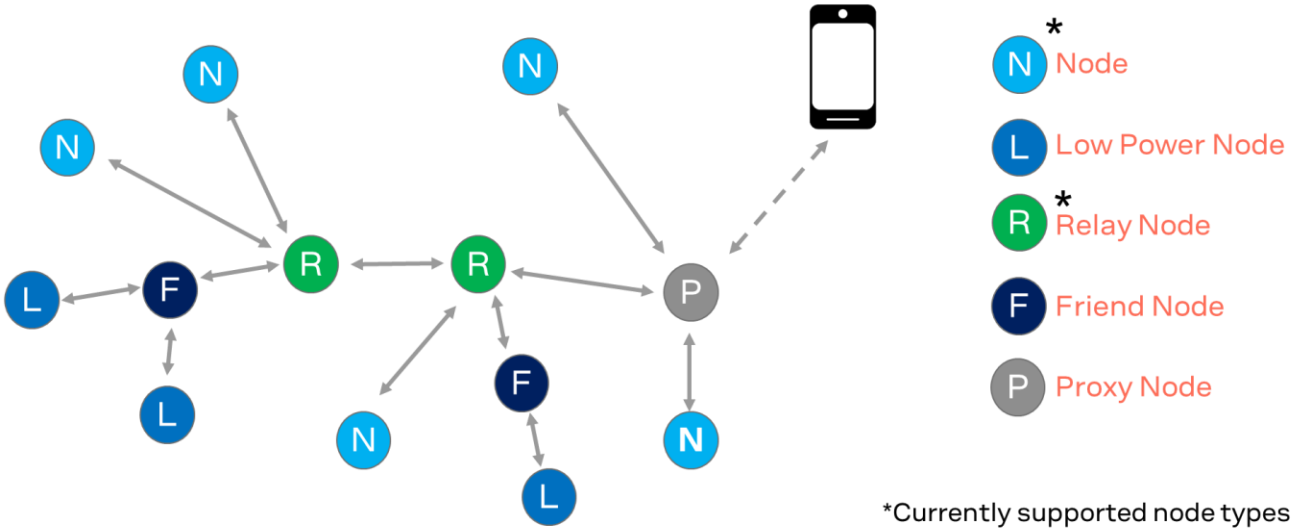


Figure 1: Different node types in Bluetooth Mesh network

The nodes communicate using messages defined by opcodes, which is a number that identifies the message.

Each node has a "model", which defines the functionality of the node, similar to the general structure of a mesh node in Figure 2. The models can be generic, as is predefined in the Bluetooth Mesh standard, or proprietary and defined by the implementer. With u-connect software, the models for the node functionality can be provided using one of the following two options:

1. JavaScript for a node, as described in chapter 3
2. AT commands and letting an external MCU handle the logic of the node, as described in chapter 2

To add a device to a mesh network, you need to "provision" it, meaning you give it an address on the network and add crypto keys for the network; see chapter 4 for a detailed description. Information about the "configuration" of a node, or how to add publish and subscription addresses for the models in the node are also provided in chapter 4.

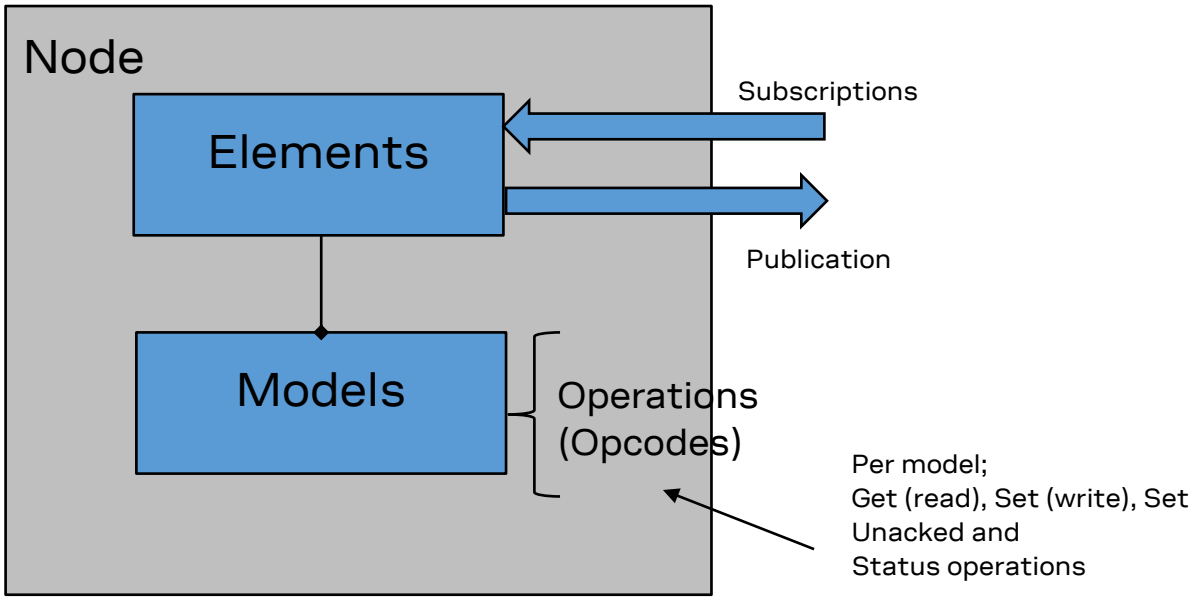


Figure 2: Elements and models in Bluetooth mesh

In a mesh node that is based on u-connect software, define the models first and then provision and configure the node.

For more information about Bluetooth Mesh in general, see [3] and [4].

1.2 Bluetooth Mesh with u-connect software

Bluetooth Mesh can be used with both u-connectXpress and u-connectScript software. With u-connectXpress software, you need to add an external MCU to handle events whereas with u-connectScript, you can design a self-contained node.

This application note will describe how to:

- Use u-connectXpress software in a Bluetooth mesh network (see chapter 2)
- Use u-connectScript software in a Bluetooth mesh network (see chapter 3)
- Provision and configure a u-blox module for a mesh network (see chapter 4)

☞ This document applies to the mesh-enabled softwares u-connectXpress Mesh Experimental 0.7 and u-connectScript Mesh Experimental 0.7. The mesh-enabled software versions for NINA-B3 can be downloaded from the u-blox website (www.u-blox.com).

1.3 Example used in this document

This document provides sample instructions for building a very small mesh network with two nodes using u-connect software. The sample setup has one server node and one client node. The server node will implement an OnOff server that enables turning on or off an LED over the mesh network, corresponding to a mesh-enabled light-bulb or smart-plug.

The server will be implemented using u-connectScript and the client that sends the messages will be implemented in u-connectXpress.

Both the server and client are running on NINA-B3 Evaluation kits.

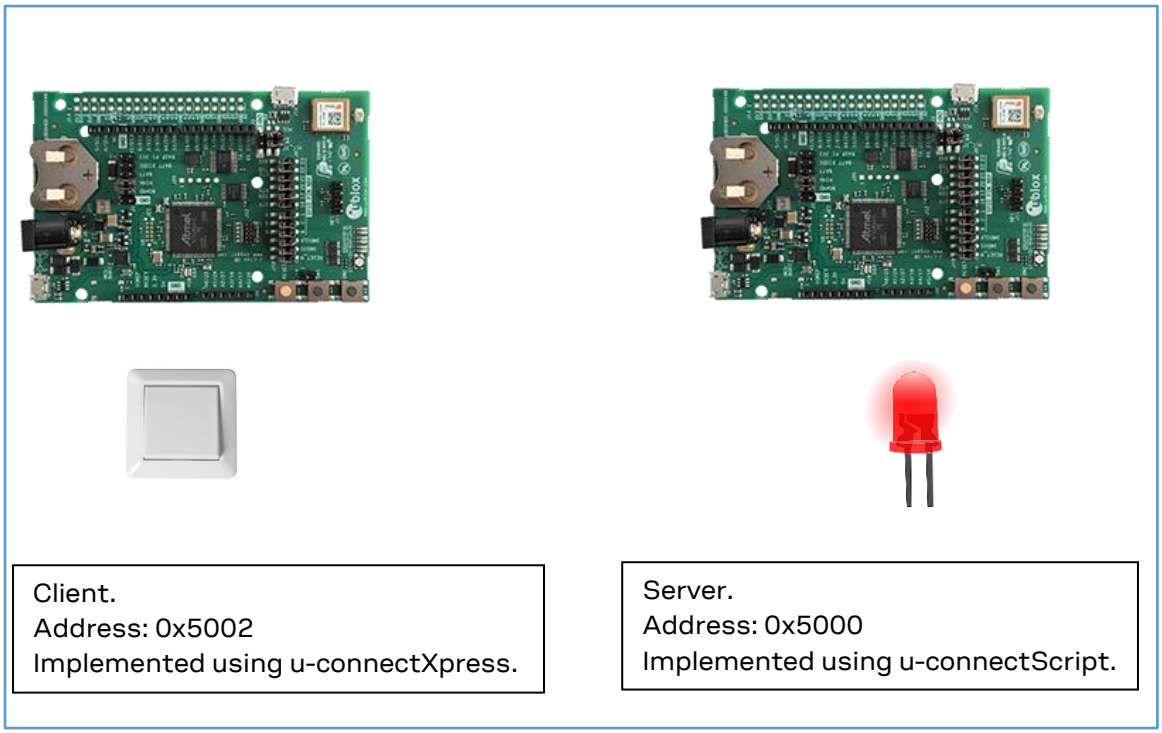


Figure 3: Overview of the sample application

For details on the NINA-B3 Evaluation kits, see the EVK-NINA-B3 User Guide [13].

For complete scripts and configuration files, refer to github [7].

It is also possible to implement the server models in u-connectXpress and client models in u-connectScript; however, this information is not explained in this document.

1.4 Configuring the Evaluation kits

This section provides information on how to configure the nodes in order to communicate on the same network and with the applications running on each node.

The server node will turn on or off the LED by simply writing to a GPIO pin on the EVK.

1. Configure one evaluation kit as a client, using u-connectXpress Mesh Experimental 0.7, as described in chapter 2.
2. Configure one evaluation kit as a server, using u-connectScript Mesh Experimental 0.7, as described in chapter 3
3. Run the example, by issuing AT-commands to the client. See chapter 5 for example commands.

All u-connect software mesh nodes have the Bluetooth mesh relay node functionality enabled by default so that it is possible to easily extend this example to several nodes.

1.5 Clearing the non-volatile memory

During development, more than just `AT+UFACTORY` is required to make a complete factory reset, including removing all mesh configuration, provisioning data and model definitions:

```
# 1) Clear the model definition and remove all scripts:
AT+UFACTORY
AT+CPWROFF
```

```
# The module will restart, without starting any script:
failed to read file "init.js"

# 2) Clear the mesh stack's flash memory and clear the local unicast address:
AT+UBTMCLR
# The module will restart again, still without scripts
failed to read file "init.js"
AT+ UBTMADRCLR
AT&W
AT+CPWROFF
# The module will restart again, still without scripts
failed to read file "init.js"

# 3) Now re-upload the scripts, and issue AT+CPWROFF again, and the scripts will start.
```

If it is only desired to unprovisioned state, step 2 above is enough. In that case, the previously defined model is kept.

The module will remain in the current state, if only the files are uploaded.

2 Mesh models in the u-connectXpress software

2.1 Introduction

In u-connectXpress, the functionality of the models is implemented using AT commands and the logic must reside on an external MCU. The external MCU communicates over a serial interface (UART) using AT commands as shown in Figure 4.

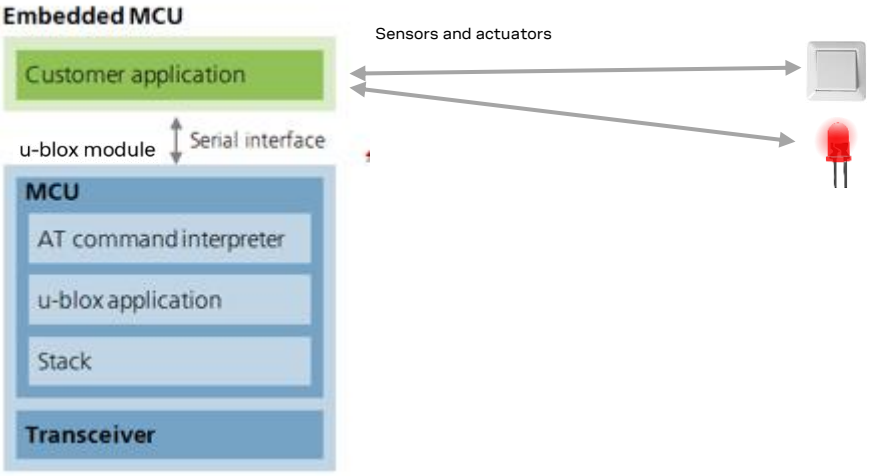


Figure 4: Software architecture of u-connectXpress

The mesh functionality in u-connectXpress is implemented using AT commands described in appendix A.

u-connectXpress nodes can be configured easily using the s-center application [8] from u-blox; however, any terminal application that can connect to a UART can also be used. If you are unfamiliar with u-connectXpress, watch the introductory video provided in the reference [10].

The u-connectXpress Mesh Experimental 0.7 software can be installed on any EVK-NINA-B3 Evaluation Kit. Instructions on reflashing the EVK-NINA-B3 using the s-center application [8] can be found in the Software section of the NINA-B3 System Integration Manual [12].

2.2 Example model in u-connectXpress

In this section a Generic OnOff client is defined in the u-connectXpress node.

When using the predefined Bluetooth SIG models, messages need not be added explicitly as they are already included in the stored model. The models are identified using the numbers as described in the Mesh Model Specification [3]. The Generic OnOff client has the SIG Model ID - 0x1001.

```
# Model index 0, Generic OnOff client
AT+UBTMMODG=0,1001
AT+UBTMELM=0,0
AT&W
AT+CPWROFF
```

As shown above, add a generic model (`AT+UBTMMODG`) and then instantiate it to an element on the node (`AT+UBTMELM`). Then, store the new model and element and restart the node.

The node is now ready to be provisioned and configured as described in chapter 4.2.1 and 4.4.2.

The `ex_mesh_o_o_c_at` directory in github contains files with the AT-command sequences mentioned above.

If using a proprietary model, define the model and its associated messages using `AT+UBTMMOD` and define your opcodes using `AT+UBTMOPC`, see appendix A for additional information.

3 Mesh models in the u-connectScript software

3.1 Introduction

With u-connectScript, a fully independent node can be implemented without the need of an external MCU, as described in Figure 5, by letting the NINA-B3 control I/O directly using its GPIO pins.

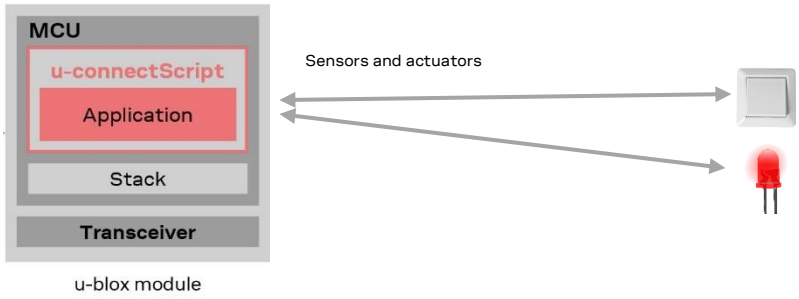


Figure 5: Software architecture with u-connectScript

The code examples in this chapter implement the Generic OnOff Server as described in chapter 1.3. These examples requires the u-connectScript Mesh Experimental 0.7 software.

Installing the u-connectScript Mesh Experimental 0.7 software is only possible from EVK revision -01 or later. The modules with older EVKs can be used only with u-connectXpress version. To verify the version of an EVK, run the AT command **ATI0.** If the reply is **"NINA-B311-XXB-00"** the module can handle both u-connectXpress and u-connectScript Mesh Experimental 0.7 software versions. If the response is **"NINA-B311-00B-00",** then the module is ony capable of running the u-connectXpress software.

Instructions on reflashing the EVK-NINA-B3 using the s-center application [8] can be found in the Software section of the NINA-B3 System Integration Manual [12].

Use Visual Studio Code with the u-blox Script IDE extension, version 1.0.2 or later, to download and modify the examples to the EVK-NINA-B3, since `AT+UJSCFM` must be executed prior to `AT+UDWNFILE`.

For more information about u-connectScript and the u-blox Script IDE, see u-connectScript with NINA-B3 series Application Note [2], or watch the video series provided in reference [11]. The complete API documentation is also available on github [7], as well as files implementing the example described in this application note.

3.2 Software structure of a node for mesh implemented in JavaScript

3.2.1 Introduction

The JavaScript code in the module has to perform the following three actions as described in the following sections:

1. Initialize the module.
2. Define the models to be used on the node. The models can be either proprietary models or predefined Bluetooth SIG models; then define the application logic for the Mesh node.
3. Set up callbacks defining the behavior for addresses of the received messages to an element within this node.

3.2.2 Initialization

To begin with, a u-connectScript mesh node should do some initial settings and startup of the system.

```
print("Mesh LED OnOff example");

/* Init module */
MiscAtExecute("AT+UBTLE=2"); //set BLE peripheral

let nameStrCommand = "AT+UBTLN=";
let localAddrStruct = MeshLocalAddressRead();

// Give the node a name
if (localAddrStruct.numberLocalAddr > 0) {
    nameStrCommand = nameStrCommand + "\"u-blox Tag 0x" +
    localAddrStruct.localAddr + "\"";
} else {
    nameStrCommand = nameStrCommand + "\"u-blox Mesh Tag\""; // If unprovisioned
}
MiscAtExecute(nameStrCommand);
print ("Executed " + nameStrCommand);

// Start the Bluetooth and Mesh on the node
let result = SystemBleStart();
assert_are_equal(result, system.RESULT_OK, "BLE system start failed");
result = SystemMeshStart()
assert_are_equal(result, system.RESULT_OK, "Mesh start failed");
```

Figure 6: Initialization of a u-connectScript mesh node

An example of a simple startup sequence can be seen in Figure 6, the basic steps are:

- Setting the module to Bluetooth low energy role peripheral
- Giving the node a name
- Starting Bluetooth and Mesh

3.2.3 Defining models and waiting for a complete setup

Once Bluetooth and Mesh are started on the node as described in section 3.2.2, the node's models and elements need to be defined. These will later be used in the provisioning and configuration of the node. The steps taken in the code are reflected in Table 3 in chapter 4.1.

As the node must be provisioned and configured for full functionality, it is recommended to have a step in the code that checks for the provisioning and configuration status of the node. A sample setup sequence is provided in Figure 7.

```
let checkMeshModelElementStatus = function() {
    let version = MeshVersionRead();
    let nodeConfig = MeshNodeConfigRead();
    let configStrings = ["Empty", "Mod+Elems1", "Mod+Elems2", "Provisioned",
"Configured"];

    print ("Current version", version, "Config state", configStrings[nodeConfig]);

    if (nodeConfig === 0) {
        print("No mesh structure, creating elements & models...")
```

```
        // Model index 0, Generic OnOff server, SIG Model ID 0x1000
        MeshModelGenericCreate(0, "\x10\x00");
        // Element nr, model index
        // Element #0: Generic OnOff, model 0
        MeshElementAdd(0,0);
        MeshVersionWrite(1);
        MiscAtExecute("AT&W");
        MiscAtExecute("AT+CPWROFF");
        return false;
    } else if (nodeConfig === 1) {
        print("Elements and models need one more reboot...");
        MiscAtExecute("AT+CPWROFF");
        return false;
    } else if (nodeConfig === 2) {
        print("Device needs to be provisioned");
        print("After provisioning, use AT&W and AT+CPWROFF to save local address in
flash");
        return false;
    } else if (nodeConfig === 3) {
        print("Mesh node needs to be configured");
        return false;
    }
    print("Mesh node subscribes/publishes to at least one address")
    return true;
};
```

Figure 7: A code snippet defining a simple Generic Bluetooth SIG OnOff client model, and verifying the status of the node

The code in Figure 7 uses the four setup states as explained in Table 3 in chapter 4.1.

By reading the current state using the function `MeshNodeConfigRead()` action is then taken depending on the current state.

1. In state 0, define the elements and models used. In this example, a Bluetooth SIG predefined Generic OnOff server is used and there is only one element in the node, just as needed for the client defined, provisioned and configured in chapter 2.2. Once you define the elements and models used, save and reboot.
2. At state 2, do an extra reboot after creating the internal structures.
3. In state 3, the device waits to be provisioned, that is get addresses and encryption keys to be used on the network. Provision using AT commands or using a smartphone app as described in chapter 4, then save and reboot using AT commands. See chapter 4.2.2 for an AT-command sequence to be used with the client defined, provisioned and configured according to chapter 2.2.
4. In state 4, the node waits to be configured with publishing and subscription addresses. See chapter 4.4.1 for a configuration example matching for the client defined, provisioned and configured in chapter 2.2.

3.2.4 Callbacks

Received mesh messages are handled in callbacks. Four different kinds of callbacks are available:

- Eddystone
- iBeacon
- Data Events
- Reliable Events callback (this callback is called when a reliable publish fails)

The callback functions must have the format defined in the mesh library files. The callbacks are added and the execution starts once the model status is checked and the setup is complete, as described in section 3.2.3.

```
let doMesh = checkMeshModelElementStatus();

if (doMesh) {
    MeshSetDataEventCallback(receiveCallback, "receiveCallback");
    MeshSetReliableEventCallback(reliableStatusCallback, "reliableStatusCallback");
    /* Configure GPIO pin */
    GpioDelete(redPin);
    result = GpioOpen(redPin, gpio.MODE_OUTPUT, gpio.VALUE_HIGH);
    assert_are_equal(result, gpio.RESULT_OK, "redPin creation failed");
    print("Ready to toggle LED...");
    setLEDState(redPin, 1);
 }
```

Figure 8: Adding callbacks for the received mesh messages

In this stage, define the GPIO pin that controls the LED.

The actual callbacks define the functionality to perform when receiving messages. In this example scenario, the generic OnOff model from the Bluetooth SIG, which has the following three mandatory input messages to handle as shown in Table 1 is implemented.

For more information on the Generic OnOff model, refer to the Mesh Model Specification found in reference [3].

Element	SIG Model ID	States		Messages	Rx	Tx
Main	0x1000	0x00	Off	Generic OnOff Get	M	
		0x01	On	Generic OnOff Set	M	
				Generic OnOff Set Unacknowledged	M	
				Generic OnOff Status		M

Table 1: Generic OnOff model characteristics

The `Generic OnOff Status` message is sent as a response to the `Generic OnOff Get` and `Generic OnOff Set` messages. The receive callback that handles these messages sets the LED status and sends the response as listed in Figure 9.

The opcode (4 digit hexadecimal number) for the messages used is defined in the Mesh Model specification provided in reference [3].

Field	Size (octets)	Notes
OnOff	1	The target value of the Generic OnOff state
TID	1	Transaction Identifier
Transition time	1	(Optional)
Delay	1	(Optional)

Table 2: Parameters for Generic OnOff Set message

The parameters in the Generic OnOff Set message that must be managed are described in Table 2. The optional parameters are ignored in this example, but may be used in other applications.

```
let reliableStatusCallback = function (reliableId, status, userdata) {
    print("Reliable publish with id ", reliableId, " failed with status ", status);
};
let receiveCallback = function (pMeshDataEvent, userdata) {
    // Convert event to meshDataEventObject type (convert to JavaScript object)
    let meshEvent = s2o(pMeshDataEvent, meshEventStructToObject);
    let s = mkstr(meshEvent.pEventData, meshEvent.dataLength);
```

```
    // Get OnOff field value
    let onOff = s.at(0);

    if (meshEvent.opcode === 0x8201) {   // 0x8201 = Generic OnOff Get
        // Reply with Generic OnOff Status (0x8204)
        MeshReply(meshEvent.eventHdl, "\x82\x04", ledStatus, 1);
    } else if (meshEvent.opcode === 0x8202) { // 0x8202 = Generic OnOff Set
        setLEDState(redPin, onOff);
        // Reply with Generic OnOff Status (0x8204)
        MeshReply(meshEvent.eventHdl, "\x82\x04", ledStatus, 1);
    } else if (meshEvent.opcode === 0x8203) { // Generic OnOff Set Unacknowledged
        setLEDState(redPin, onOff);
    } else {
        print("Unknown message received");
    }
};
```

Figure 9: Mesh callbacks

In Figure 8 and Figure 9, there is also callback for reliable messages, which is called if the publishing of a reliable message fails. In this example's server model, no reliable messages are published, so the `reliableStatusCallback()` is a dummy.

The receive callback `receiveCallback()` first converts the incoming data to a `meshDataEventObject` struct that can be used to access the incoming data and identify the opcode.

```
// Convert event to meshDataEventObject type
    let meshEvent = s2o(pMeshDataEvent, meshEventStructToObject);
```

The `pEventData` and `opcode` fields of the struct can be accessed and appropriate action taken, which is to turn on or off the LED by calling the function `setLEDState()`.

In the case of `Generic OnOff Set` messages, the requested state is received in the first byte of the `pEventData` field.

```
    let s = mkstr(meshEvent.pEventData, meshEvent.dataLength);
    // Get OnOff field value
    let onOff = s.at(0);
```

For the `Generic OnOff Get` and `Generic OnOff Set` messages that require a response with a status message, reply back to the sender using the `MeshReply` function.

```
        // Reply with Generic OnOff Status (0x8204)
        MeshReply(meshEvent.eventHdl, "\x82\x04", ledStatus, 1);
```

The complete example is stored as ex_mesh_o_o_s.js on github [8].

4 Provisioning and configuration of u-blox modules

4.1 Introduction

A mesh node needs to be provisioned with the following:

- Network key(s)
- Device key
- Application key(s)
- Unicast Address

A u-blox mesh module can be provisioned either using a smartphone application or via AT commands.

The configuration step is needed in order to give the u-blox module some information that is specific to the elements and models configured on the module, such as:

- Publishing address for messages
- Connecting application keys to the model instances
- Subscription addresses

Provisioning and configuration is done **after** the node is loaded with the initial models.

The node goes through the following five states during the provisioning and configuration, as described in Table 3.

State	Description
0	Node completely empty. Add elements and opcodes followed by store and reboot (AT&W + AT+CPWROFF). This is described in chapter 2.
1	Internal structures built. Needs a new reboot with AT+CPWROFF
2	Node can now be provisioned with AT commands or via GATT proxy (smartphone app)
3	Node is provisioned but needs to be configured for publish/subscribe. AT+UBTMCLR will bring node back to State 2
4	Node subscribes to at least one address. AT+UBTMCLR will bring node back to State 2

Table 3: State transitions during the setup of a mesh node

The AT commands used in Table 3 are described in the u-connect AT Commands Manual [1] or in appendix A for Mesh specific commands.

The u-connectScript and u-connectXpress nodes are also provisioned and configured in the same way either using AT commands over UART or the nRF mesh application as described in the following sections.

After provisioning, the module has enough information to connect to the local mesh network but needs further configuration regarding binding application keys to models, publishing and subscription.

For the initial release of the u-connect software, the module must subscribe to at least one address (could be a dummy address) to reach the configuration state 4 mentioned in Table 3.

4.2 Provisioning using AT commands

See appendix A for more information on the AT commands used. Keys for your own network can be generated from an online UUID generator, for example.

4.2.1 Client node

Below is an example of provisioning a module using AT commands. This is the client node in our example:

```
# Set network key index 0
AT+UBTMNKY=0,5F5F6E6F726469635F5F73656D695F5F

# Define  application key index 0, network index 0
AT+UBTMAKY=0,0,5F116E6F726469635F5F73656D695F5F

# Set unicast address for node (only 1 address in this example). Address 5002 (see chapter 1.3)
AT+UBTMADR=5002,1

# Set Device Key for node
AT+UBTMDKY=000102030405060708090A0B0C0D0E0F

# Store and reset
AT&W
AT+CPWROFF
```

4.2.2 Server node

This example provisions a server that can be controlled by a client provisioned according to the example in 4.2.1:

```
# Set network key index 0 (same as client module)
AT+UBTMNKY=0,5F5F6E6F726469635F5F73656D695F5F

# Define  application key index 0, network index 0 (same as client module)
AT+UBTMAKY=0,0,5F116E6F726469635F5F73656D695F5F

# Set unicast address for node (only 1 address in this example). Address 5000 (see chapter 1.3)
AT+UBTMADR=5000,1

# Set Device Key for node (can be different from other module, here a single key is used for simplicity)
AT+UBTMDKY=000102030405060708090A0B0C0D0E0F

# Store and reset
AT&W
AT+CPWROFF
```

Upon restart, the example script will print:

```
Mesh LED OnOff example
Executed AT+UBTLN="u-blox Tag 0x5000"
Current version 1 Config state Provisioned
Mesh node needs to be configured
```

4.3 Provisioning using a smartphone application

The u-blox mesh nodes can also be configured using the nRF Mesh application available on Google Play Store as mentioned in reference [5] .

☞ After the node has been provisioned by the smartphone, the provision data currently needs to be written to flash using the AT-commands `AT&W` followed by `AT+CPROFF`. This applies to both u-connectScript Mesh Experimental 0.7 and u-connectXpress Mesh Experimental 0.7.

4.4 Configuration of a u-blox mesh node

The configuration of a mesh node is done via AT commands. Example of the items that need to be configured are:

- Binding application keys to instantiated models
- Set up publishing addresses for instantiated models
- Add subscriptions for status messages from specific elements

4.4.1 Server node

```
# Bind element 0, model 0, to application key 0
AT+UBTMAKB=0,0,0

# Dummy subscription address to take node to state 4, see Table 3
AT+UBTMSUB=0,0,C001

# Restart module
AT+CPWROFF
```

Upon restart, the example script will print

```
Mesh LED OnOff example
Executed AT+UBTLN="u-blox Tag 0x5000"
Current version 1 Config state Configured
Mesh node subscribes/publishes to at least one address
Ready to toggle LED...
Dev Info 0071 0000 0000
```

4.4.2 Client node

```
# Bind element 0, model 0, to application key 0
AT+UBTMAKB=0,0,0

# Set publishing address for OnOff client address 0x5000 (Server node)
AT+UBTMPAD=0,0,5000

# Restart module
AT+CPWROFF
```

5 Running the sample mesh network

The very small mesh network should now be set up and ready to run. OnOff messages can be sent from the client node using AT commands from a terminal application, for example s-center.

```
# Turn on red LED by sending Generic OnOff Set (0x8202) with parameter 01 (On) and TID 00
AT+UBTMPUB=0,0,8202,0100
# Server node responds with Generic OnOff Status (0x8204). See appendix A.9 for parameters
+UUBTMRCV:00,0,0,8204,1,5000,1,5002,5,-53,01

# Acknowledge the reception of the Status message
AT+UBTMRPY=00

# Turn off red LED by sending Generic OnOff Set Unacknowledged (0x8203)
# with parameter 00 (Off) + TID 00
AT+UBTMPUB=0,0,8203,0000

# No status message for unacknowledged set

# Query red LED value by sending Generic OnOff Get (0x8201)
AT+UBTMPUB=0,0,8201
# Server node responds with Generic OnOff Status (0x8204). See appendix A.9 for parameters
+UUBTMRCV:00,0,0,8204,1,5000,1,5002,5,-53,00

# Acknowledge the reception of the Status message
AT+UBTMRPY=00
```

Appendix

A AT commands for mesh

A.1 Set Device Info

AT Command	Description
AT+UBTMDID = <company ID>, <product ID>, <version ID>	Add node composition data
AT+UBTMDID?	

A.1.1 Description

Set the parameters that identify the node when a provisioner is preparing to provision the node.

A.1.2 Syntax

Response	Description
OK	Successful write response
ERROR	Error Response

A.1.3 Defined values

Parameter	Type	Default Value	Min. Value	Max. Value	Description
+UBTMDID:<company ID>, <product ID>, <version ID>					
company ID	Byte array 2 octets	-	0x0000	0xFFFF	16-bit company identifier assigned by the Bluetooth SIG
product ID	Byte array 2 octets	-	0x0000	0xFFFF	16-bit vendor-assigned product identifier
version ID	Byte array 2 octets	-	0x0000	0xFFFF	16-bit vendor-assigned product version identifier

A.2 Add Model

AT Command	Description
AT+UBTMMOD=<model index>,<Type>,<company ID>,<model ID>	Adds a model
AT+ UBTMMOD =<model index>	Reads model at index <model index>

A.2.1 Description

Defines and adds a model. The model can later be instantiated to an element through AT+UBTMELM.

A.2.2 Syntax

Response	Description
+UBTMMOD:<number of opcodes>,<Type>,<Company Id><Model Id>,<opcode >[<opcode>[...]]	Read response
OK	Successful write response
ERROR	Error Response

A.2.3 Defined values

Parameter	Type	Default Value	Min Value	Max Value	Description
Model index	Integer	-	0	100 (in reality RAM memory sets the limit)	The Index the module will be saved in
Type	Integer	0	0	1	0: Server model 1: Client Model
Company ID	Byte array 2 octets	-	0	0xFFFF	Bluetooth SIG Company Identifier
Model ID	Byte array 2 octets	-	0	0xFFFF	vendor-assigned model identifier
Number of opcodes	Integer				
opcode	Byte array 2 octets				Opcodes are added to the model via AT+UBTMOPC command.

A.3 Add Generic Model

AT Command	Description
AT+UBTMMODG=<model index>,<model ID>	Adds a generic SIG model

A.3.1 Description

Add a generic model (as specified in Mesh Model Specification 1.0) and its corresponding opcodes. The opcodes can be listed using AT+ UBTMMOD =<model index>. The model can later be instantiated to an element through AT+UBTMLELM.

A.3.2 Syntax

Response	Description
OK	Successful write response
ERROR	Error Response

A.3.3 Defined values

Parameter	Type	Default Value	Min Value	Max Value	Description
Model index	Integer	-	0	100 (in reality RAM memory sets the limit)	The Index the module will be saved in
Model ID	Byte array 2 octets	-	0	0xFFFF	SIG assigned identifier

A.4 Add Opcode

AT Command	Description
AT+UBTMOPC=<model index>,<opcode>	Adds an opcode to an existing model

A.4.1 Description

Add an opcode to a previously defined model. The opcodes of the model can be listed using AT+ UBTMMOD =<model index>.

A.4.2 Syntax

Response	Description
OK	Successful write response
ERROR	Error Response

A.4.3 Defined values

Parameter	Type	Default Value	Min Value	Max Value	Description
Model index	Integer	-	0	100 (in reality RAM memory sets the limit)	The Index the module will be saved in
opcode	Byte array 2 octets	-	-	-	Bluetooth SIG or Vendor specific Opcode

A.5 Add element

AT Command	Description
AT+UBTMELM=<element index>,<model index>	Add a model to an element.
AT+UBTMLELM=<element index>	Reads models linked to element at index <element index>

A.5.1 Description

Add the model at model index to the element at the given element index. Creates a new element if necessary or adds the model to an existing element.

A.5.2 Syntax

Response	Description
OK	Successful write response
ERROR	Error Response

A.5.3 Defined values

Parameter	Type	Default Value	Min Value	Max Value	Description
+UBTMELM:<number of models>,<modelix>[<modelix>[...]]					
Element Index	Integer	-	0	29	Element index to add
model index (1..n)	Integer	-	0	100	The model index as specified in AT+UBTMMOD
number of models	Integer				
model ix	Byte array 1 octet				Hexadecimal model index

A.6 Publish

AT Command	Description
AT+UBTMPUB=<element index>, <model index>, <opcode>, [<data>]	Publishes the data

A.6.1 Description

Publishes an access layer message to the publish address of the model instance.

A.6.2 Syntax

Response	Description
OK	Successful write response
ERROR	Error Response

A.6.3 Defined values

Parameter	Type	Default Value	Min Value	Max Value	Description
Element index	Integer	-	0	29	Index of the element
Model index	Integer	-	0	100	Index of the model
Opcode	Byte array 2 octets	-	-	-	Bluetooth SIG or Vendor specific Opcode
data	Byte array Max 128 bytes	-	-	-	Message to be published

A.7 Reliable Publish

AT Command	Description
AT+UBTMRPUB=<element index>, <model index>, <opcode>, <reply opcode>, <timeout>,<reliable ID>, [<data>]	Starts publishing a reliable message

A.7.1 Description

Start publishing a reliable message.

A.7.2 Syntax

Response	Description
OK	Successful write response
ERROR	Error Response

A.7.3 Defined values

Parameter	Type	Default Value	Min Value	Max Value	Description
Element index	Integer	-	0	29	Index of the element
Model Index	Integer		0	100	Index of the model
Opcode	Byte array 2 octets	-	-	-	Bluetooth SIG or Vendor specific Opcode
Reply Opcode	Byte array 2 octets	-	-	-	Bluetooth SIG or Vendor specific Opcode
Timeout	Integer	-	0	60	The time (sec) until the message is timed out
Reliable ID	Integer	-	0	0xFFFFFFFF	The ID of this message that will be sent back in the status event (user defined)
data	Byte array Max 128 bytes	-	-	-	Message to be published

A.8 Reply

AT Command	Description
AT+UBTMRPY=<event handle>[,<opcode>,<data>]	Reply to a message received with the event +UUBTMRCV

A.8.1 Description

Reply to a reliable GET or SET. Shall also be called (without the optional parameters) after receiving a status message.

A.8.2 Syntax

Response	Description
OK	Successful write response
ERROR	Error Response

A.8.3 Defined values

Parameter	Type	Default Value	Min Value	Max Value	Description
Event handle	Integer	-	-	-	The handle received with +UUBTMRCV which this command will reply to
Opcode	Byte array 2 octets	-	-	-	Bluetooth SIG or Vendor specific Opcode
Data	Byte array Max 128 bytes	-	-	-	The reply message

A.9 Message Event

AT Event	Description
+UUBTMRCV:<event handle>,<element index>, <model index>,<opcode>,<source address type>, <source address>,<destination address type>, <destination address>,<data>	An unsolicited event from the mesh stack

A.9.1 Description

An unsolicited event from the mesh stack. It contains a message to an instance of a model with the given opcode. The user must always reply to this event with an AT+UBTMRPY command, the reply shall use the event handle.

☞ Unsolicited messages received from a group address does not need a reply. A message sent to a group address will have the handle “FF”.

A.9.2 Defined values

Parameter	Type	Description
Event handle	Integer	The event handle which will be used with the reply message.
element index	Integer	Index of the element
Model index	Integer	Index of the model
opcode	Byte array 2 octets	Bluetooth SIG or Vendor specific Opcode
source address type	Integer	0: Invalid 1: Unicast 2: Virtual* 3: Group
source address	Byte array 2 octets	Source address of the message
destination address type	Integer	0: Invalid 1: Unicast 2: Virtual* 3: Group * Virtual addresses are not supported yet
destination address	Byte array 2 octets	Destination address of the message

Parameter	Type	Description
TTL value	Integer	TTL (number of hops)
RSSI	Integer	RSSI value for last hop in mesh network
data	Byte array Max 128 bytes	The message

A.10 Publish Status event

AT Event	Description
+UUBTMPSE=<reliable ID>,<status>	An unsolicited event sent when an AT+UBTMRPUB fails or is cancelled

A.10.1 Description

This event is sent when a reliable publish message times out or has been cancelled by the user.

A.10.2 Defined values

Parameter	Type	Description
Reliable ID	Integer	The ID passed by the user when calling AT+UBTMRPUB
status	Integer	1: timed out 2: cancelled* * Cancelling a reliable message is not supported yet

A.11 Node Version

AT Command	Description
AT+UBTMVER=<version ID>	Sets a user defined value for the version of this node
AT+ UBTMVER?	Reads the current version and node configuration status

A.11.1 Description

Set a number that identifies the version of this Node. The meaning and usage of the version number is free to be defined by the user.

Reads the version number.

A.11.2 Defined values

Parameter	Type	Default Value	Min Value	Max Value	Description
Version ID	Integer	0	0	0xFFFFFFFF	The user defined version ID

A.11.3 Defined values

Parameter	Type	Default Value	Min Value	Max Value	Description
+UBTMVER:< Version ID >,<Config status>					
Version ID	Integer	0	0	0xFFFFFFFF	The user defined version ID
Config status	Integer	-	0	4	0: Node empty 1: Models and elements step #1 2: Models and elements finished 3: Provisioned 4: Configured (at least partially)

A.12 Net key

AT Command	Description
AT+UBTMNKY=<netkey index>,<netkey>	Define network key

A.12.1 Description

Add a subnetwork and its associated network key index.

A.12.2 Syntax

Response	Description
OK	Successful write response
ERROR	Error Response

A.12.3 Defined values

Parameter	Type	Default Value	Min Value	Max Value	Description
netkey index	Integer	-	0	0xFFFF	The network key index of the subnetwork being added
netkey	Byte array 16 octets	-	-	-	The network key

A.13 App key

AT Command	Description
AT+UBTMAKY=<appkey index>,<netkey index>,<appkey>	Define application key for the node

A.13.1 Description

Add an application key and its associated key index to a subnetwork.

A.13.2 Syntax

Response	Description
OK	Successful write response
ERROR	Error Response

A.13.3 Defined values

Parameter	Type	Default Value	Min Value	Max Value	Description
appkey index	Integer	-	0	0xFFFF	The application key index of the application key being added
netkey index	Integer	-	0	0xFFFF	The index of the subnetwork this application key belongs to
appkey	Byte array 16 octets	-	-	-	The application key

A.14 Subscription

AT Command	Description
AT+UBTMSUB=<element index>,<model index>,<subscription address>	Add subscription address

A.14.1 Description

Add the specified address to the global subscription list and immediately starts a subscription for the model instance defined by the specified element.

A.14.2 Syntax

Response	Description
OK	Successful write response
ERROR	Error Response

A.14.3 Defined values

Parameter	Type	Default Value	Min Value	Max Value	Description
element index	Integer	-	0	5	Index of the element
Model index	Integer	-	0	29	Index of the model
subscription address	Byte array 2 octets	-	-	-	The raw 16-bit address to subscribe to

A.15 Subscription Delete

AT Command	Description
AT+UBTMSUBD=<element index>,<model index>,<subscription address>	Delete subscription address

A.15.1 Description

Cancel subscription of the specified address for the model instance defined by the specified element.

A.15.2 Syntax

Response	Description
OK	Successful write response
ERROR	Error Response

A.15.3 Defined values

Parameter	Type	Default Value	Min Value	Max Value	Description
element index	Integer	-	0	5	Index of the element
Model index	Integer	-	0	29	Index of the model
subscription address	Byte array 2 octets	-	-	-	The raw 16-bit address to subscribe to

A.16 Local address

AT Command	Description
AT+UBTMADR = <first unicast addr>, <number of unicast addresses>	Add local unicast addresses. All addresses are consecutive based on the first one.
AT+UBTMADR?	Reads the unicast address(es)

A.16.1 Description

Set the unicast addresses of the node. Part of provisioning a node.

A.16.2 Syntax

Response	Description
OK	Successful write response
ERROR	Error Response

A.16.3 Defined values

Parameter	Type	Default Value	Min Value	Max Value	Description
+UBTMADR:<first unicast addr>, <number of unicast addresses>					
first unicast addr	Byte array 2 octets	-	-	-	First address in the range of unicast addresses
number of unicast addresses	Integer	-	1	10	Number of addresses in the range of unicast addresses

A.17 Local address clear

AT Command	Description
AT+UBTMADRCLR	Clear local unicast address(es)

A.17.1 Description

☞ Initial mesh software release specific command, may be removed in upcoming releases.

The flash memory is split in one part which can be cleared with AT+UFACTORY (product settings and JavaScript files) and one part for the mesh stack (mesh keys, configuration etc.). This command will erase the NVDS product setting storage of the node's address. AT+UBTMADR is used to set the node's Bluetooth device name in a script before the Bluetooth and mesh stack are initialized. In a product implementation this command is probably not needed.

A.17.2 Syntax

Response	Description
OK	Successful read response

A.18 Bind app key

AT Command	Description
AT+UBTMAKB = <element index>, <model index>,<appkey index>	Bind model instance to the app key

A.18.1 Description

Bind model to the app key both for publish and subscription.

A.18.2 Syntax

Response	Description
OK	Successful write response
ERROR	Error Response

A.18.3 Defined values

Parameter	Type	Default Value	Min Value	Max Value	Description
element index	Integer	-	0	5	Index of the element
Model index	Integer	-	0	29	Index of the model
appkey index	Integer	-	0	0xFFFF	The application key index of the application key added with +UBTMAKY

A.19 Publish address

AT Command	Description
AT+ UBTMPAD = <element index>, <model index>, <publish address>	Add local publish address

A.19.1 Description

Set the publish address for the given model instance.

A.19.2 Syntax

Response	Description
OK	Successful write response
ERROR	Error Response

A.19.3 Defined values

Parameter	Type	Default Value	Min Value	Max Value	Description
element index	Integer	-	0	5	Index of the element
Model index	Integer	-	0	29	Index of the model
publish address	Byte array 16 octets	-	-	-	Address to set as the current publish address

A.20 Device key

AT Command	Description
AT+UBTMDKY = <device key>	Add device key

A.20.1 Description

Add a device key. The key is implicitly bound to all network keys. This command will also bind this key to the default Config Server which all mesh nodes shall include.

A.20.2 Syntax

Response	Description
OK	Successful write response
ERROR	Error Response

A.20.3 Defined values

Parameter	Type	Default Value	Min Value	Max Value	Description
device key	Byte array 16 octets	-	-	-	The device key

A.21 Set TTL

AT Command	Description
AT+UBTMTTL = <element index>, <model index>, <TTL>	Define publication TTL for a model instance

A.21.1 Description

Set the publication TTL value for the given model instance. The setting is only valid until the next power cycle of the node.

A.21.2 Syntax

Response	Description
OK	Successful write response
ERROR	Error Response

A.21.3 Defined values

Parameter	Type	Default Value	Min Value	Max Value	Description
element index	Integer	-	0	5	Index of the element
Model index	Integer	-	0	29	Index of the model
TTL	Integer	-	0	0xFF	New default TTL value

A.22 Set relay status

AT Command	Description
AT+UBTMRLY = <relay status> [,< RetransmitCount>,< TxIntervalSteps>]	Define relay status for the node

A.22.1 Description

Set the relay status for the node. The setting is stored persistently.

A.22.2 Syntax

Response	Description
OK	Successful write response
ERROR	Error Response

A.22.3 Defined values

Parameter	Type	Default Value	Min Value	Max Value	Description
relay status	Integer	-	0	1	0: off 1: on
RetransmitCount	Integer	0	0	0xFF	Number of transmissions for each packet
TxIntervalSteps	Integer	0	20	10240	Advertisement interval in milliseconds

A.23 Clear mesh stack flash storage

AT Command	Description
AT+UBTMCLR	Erase all persistent storage used by the mesh stack.

A.23.1 Description

Initial mesh software release specific command, may be removed in upcoming releases.

In the current implementation, the node's configuration is stored using two separate flash storage implementations. To clear the mesh stack configuration on a script node, the following sequence should be used:

```
AT+UBTMCLR
<triggers system restart>
AT+UBTMADRCLR
AT&W
AT+CPWROFF
```

A.23.2 Syntax

Response	Description
OK	Successful call response

A.24 Enter script configuration mode

AT Command	Description
AT+UJSCFM	Enter script configuration mode.
AT+UJSCFM?	Read out current mode.
Responses	**Description**
<CR><LF>OK<CR><LF>	Device will enter or is in script configuration mode.
<CR><LF>ERROR<CR><LF>	Device is not in config mode or command was not correctly executed.

A.24.1 Description

This command is used to enter the script configuration mode.

It can be comparable with the already available production mode (+UPROD). The system will, after being issues, reboot and start in script configuration mode (first event +STARTUP). In this mode no scripts will be started/executed. Mesh, advertising, scanning etc will not be initialized since no scripts can be run. To exit this mode and go to normal mode a reboot is needed (+CPWROFF).

The following AT commands can only be executed in script configuration mode:

- Javascript configuration +UJSCFG
- Javascript deploy +UJSCOMPLETE
- Download file +UDWNFILE
- Delete file +UDELFILE

The following AT commands can be executed both in normal mode and script configuration mode:

- List files +ULSTFILE
- Read file +URDFILE

Example:

```
AT+UJSCFM
OK

+STARTUP
AT+UJSCFM?
OK
AT+UJSCFG=0,10
+UJSCFG:0,10
OK
```

```
AT+CPWROFF
OK

+STARTUP
AT+UJSCFM?
ERROR
```

B Glossary

Abbreviation	Definition
ASCII	American Standard Code for Information Interchange
CPU	Central Processing Unit
EVK	Evaluation Kit
GPIO	General-purpose input/output
MCU	Micro Controller unit
RSSI	Received signal strength indication
TTL	Time To Live
UART	Universal Asynchronous Receiver-Transmitter
UUID	Universally Unique Identifier

Table 4: Explanation of the abbreviations and terms used

Related documents

[1] u-connect AT Commands Manual, Doc. No. UBX-14044127
[2] u-connectScript with NINA-B3 series Application Note, Doc. No. UBX-18031558
[3] Bluetooth Mesh Networking Specifications, https://www.bluetooth.com/specifications/mesh-specifications/
[4] aBlog entry introducing Bluetooth Mesh, https://www.bluetooth.com/blog/introducing-bluetooth-mesh-networking/
[5] nRF mesh application - https://play.google.com/store/apps/details?id=no.nordicsemi.android.nrfmeshprovisioner
[6] Eddystone beacon information, https://developers.google.com/beacons/
[7] iBeacon information, https://developer.apple.com/ibeacon/
[8] u-connectScript at github, https://github.com/u-blox/u-blox-nina-b3-javascript
[9] s-center, https://www.u-blox.com/en/product/s-center
[10] Youtube playlist introducing u-connectXpress, https://www.youtube.com/watch?v=EYeD7F6LJ3A&list=PLSzSoRUA4EXuYX7OH94xxzi3TLMhVSx7X
[11] Youtube playlist introducing u-connectScript - https://www.youtube.com/playlist?list=PLSzSoRUA4EXuD4uovkz4ODPNdpdHP4DnT
[12] NINA-B3 Series System Integration Manual, Doc. No. UBX-17056748
[13] EVK-NINA-B3 User Guide, Doc. No. UBX-17056481

☞ For regular updates to u-blox documentation and to receive product change notifications, register on our homepage (www.u-blox.com).

Revision history

Revision	Date	Name	Comments
R01	11-Jul-2019	mape, flun	Initial release.

Contact

For complete contact information, visit us at www.u-blox.com.

u-blox Offices

North, Central and South America

u-blox America, Inc.

Phone: +1 703 483 3180
E-mail: info_us@u-blox.com

Regional Office West Coast:

Phone: +1 408 573 3640
E-mail: info_us@u-blox.com

Technical Support:

Phone: +1 703 483 3185
E-mail: support@u-blox.com

Headquarters
Europe, Middle East, Africa

u-blox AG

Phone: +41 44 722 74 44
E-mail: info@u-blox.com
Support: support@u-blox.com

Asia, Australia, Pacific

u-blox Singapore Pte. Ltd.

Phone: +65 6734 3811
E-mail: info_ap@u-blox.com
Support: support_ap@u-blox.com

Regional Office Australia:

Phone: +61 2 8448 2016
E-mail: info_anz@u-blox.com
Support: support_ap@u-blox.com

Regional Office China (Beijing):

Phone: +86 10 68 133 545
E-mail: info_cn@u-blox.com
Support: support_cn@u-blox.com

Regional Office China (Chongqing):

Phone: +86 23 6815 1588
E-mail: info_cn@u-blox.com
Support: support_cn@u-blox.com

Regional Office China (Shanghai):

Phone: +86 21 6090 4832
E-mail: info_cn@u-blox.com
Support: support_cn@u-blox.com

Regional Office China (Shenzhen):

Phone: +86 755 8627 1083
E-mail: info_cn@u-blox.com
Support: support_cn@u-blox.com

Regional Office India:

Phone: +91 80 405 092 00
E-mail: info_in@u-blox.com
Support: support_in@u-blox.com

Regional Office Japan (Osaka):

Phone: +81 6 6941 3660
E-mail: info_jp@u-blox.com
Support: support_jp@u-blox.com

Regional Office Japan (Tokyo):

Phone: +81 3 5775 3850
E-mail: info_jp@u-blox.com
Support: support_jp@u-blox.com

Regional Office Korea:

Phone: +82 2 542 0861
E-mail: info_kr@u-blox.com
Support: support_kr@u-blox.com

Regional Office Taiwan:

Phone: +886 2 2657 1090
E-mail: info_tw@u-blox.com
Support: support_tw@u-blox.com