aws

Programmer's Guide

AWS IoT ExpressLink

AWS IoT ExpressLink: Programmer's Guide

Table of Contents

AWS IoT ExpressLink programmer's guide v1.3

This document defines the Application Programming Interface (API) that all AWS IoT ExpressLink compliant connectivity modules are required to implement to connect any host processor to the AWS cloud.

If you have questions or issues that are not answered here, please visit the AWS re:Post for AWS IoT ExpressLink page.

See the Document history for changes in this version.

Topics

AWS IoT ExpressLink commands

See these sections for descriptions of AWS IoT ExpressLink commands in the following general categories:

Tables

1 Overview

An AWS IoT ExpressLink qualified module is a hardware connectivity module that communicates with a host processor by means of a serial interface (UART) and allows it to quickly and securely access AWS IoT Core and its services. In so doing, an ExpressLink module offloads complex and undifferentiated workloads, such as authentication, device management, connectivity, and messaging, from the application (host) processor. ExpressLink modules were conceived after discussions with microcontroller vendors, OEMs, and module makers regarding the complexity and

repetitiveness of migrating existing hardware and software designs to new or different MCUs and RTOSs. They enable a scalable migration for millions of embedded applications to cloud-connected applications.

1.1 Goals

The top-level goals are to:

- Accelerate time to market for IoT devices.
- Ease the transition to cloud connected solutions:
  - Reduce the skill gap required for cloud-connected embedded applications.
  - Allow OEMs to migrate existing designs by adding ExpressLink to existing applications with minimal modification to the existing application code.
- Dramatically reduce the resources embedded devices require to connect to AWS IoT Core and publish and subscribe to topics, regardless of the connectivity solution chosen (Wi-Fi, ethernet, or cellular):
  - An abstract API does not reveal (leak) implementation details to the customer application.
  - Configuration parameters (implementation dependent) are easily isolated.
  - Requires minimal hardware connections with defined pinouts (two wire minimum).
  - Provides stateless module communication (command mode only, single configuration).
- Support a hardware root of trust-based unique identity that allows for an optimal out-of-the-box experience and high-volume quick manufacturing, even when using untrusted Contract Manufacturers, by taking advantage of the AWS Multi Account Registration feature.
- Provide a quick evaluation experience out-of-the-box without requiring an AWS account.
- Simplify onboarding with an additional late binding option.
- Offer easy updates over the air (and over the wire) so the module and host processor can ensure security throughout the life of the product.
- Connect to standard AWS IoT Core services without additional cost and allow for heterogeneous fleets.

2 Hardware

An AWS IoT ExpressLink qualified module is generally composed of the following elements: (see block diagram)

- a module processor (MCU), that handles the AT command parser and manages the network connectivity protocols (ethernet, Wi-Fi, cellular)
- a minimum of six I/Os
- a pre-provisioned secure element or equivalent secure enclave that provides crypto hardware acceleration, random number generation and secure key storage
- a non-volatile memory that provides bulk storage sufficient to support the module's own over-the-air updates (OTA) and host processor OTA (HOTA)

2.1 Block diagram

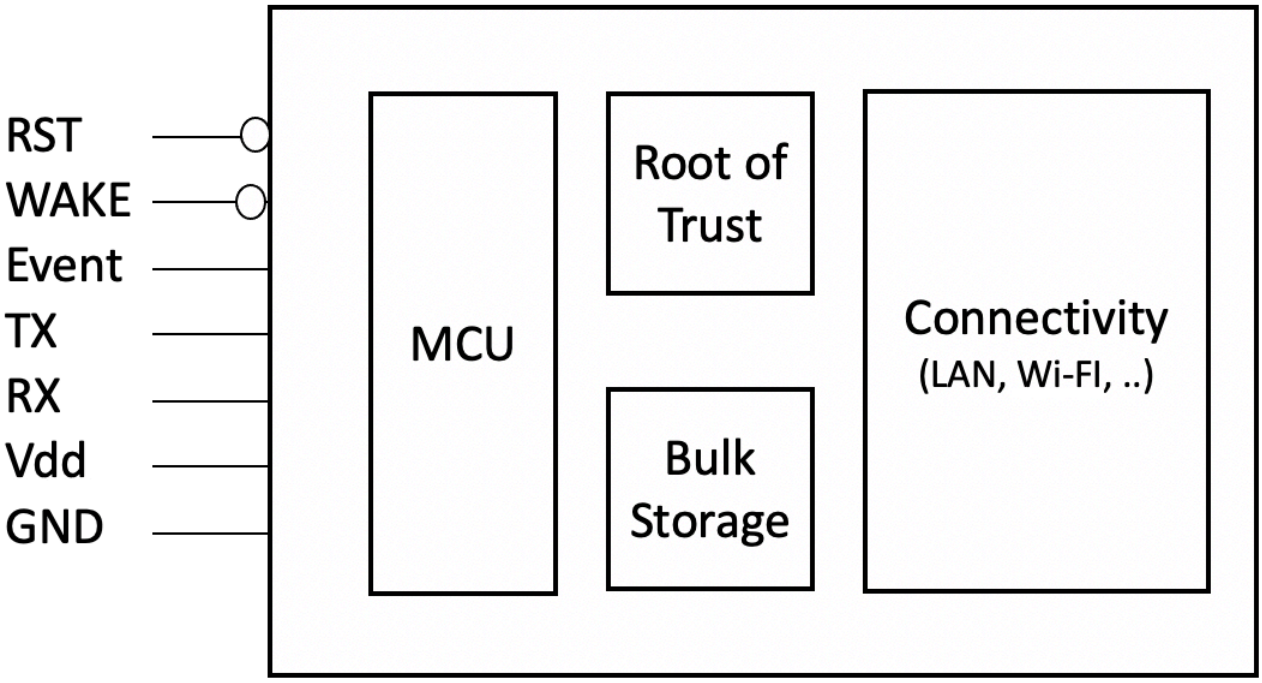


Figure 1 - Simplified block diagram

2.2 Pin definitions

2.2.1 GND (input) – Ground

2.2.2 VCC (input) – 3.3v

2.2.3 TXD (output) – Serial interface Universal Asynchronous Receiver the Transmitter (UART) TX from module

UART output to the host processor/application processor.

2.2.4 RXD (input) – Serial interface Universal Asynchronous Receiver the Transmitter (UART) RX to module

UART input to the ExpressLink, from the host processor/application processor.

2.2.5 RST (input) – holds module in reset

When asserted (low), the ExpressLink module is held in reset (low power, disconnected, all queues emptied and error conditions cleared).

2.2.6 WAKE (input) – low-power sleep mode wakeup

When not asserted (high), the ExpressLink module is allowed to enter a low power sleep mode. If in low power sleep mode and asserted (low), this will awake the ExpressLink module.

2.2.7 Event (output) – Asynchronous Event Flag

When asserted, the ExpressLink module indicates to the host processor that an event has occurred (disconnect error or message received on a subscribed topic) and a notification is available in the event queue waiting to be delivered. It is de-asserted when the event queue is emptied. A host processor can connect an interrupt input to this signal (rising edge) or can poll the event queue at regular intervals (see 8.2.1 EVENT? »Request the next event in the queue«).

3 Run states

An ExpressLink module operates as a state machine that moves through a number of internal states. See figure below for details.

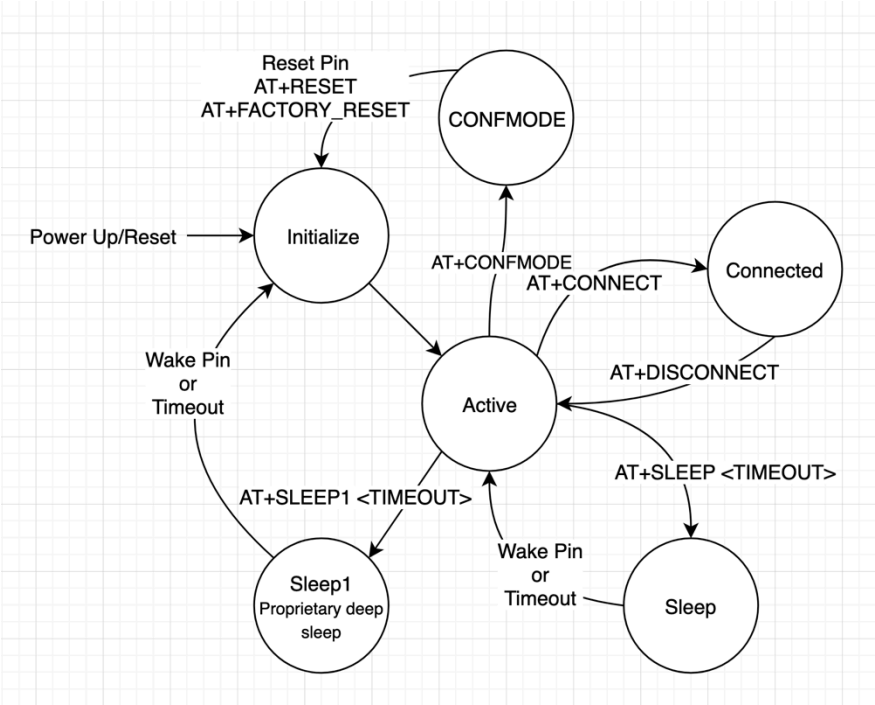


Figure 2 - ExpressLink internal states diagram (partial)

The application or host processor is presented with a small command set that is independent from the connectivity solution offered by the specific module (such as ethernet, cellular, and Wi-Fi).

The serial interface is designed to be stateless, with all interactions initiated exclusively from the host side. When an asynchronous event occurs (a message is received or an internal error condition occurs), the ExpressLink module queues the event and flags its availability to the host via the Event pin. A host can choose to ignore the Event pin (to conserve I/Os) and instead poll the module periodically. (See 8.2 Event handling commands.)

4 ExpressLink commands

4.1 Introduction

4.1.1.1 These commands are sent to and from the UART. The default UART configuration shall be 115200, 8, N, 1 (baud rate: 115200; data bits: 8; parity: none; stop bits: 1). There is no hardware or software flow control for UART communications.

4.1.1.2 The baud rate is NOT configurable.

4.1.1.3 No Local Echo is provided.

> **Note**
>
> Communication between the ExpressLink modules and the AWS Cloud are encrypted both during transmission (using the TLS 1.2 protocol) and while at rest. However, the serial interface (UART) between the host processor and the module isn't encrypted. If sensitive data needs to be transmitted to and from the ExpressLink module, and unauthorized persons can potentially gain physical control of the device, we recommend that the host processor and the corresponding cloud application implement a suitable, additional end-to-end message encryption scheme.

4.2 ExpressLink commands format

All ExpressLink commands assume the following general format:

AT+*{command}[#]{separator}[parameter]{EOL}*

Where:

4.2.1 *{command}*

A short, alphabetical character string (including "_", "!", and "?") that matches one of the commands listed in the following sections (CONNECT, TIME?, FACTORY_RESET).

> **Note**
>
> Commands are not case sensitive, although in this document, uppercase is always used for consistency.

Returns:

4.2.1.1 ERR3 COMMAND NOT FOUND

If the command is unknown, then the module returns 'COMMAND NOT FOUND'.

4.2.2 *[#]*Optional numerical suffix (first parameter)

An optional decimal (0..N) suffix qualifier (multiple digits allowed) is used by selected commands as a first numerical parameter.

Returns:

4.2.2.1 ERR4 PARAMETER ERROR

If a numerical suffix was provided but the command did not expect it, or if a numerical suffix is missing but required, the module returns 'ERR4 PARAMETER ERROR'.

4.2.2.2 ERR7 OUT OF RANGE

If the numeric suffix is out of the valid range for the command, the module returns 'ERR7 OUT OF RANGE'.

4.2.4 *{seperator}*

A single ascii space character (0x20).

Returns:

4.2.4.1 ERR2 PARSE ERROR

Return ERR2 PARSE ERROR if ANY character other than 0x20 is present after the numerical suffix or '?' in the command string.

4.2.5 *[parameter] optional parameter*

An (escaped) ASCII string with the data required for the command.

Returns:

4.2.5.1 ERR4 PARAMETER ERROR

Return ERR4 PARAMETER ERROR if the command is unable to process the parameter supplied.

4.2.6 *{EOL}*

The ASCII *line feed* character (0x0a) OR the ASCII carriage return character (0x0d).

4.2.7 Parameter string note

The parameter string includes all bytes from the separator to the *{EOL}*, not including either the separator or the *{EOL}*. ALL ASCII values from 0 - 0xFF are valid in the parameter string which allows for binary payloads if proper escaping is performed as detailed in 4.3 Delimiters and escaping.

4.3 Delimiters and escaping

The format described in the previous section, and the specific choice of delimiters, removes the need for quotes surrounding parameters, and for other delimiters between successive parameters. As a further benefit, this removes the need for most escaping sequences with the exclusion of the ASCII characters *{EOL}* (0x0a or 0x0d) and backslash ('\').

4.3.1 *{EOL}* in the parameter string (input escaping)

if a line feed character (0x0a) or carriage return character (0x0d) is required in the parameter string, it must be replaced by the backslash escaped sequence as follows:

4.3.1.1 Line feed is escaped as: 0x5C 0x41 or '\A'.

4.3.1.2 Carriage return is escaped as: 0x5C 0x44 or '\D'.

4.3.2 Backslash ('\') in the parameter string

Backslash (0x5C) in the parameter string is replaced by the escape sequence: 0x5C 0x5C ('\\').

4.3.2.1 All other combinations of the escape sequence are illegal and the module returns 'ERR5 INVALID ESCAPE'.

4.3.3 Formatting and Parsing Errors

Parsing of a command is immediately terminated when the first formatting error condition is detected. The module then discards the remainder of the command input up to the closing EOL character and reports the appropriate error code as indicated in 4.6 Command responses and error codes.

4.3.4 EOL in the command response (output escaping)

If a line feed character (0x0a) or a carriage return character (0x0d) is present in a command response string, it is replaced by the backslash escaped sequence as follows:

4.3.4.1 Line feed is escaped as: 0x5C 0x41 or '\A'.

4.3.4.2 Carriage return is escaped as: 0x5C 0x44 or '\D'.

4.3.5 Backslash ('\') in the command response

Backslash (0x5C, '\') in a command response is replaced by the escape sequence: 0x5C, 0x5C or '\\'.

4.4 Maximum values

4.4.1 Maximum bytes in the formatted command string

The formatted command string as received by ExpressLink can be up to 9K bytes in length.

AT+[up to 9K bytes]*{EOL}*

4.4.2 Maximum command word size

The command word portion of the command string can be up to 32 bytes long.

4.5 Data processing

4.5.1 Data entry

The data entry for a command begins with the 'AT+' and ends with the *{EOL}*. The module will not begin running a command before it receives the *{EOL}*.

4.5.2 Data overflow

If the data buffer overflows during the data entry phase of a command, the ExpressLink module continues to accept, but discards, the incoming data until the next *{EOL}* arrives.

4.5.2.1 The module returns 'ERR1 OVERFLOW' and the entire message is discarded.

4.5.3 Data arriving after *{EOL}*

Any data that arrives after *{EOL}* and before 'AT+' will be ignored and discarded. Note that this includes multiple *{EOL}* characters–they will be ignored and discarded.

Example

```
abcdefAT{EOL}         spurious characters preceding a command are ignored
OK{EOL}

AT{0x0a}{0x0d}{EOL}   line feed followed by carriage return
OK{EOL}

AT{0x0d}{0x0a}{EOL}   carriage return followed by line feed
OK{EOL}

AT{0x0d}{0x0d}{EOL}   multiple carriage returns
OK{EOL}
```

4.6 Command responses and error codes

All commands respond according to the response format described in section 4.6.1 General response formats when the command is completed. In some cases, this can take a significant amount of time, but under no circumstances longer than the *response timeout* defined in section 4.6.2 Response timeout.

4.6.1 General response formats

OK*[#]*|ERR*`{#}{separator}[detail]{EOL}`

Where:

OK*[#]*

Indicates that the command was valid and ran correctly. The optional numerical suffix `[#]` indicates the number of additional output lines, with no additional lines expected if this suffix is omitted.

ERR*{#}*

Indicates the command was invalid or an error occurred while running it. The required numerical suffix is an error code as defined in Table 1 - Error codes.

{separator}

Is a *single* ASCII space character (ASCII 0x20).

[detail]

Is an optional ASCII string that contains the command response or error description.

{EOL}

Is composed of a *carriage return* (ASCII 0x0d) followed by a *newline* character (ASCII 0x0a).

`4.6.1.1`

A host application receiving an error response is recommended to stop parsing any additional character (*detail*) past the error # as those are meant to assist the user during debugging with a terminal (unless it intends to present it on a user interface or display).

4.6.1.2

Similarly on a successful response, the *detail* may consist of multiple parts. To ensure future compatibility, a host application must stop parsing past the required parts as documented in the current technical specification revision.

Table 1 - Error codes

Code	ExpressLink text	Description
1	OVERFLOW	More bytes have been received than fit in the receive buffer.
2	PARSE ERROR	Message not formatted correctly.
3	COMMAND NOT FOUND	Invalid command.
4	PARAMETER ERROR	Command does not recognize the parameters.
5	INVALID ESCAPE	An incorrect escape sequence was detected.
6	NO CONNECTION	Command requires an active connection to AWS IoT.
7	OUT OF RANGE	The index provided is out of range (0 or greater than MaxTopic).
8	PARAMETER UNDEFINED	The key provided references an empty configuration parameter.
9	INVALID KEY LENGTH	Key is longer than 16 characters.
10	INVALID KEY NAME	A non-alphanumeric character was used in the key name.
11	UNKNOWN KEY	The supplied key cannot be found in the system.
12	KEY READONLY	The key cannot be written.
13	KEY WRITEONLY	The key cannot be read.

Code	ExpressLink text	Description
14	UNABLE TO CONNECT	The module is unable to connect.
15	TIME NOT AVAILABLE	A time fix could not be obtained.
16	LOCATION NOT AVAILABLE	A location fix could not be obtained.
17	MODE NOT AVAILABLE	The requested mode is not available.
18	ACTIVE CONNECTION	An active connection prevents the command from running.
19	HOST IMAGE NOT AVAILABLE	A host OTA command was issued but no valid HOTA image is present in the OTA buffer.
20	INVALID ADDRESS	The OTA buffer pointer is out of bounds (> image size).
21	INVALID OTA UPDATE	The OTA update failed.
22	[reserved]	
23	INVALID SIGNATURE	A signature verification failed.
24	SHADOW ERROR	Shadow support disabled, not initialized, or request rejected.
25	NOT ALLOWED	The module cannot accept the command at this time (it is busy or operating in a mode that conflicts with the request).
26	INVALID CERTIFICATE	The certificate was invalid or corrupted.
27	BLE ERROR	Any error related to failed execution of BLE commands.
28	CONFIGURATION ERROR	When a command is entered but the correct configuration is not set.

Code	ExpressLink text	Description
29	INSUFFICIENT SECURITY	When a subscription to a characteristic has insufficient security levels.

> **Note**
>
> Refer to section 4.3 Delimiters and escaping for how special characters are escaped in the command response string.

4.6.2 Response timeout

The maximum runtime for every command must be listed in the module manufacturer's datasheet. No command can take more than 120 seconds to complete (the maximum time for a TCP connection timeout).

4.6.3 AT »Communication test«

By sending only the 'AT' (attention) command, the host can verify the presence and readiness of the module command parser.

Example:

```
AT{EOL}    # request the module's attention
```

Returns:

OK*{EOL}*

If the module is connected and the command parser active, then the module returns 'OK'.

4.7 Power and connection control

4.7.1 CONNECT? »Request the connection status«

Requests the current status of the connection to the AWS cloud and the device onboarding state (see 21.3.2 ExpressLink onboarding states and transitions). The connection status indicates the

completion of the entire sequence of actions required for the module to connect and authenticate with the AWS cloud. The onboarding state is determined by comparing the current Endpoint configuration parameter (string) against the module default Endpoint (staging account) string that is hardcoded as the factory reset value for the parameter (see the Endpoint entry in Table 2 - Configuration Dictionary Persistent Keys).

Returns:

`OK {status}{onboarded}[CONNECTED/DISCONNECTED][STAGING/CUSTOMER]`

4.7.1.1 `OK 1 0 CONNECTED STAGING`

If the device is connected to the staging account, then the module returns 'OK 1 0 CONNECTED STAGING'.

4.7.1.2 `OK 0 0 DISCONNECTED STAGING`

If the device is not connected to the staging account, then the module returns 'OK 0 0 DISCONNECTED STAGING'.

4.7.1.3 `OK 1 1 CONNECTED CUSTOMER`

If the device is connected and onboarded (customer account), then the module returns 'OK 1 1 CONNECTED CUSTOMER'.

4.7.1.4 `OK 0 1 DISCONNECTED CUSTOMER`

If the device is not connected (customer account), then the module returns 'OK 0 1 DISCONNECTED CUSTOMER'.

4.7.2 CONNECT »Establish a connection to an AWS IoT Core Endpoint«

Request a connection to the AWS IoT Core account, indicated by the endpoint configuration parameter. This command activates the (wireless) connectivity features of the ExpressLink module and, if successful, will bring the device to a higher power consumptions state. If the configuration parameter LWTConfig is set (not empty) in the Table 2 - Configuration Dictionary Persistent Keys, a “last will and testament” is requested upon connecting to the MQTT broker using the LWTConfig options and the LWTMessage.

> **Note**
>
> This command is blocking. The connection process can require a long time during which no further communication is possible with the module until one of the following responses

is returned to the host. (For a non-blocking option, see the asynchronous command 4.7.8 CONNECT! »Non-blocking request to connect to IoT Core«.

Returns:

4.7.2.1 OK 1 CONNECTED

The module has successfully connected to AWS IoT Core.

4.7.2.2 ERR14 *{#hint}* UNABLE TO CONNECT *[detail]*

The module is unable to connect. Additional clues can be provided by the mandatory *{#hint}* numerical code and the optional *[detail]* field. The hint numerical codes indicate the state of advancement of the connection process when the failure occurred so that meaningful debugging tips can be provided in the module documentation (including datasheets and FAQs). They are numbered according to the following sequence of steps:

1.	**Backoff algorithm imposed delay** (see 4.7.2.4)
2.	**Failed to access network** – reported by a Wi-Fi module when it fails to connect to a local access point/router or by a cellular module if it fails to connect to the nearest cell tower. **Tip** Check SSID/passphrase or local router state. After this step the device is assumed to be able to communicate over the network (it has obtained an IP address).
3.	**Failed to reach AWS endpoint** – reported when the device fails to connect to an AWS endpoint. **Tip** Check the endpoint configuration parameter (URL)

	After this step, the device is assumed to have reached an AWS server.
4.	**Failed to securely authenticate with AWS** – reported when the device fails to upgrade the socket to a secure socket (TLS). **Tip** Check if the AWS root certificate might have expired. After this step, a secure socket is established with AWS.
5.	**Failed to login AWS (MQTT) broker** – reported when the MQTT login is unsuccessful **Tip** Check if the device certificate is present in the customer account registry. After this step, the device should be able to issue MQTT commands.
6.	**Failed to register for Jobs** – reported when the device fails to publish or subscribe to standard AWS topics used for JOBS/OTA (connection dropped by AWS server) **Tip** Check policies attached to device certificate. After this step, the device is connected and fully functional.

Different modules will interpret the hint codes according to the specific wireless/networking stack that is applicable for the given technology and will provide meaningful tips in the module documentation. Some of the steps might not be applicable to all technologies (for example, the hint code for step 2 might not apply for a LoRA or Bluetooth module that transitions directly

from step 1 to 3). Similarly, additional intermediate hint codes can be provided using dot notation, as applicable, to provide finer granularity (for example, a hint code 5.1 can be added between step 5 and step 6).

4.7.2.3 OK 1 CONNECTED

If the ExpressLink module is *already connected*, issuing a CONNECT command returns immediately with a success response ('OK 1 CONNECTED').

4.7.2.4 ERR14 *{#hint}* UNABLE TO CONNECT *[detail]*

In case of a connection failure, the ExpressLink module keeps a timestamp of the event. This is used to ensure that a subsequent (repeated) connection request complies with the correct *backoff timing* limits. If the request from the host is repeated too soon after the previous attempt (the interval between requests is shorter than the prescribed minimum backoff time), the ExpressLink module will return ERR14 with an appropriate hint code. The necessary delay will increase according to the backoff algorithm until a successful connection is established.

4.7.2.5 ERR25 NOT ALLOWED*{EOL}*

The CONNECT command cannot be issued when the device is in CONFMODE or otherwise busy with activities that require conflicting resources.

Examples:

```
AT+CONNECT        # request to connect
OK 1 CONNECTED    # connection established successfully
```

Or

```
ERR14 3 UNABLE TO CONNECT Invalid Endpoint?    # Error detail and hint detail/tip
 provided
ERR14 5 UNABLE TO CONNECT                     # Hint code but no hint detail
 provided
```

Or

```
ERR25 NOT ALLOWED  # The command cannot be accepted until the asynchronous
 connection
                   #   attempt is completed successfully or otherwise
```

4.7.3 DISCONNECT »Leave the connected state and enter the active state«

This command allows the host to prepare for a transition to low power (using the SLEEP command), or to update the connection parameters before it attempts to reconnect again with the changed parameters (using a new CONNECT command).

Returns:

4.7.3.1 `OK 0 DISCONNECTED`

Note that if already disconnected, the command will return immediately with a success value ('OK 0 DISCONNECTED').

4.7.3.2 Transitioning from a connected to a not-connected state always produces a CONLOST event (see Table 4 - ExpressLink event codes) independently of the cause. For example, a CONLOST event is produced as the result of the DISCONNECT command, when the server/endpoint drops the connection, or when local wireless connectivity is lost.

4.7.4 SLEEP*[#] [duration]* »Request to enter a low power mode«

This command forces the module to enter a low power mode. ExpressLink module manufacturers can implement specific low power modes with different values (#) that correspond to deeper sleep states (as capable) to provide the lowest power consumption and longest possible battery life in diverse applications and use cases. The manufacturer documents the power consumption figures achievable in such modes in the module datasheet.

4.7.4.1 The [duration] parameter

If present, this indicates the number of seconds before the module awakes automatically.

4.7.4.2 Absence of duration

If the duration parameter is absent, the module remains in low power mode until:

1. a hardware Reset is generated by the host lowering the **RST pin**.
2. a wakeup event is generated by the host lowering the **WAKE pin**.
3. a new AT command is sent by the host using the serial interface (this might not be possible in case of advanced (deep) sleep modes, see 4.7.4.4)

4.7.4.3 A SLEEP command without a numerical suffix defaults to mode 0.

Mode 0 is the default low power mode where the ExpressLink module reduces its power consumption as much as possible while it still maintains the serial interface active and preserves the contents of all configuration parameters.

4.7.4.4 Before entering SLEEP mode, the device will empty the event queue.

Advanced low power modes can disable the serial command interface. In these cases, in absence of the sleep duration parameter, the only way to awaken the device is to apply an external reset or wake signal. *Deep sleep* states might cause loss of part or all volatile (RAM) information, including all module state information including configuration parameters that are not maintained in non-volatile memory (for example, Topics). The host processor must *reconfigure* such parameters as required by the application.

Returns:

4.7.4.5 `OK` *`{mode}[{detail}]`*

The device is ready and will proceed to the lower power mode selected immediately after sending the reply (and flushing the serial port output). {mode} indicates the sleep mode activated.

4.7.4.6 `ERR18 ACTIVE CONNECTION`

The device cannot transition to a low power mode because there is an active cloud connection. Use the DISCONNECT command first to shut down the connection.

4.7.4.7 Sleep mode fall back

When the host requests a SLEEP mode higher than any implemented on the specific ExpressLink model, the module will fall back to the nearest/highest mode available. (For example, SLEEP9, might fall back to SLEEP3 if mode 3 is the highest available or simply SLEEP if no advanced modes are available.) The actual sleep mode activated is reported in the response.

4.7.4.8 Upon returning to the active state, a STARTUP event is generated and added to the event queue.

(See 8 Event handling.)

4.7.4.9 Sleep modes for devices supporting Bluetooth Low Energy (BLE)

Modules implementing the BLE API are expected to continue to offer support for the BLE feature while in default low power mode (SLEEP, SLEEP0)- BLE events are able to awaken the module. Advanced low power modes (SLEEP1 and greater) can disable the BLE radio to further reduce the power consumption of the device. Refer to the specific manufacturer's module datasheet to determine which modes are available and how they affect the BLE functionality.

Example 1:

```
AT+SLEEP 100  # Disconnect and suspend all activities for 100 seconds
OK 0          # Enters sleep mode 0 (default)
AT+CONNECT    # Resume connection
```

Example 2:

```
AT+SLEEP9     # Request a deep sleep (proprietary mode) indefinitely
OK 3          # Enters nearest/deepest sleep mode available on this model
```

Note that the device might require a hardware reset/wake event to be re-awakened, and all status (non-volatile) information might be lost requiring a new initialization and configuration.

Example 3:

```
AT+SLEEP SOME TEXT
ERR4  PARAMETER ERROR     # a numerical value is expected for {duration}
```

Example 4:

```
AT+SLEEP9A
ERR4  PARAMETER ERROR     # a numerical value is expected for {mode}
```

4.7.5 CONFMODE *[parameter]* »Activate modal credential entry«

Some ExpressLink modules require the user to enter private/local credentials manually (for example, the Wi-Fi SSID and passphrase) or by means of a dedicated (mobile) application. The host can request the ExpressLink module to enter a special configuration mode (or CONFMODE, see Figure 2) to enable or repurpose an interface (such as BLE or Wi-Fi) to receive the user input. Refer to the module manufacturer's datasheet for details specific to your model.

Example 1: An ExpressLink Wi-Fi module could use this command to enter a SoftAP mode, temporarily assume the role of an Access Point, and serve an HTML form. This would allow the user to enter the local Wi-Fi router credentials using a mobile device web browser. The optional parameter could be used to provide a customized, unique SSID based on the device UID.

Example 2: If a Bluetooth interface is available, the ExpressLink module could receive the credentials using a serial interface (SPP profile). For Bluetooth LE modules, this could be performed using a dedicated (GATT) service using a custom mobile application.

Returns:

4.7.5.1 `OK CONFMODE ENABLED`

The device has entered CONFMODE and is ready to receive user input.

4.7.5.2 `ERR17 MODE NOT AVAILABLE`

This ExpressLink model/version does not support CONFMODE.

4.7.5.3 `ERR18 ACTIVE CONNECTION`

The device cannot enter CONFMODE because it is currently connected. The host must disconnect first.

4.7.5.4 While in CONFMODE, an ExpressLink module can still process all commands that do not require an active connection (for example, 'AT+CONF? Version').

4.7.5.5 Commands that require an active connection return 'ERR6 NO CONNECTION'. Attempting to issue a CONNECT command while in CONFMODE results in an 'ERR14 UNABLE TO CONNECT'.

4.7.5.6 The host may issue a RESET command at any time to exit CONFMODE (see Figure 2).

4.7.5.7 A CONFMODE notification event (see Table 4 - ExpressLink event codes is provided to the host when the entry of new credentials has been completed. Only after that can the host issue a new CONNECT command to attempt to establish a connection using the newly entered credentials.

4.7.6 RESET »Request a full reset of the ExpressLink internal state«

This command disconnects the device (if connected) and resets its internal state. Non-persistent configuration parameters (see Table 3 - Configuration dictionary non-persistent keys) are reinitialized, all subscriptions are terminated, and the message queue is emptied.

Returns:

4.7.6.1 `OK{EOL}`

If the command was successful, the module returns 'OK'.

4.7.6.2 A **STARTUP** event is added to the event queue when the process is completed.

4.7.7 FACTORY_RESET »Request a factory reset of the ExpressLink module«

This command performs a full factory reset of the ExpressLink module, including re-initializing all non-persistent configuration parameters (see Table 3 - Configuration dictionary non-persistent keys) and selected persistent parameters (as indicated in Table 2 - Configuration Dictionary Persistent Keys in the Factory Reset column), and the message queue is emptied.

Returns:

4.7.7.1 `OK{EOL}`

If the command was successful, the module returns 'OK'.

4.7.7.2 A **STARTUP** event is added to the event queue when the process is completed.

4.7.8 CONNECT! »Non-blocking request to connect to IoT Core«

Request a connection to the AWS IoT Core account indicated by the endpoint configuration parameter. This command activates the (wireless) connectivity features of the ExpressLink module and, if successful, brings the device to a higher power consumptions state. If the LWTConfig configuration parameter (see Table 2 - Configuration Dictionary Persistent Keys) is set (not empty), the “last will and testament” is requested upon connecting to the MQTT broker using the LWTConfig options and LWTMessage.

Note

This command is non-blocking and immediately returns OK or an error as documented below. However, it can take a long time for the connection process to complete, and until it is complete, other power and connection control commands are rejected. Once the connection is established, a CONNECT event is issued. (For a blocking option, see the synchronous command 4.7.2 CONNECT »Establish a connection to an AWS IoT Core Endpoint«.

Returns:

4.7.8.1 `OK{EOL}`

The module has accepted the request and initiated the process to connect to AWS IoT Core. Note that the connection process can require a significantly long time.

4.7.8.2

A CONNECT event is generated when the process is completed or terminated with an error. A hint code is provided as the event parameter (with the same interpretation provided in 4.7.2.2 for the CONNECT ERR14 response). In case of success, the hint-code will be 0.

4.7.8.3

If the ExpressLink module is already connected, issuing a CONNECT! command will immediately produce a CONNECT event with a success hint code (0).

4.7.8.4

In case of a connection failure, the ExpressLink module will keep a timestamp of the event. This will be used to ensure that a subsequent (repeated) connection request will comply with the correct backoff timing limits. If the request from the host is repeated too soon after the previous attempt (a shorter interval than the prescribed minimum backoff time) the ExpressLink module will produce a CONNECT event with the Backoff hint code. Delays will increase according to the backoff algorithm until a successful connection is established.

4.7.8.5 `ERR25 NOT ALLOWED{EOL}`

The device is in CONFMODE or a CONNECT! command is already in progress.

5 Messaging

5.1 Messaging topic model

The ExpressLink messaging system relies on a list of topics defined in the configuration dictionary (see Table 2 - Configuration Dictionary Persistent Keys). Each topic is assigned an index that can be used to dereference the assigned string value. Index 0 has a special meaning, while all other index values up to an implementation-specific maximum index can be used by the host to define additional topics. Messaging topics defined in this list are managed independently from other topics eventually used by ExpressLink to handle Jobs, OTA, and shadow updates.

5.1.1.1

Topic Index 0 is reserved as a catch-all for messages that do not match other existing topics. An attempt to send or subscribe to a topic with index 0 will return `ERR7 OUT OF RANGE`.

5.1.1.2

Topic *Index{MaxTopic}* is an implementation detail documented in the module manufacturer's datasheet.

5.1.2 Topic usage rules

Topics are defined to be compatible with the MQTT 3.1.1 standard

5.1.3 SEND*[#] message* »Publish msg on a topic selected from topic list«

Send a message on a topic provided in the configuration dictionary. The configuration parameter QoS value (see Table 3 - Configuration dictionary non-persistent keys) at the time the command is issued determines the applicable Quality of Service (only QoS levels 0 and 1 are supported!).

Where:

[#]

The index of a topic in CONFIG dictionary (1..MaxTopic).

message

The message to publish (string).

Returns:

5.1.3.1 OK*{EOL}*

If the message is sent successfully, then the module returns 'OK'.

Example 1:

```
AT+SEND2 Hello World    # Publish 'Hello World' on Topic2
OK                      # The message will be sent
```

5.1.3.2 ERR6 NO CONNECTION

If no connection has been made, then the module returns 'NO CONNECTION'.

Example 2:

```
AT+SEND1 Hello World    # Publish Hello World on Topic1
ERR6 NO CONNECTION      # A connection has not been established
```

5.1.3.3 ERR7 OUT OF RANGE

If the supplied topic index is larger than the maximum allowed topic number, the module returns 'OUT OF RANGE'.

Example 3:

```
AT+SEND99 Hello World      # Publish Hello World on Topic99
ERR7 OUT OF RANGE    # Topic 99 is not within the available range of topics for this
 device
```

5.1.3.4 ERR8 PARAMETER UNDEFINED

If the supplied topic index points to a topic entry that has not been defined (empty), the module returns 'PARAMETER UNDEFINED'.

Example 4:

```
AT+CONF Topic3={EOL}      # Define Topic3 as empty
OK

AT+SEND3 Hello World      # Publish Hello World on Topic3
ERR8 PARAMETER UNDEFINED  # The selected topic was undefined
```

5.1.4 GET »Request next message pending on any topic«

Retrieve the next message received in the order of arrival.

Returns:

5.1.4.1 OK1*{separator}{topic}{EOL}{message}{EOL}*

If a message is available on any topic, the module responds with 'OK' followed by the topic and the message.

Example:

```
AT+GET            # poll for messages received on any topic
OK1 data{EOL}     # a message was received from topic 'data' (expect another line)
Hello World{EOL}  # the actual message received
```

5.1.4.2 OK*{EOL}*

If no message was received on any topic, the module responds with 'OK' followed by *{EOL}*.

5.1.5 GET0 »Request next message pending on an unassigned topic«

Retrieve the next message received on a topic that was not in the topic list. This acts as a catch-all option and can be useful when the host subscribes to a topic then modifies the topic string in the configuration dictionary without first unsubscribing. This can also be used in combination with the AWS IoT Device Shadow features (see entry 8.2.1.3 under section 10.2 AWS IoT Device Shadow).

Note that the response to this command always produces two output lines, an exception to the general format defined in 4.6.1 General response formats.

Returns:

5.1.5.1 OK1*{separator}{topic}{EOL}{message}{EOL}*

Example:

```
AT+GET0           # poll for messages received on any unassigned topic
OK1 data{EOL}     # a message was received from topic 'data' (expect another line)
Hello World{EOL}  # the actual message received
```

5.1.5.2 OK*{EOL}*

If no message was received on any unassigned topic, the module returns 'OK' followed by *{EOL}*.

5.1.6 GET*[#]* »Request next message pending on the indicated topic«

Retrieve the next message received on a topic at the specified index (1..MaxTopic) in the topic list.

Returns:

5.1.6.1 OK*{separator}{message}{EOL}*

If a message is available on the indicated topic, the module responds with 'OK' followed immediately by the message.

Example:

```
AT+GET2          # select messages received on Topic2
OK Hello World   # a message received on the topic at index 2 in the list of topics
```

5.1.6.2 OK*{EOL}*

If a message is NOT available matching the requested topic, the module responds with 'OK' followed by *{EOL}*.

5.1.6.3 OK*{message}{EOL}*

Even if there is no active connection, a normal read from the message queue takes place and might return a valid message.

5.1.6.4 ERR7 OUT OF RANGE

If the supplied topic index is larger than the maximum allowed topic number, then the module returns 'OUT OF RANGE'.

5.1.6.5 ERR8 PARAMETER UNDEFINED

If the requested topic is not defined (empty), then the module returns 'PARAMETER UNDEFINED'.

5.1.6.6 Message queue overflow conditions

If the host fails to retrieve a message in time and so does not free up space and the buffer capacity is exceeded, an overrun occurs and *new messages arriving from the cloud may be*

lost. The condition will be reported as an OVERFLOW event (see Table 4 - ExpressLink event codes) and added to the event queue. It is then accessible to the host processor by means of the EVENT? command. If there is an overflow, the number of messages-received events in the queue will exceed the actual number of messages that are present. The depth of the message queue is an implementation detail that is documented in the module manufacturer's datasheet.

5.1.7 SUBSCRIBE*[#]* »Subscribe to Topic#«

The module subscribes to the topic and starts receiving messages. Incoming messages trigger events. The messages can be read with a GET*[#]* command.

Note that this is a stateless feature; the ExpressLink module will request a subscription to the MQTT broker, but will not retain information about its current state.

5.1.7.1 If a topic number (*[#]*) is provided, use the topic at the specified index.

> **Note**
>
> Sending a message to a topic to which a module is subscribed results in the broker sending a copy back to the module.

Example:

```
AT+CONF Topic1=sensor1/state
OK

AT+SUBSCRIBE1    # The module will subscribe to the topic sensor1/state
OK
```

Returns:

5.1.7.2 OK

The subscription request was sent to the MQTT broker for the topic specified in the configuration dictionary as Topic#.

5.1.7.3 ERR6 NO CONNECTION

If no connection has been made, then the module returns 'NO CONNECTION'.

5.1.7.4 ERR8 PARAMETER UNDEFINED

If the requested topic is not defined (empty), then the module returns 'PARAMETER UNDEFINED'.

5.1.7.5 ERR7 OUT OF RANGE

If the supplied topic index is larger than the maximum allowed topic number, then the module returns 'OUT OF RANGE'.

5.1.7.6 A SUBACK or SUBNACK event is generated when the request is accepted or rejected by the MQTT broker.

> **Warning**
>
> The host should not issue an UNSUBSCRIBE command immediately following a SUBSCRIBE command before the acknowledgment event is received. This might result in a race condition and unpredictable MQTT broker behavior.

5.1.7.7 If the topic referred to by a subscription is altered (AT+CONF), before an acknowledgment is received, the corresponding event is NOT generated.

5.1.7.8 If a new SUBSCRIBE command is issued for the same topic (before an acknowledgment is received), the previous acknowledgment event is NOT generated.

> **Note**
>
> When in the "staging" state (the device is connected to the staging account, see 4.7.1 CONNECT? »Request the connection status« for details) restrictive policies apply, including not being able to subscribe to topics that do not begin with the device's *ThingName*.
> An attempt to subscribe to such topics may result in the connection being immediately dropped.

5.1.8 UNSUBSCRIBE*[#]* »Unsubscribe from Topic#«

The device unsubscribes from the selected topic and stops receiving its messages/events.

Returns:

5.1.8.1 OK

A request to unsubscribe from the topic specified in the configuration dictionary as Topic# was sent.

> **Warning**
>
> The host should not issue an UNSUBSCRIBE command immediately following a SUBSCRIBE command before the acknowledgment event is received. This would result in a race condition and unpredictable MQTT broker behavior.

5.1.8.2 ERR6 NO CONNECTION

If no connection has been made, then the module returns 'NO CONNECTION'.

5.1.8.3 ERR8 PARAMETER UNDEFINED

If the requested topic is not defined (empty), then the module returns 'PARAMETER UNDEFINED'.

5.1.8.4 ERR7 OUT OF RANGE

If the supplied topic index is larger than the maximum allowed topic number, then the module returns 'OUT OF RANGE'.

Example:

```
AT+CONF Topic1=sensor1/state
OK

AT+SUBSCRIBE1      # The module will subscribe to topic sensor1/state
OK
...
AT+UNSUBSCRIBE1    # The module will unsubscribe from topic sensor1/state
OK
```

6 Configuration Dictionary

The configuration dictionary is a key-value store containing all the options necessary for the proper functioning of ExpressLink modules.

> **Note**
>
> All keys are case sensitive.

Configuration key-value pairs listed in Table 2 are meant to be long lived (persist) throughout the life of the application and so are stored in non-volatile memory. Note that these key-value pairs have factory preset values, and can be read only or write only.

Table 2 - Configuration Dictionary Persistent Keys

Configuration Parameter	Type	Initial Value	Factory Reset	Buff Size	Description
About	R	Vendor - Model	N	64	A concatenation of Vendor name and Model name (also see 11.1.5.3).
Version	R	X.Y.Z [suffix]	N	32	The specific module firmware version (also see 11.1.5.3). Note: an optional alphanumeric suffix may be present.
TechSpec	R	TechSpec version	N	16	The Technical Specification version this model implements (for example 'v0.6', 'v1.1.2').
ThingName	R	UID	N	64	The UID provided by the HW root of trust and present in the device certificate (also see

Configuration Parameter	Type	Initial Value	Factory Reset	Buff Size	Description
					11.1.3.1) 12.1.3 ExpressLink Birth Certificate).
Certificate	R	Device Birth Certificate	N	≥4KB	Device certificate used to authenticate with AWS cloud, signed by the manufacturer CA (also see 12.1.3 ExpressLink Birth Certificate).
CustomName	R/W	{empty}	Y	≥128	Custom Product Name, can be set by the host.
Endpoint	R/W	Staging account endpoint	Y	≥128	The endpoint of the AWS account to which the ExpressLink module connects (also see 21.3.2 ExpressLink onboarding states and transitions).
RootCA	R/W	AWS root CA	N	≥4KB	The server root certificate that will be used to authenticate the cloud Endpoint (also see 9.12 Server Root Certificate Update).
ShadowToken	R/W	ExpressLink	Y	64	The default client-token that will be used to mark Device Shadow updates.
DefenderPeriod	R/W	0	Y	≥8	The Device Defender upload period in seconds. (0 indicates the service is disabled.)

Configuration Parameter	Type	Initial Value	Factory Reset	Buff Size	Description
HOTAcertificate	R/W	{empty}	Y	≥4KB	Host OTA certificate (see 9.10 Host OTA Signature Verification).
OTAcertificate	W	Vendor OTA Certificate	N	≥4KB	Module OTA certificate. Vendor and Model specific (see 9.5 Module OTA signature verification). (Wi-Fi modules only.)
SSID	R/W	{Empty}	Y	32	SSID of local router (Wi-Fi modules only).
Passphrase	W	{Empty}	Y	64	Passphrase of local router (Wi-Fi modules only).
APN	R/W	{default}	Y	128	Access Point Name (Cellular modules only).
KeepAlive	R/W	{default}	N	>=8	The MQTT keep alive period in seconds.
LWTConfig	R/W	{default}	N	>=128	A JSON object describing the configuration of the Last Will and Testament Topic and options in case of unexpected connectio n loss. Expected keys are: “topic”, “qos”, and “retain”. Example: {“topic”: “abcde”, “qos”:0, “retain”:0}

Configuration Parameter	Type	Initial Value	Factory Reset	Buff Size	Description
LWTMessage	R/W	{default}	N	>=128	Message to publish as Last Will and Testament in case of unexpected connectio n loss. This can can be any valid MQTT message.

The additional configuration parameters in Table 3 are non-persistent. They are re-initialized at power up, and following any reset event. The host processor might have to re-configure them following a reset and (possibly) a *deep sleep* awakening (depending on the implementation).

Table 3 - Configuration dictionary non-persistent keys

Configura tion Parameter	Type	Initial Value	Buff Size	Description
QoS	R/W	0	1	QoS level selected for SEND commands
Topic1	R/W	{Empty}	≥128	Custom defined topic 1
Topic2	R/W	{Empty}		Custom defined topic 2
...				
Topic<Max Topic>	R/W	{Empty}		Custom defined topic MaxTopic
EnableSha dow	R/W	0	1	0 - disabled, or 1 - enabled
Shadow configuration parameters (required only by modules that support the Shadow feature, see 10.2 AWS IoT Device Shadow)				
Shadow1	R/W	{Empty}	64	Custom defined named shadow

Configuration Parameter	Type	Initial Value	Buff Size	Description
...				
Shadow<MaxShadow>	R/W	{Empty}		Custom defined named shadow
BLE configuration parameters (required only by modules that support BLE host control, see 13 Bluetooth Low Energy (BLE))				
BLECentral1	R/W	{Empty}	≥ 128	GAP Central discovery/ connect configurations.
BLECentral2	R/W	{Empty}	≥ 128	
...				
BLECentral<MaxBLECentral>	R/W	{Empty}	≥ 128	
BLEGATT1	R/W	{Empty}	≥ 128	GATT Characteristic definitions (JSON).
BLEGATT2	R/W	{Empty}	≥ 128	
...				
BLEGATT<MaxBLEGatt>	R/W	{Empty}	≥ 128	
BLEPeripheral	R/W	{Empty}	≥ 128	GAP Peripheral advertising configuration.
BLEAllowList	R	{Empty}	≥ 128	BLE Allow list
BLEBondLIstl	R	{Empty}	≥ 128	BLE Bonding list

6.1 Data values referenced

6.1.1.1 Maximum key length is 16 characters

A parameter name (key) can be from 1 to 16 characters.

6.1.1.2 ERR9 INVALID KEY LENGTH

If a parameter name (key) exceeds 16 characters, the ExpressLink module returns 'ERR9 INVALID KEY LENGTH'.

6.1.1.3 Valid key characters are 0-9, A-Z, a-z

A parameter name (key) may only contain alphanumeric characters.

6.1.1.4 ERR10 INVALID KEY NAME

If a non-alphanumeric character is used in a key name, then the ExpressLink module returns 'ERR10 INVALID KEY NAME'.

6.1.1.5 ERR11 UNKNOWN KEY

If the parameter name (key) is not found in Table 2 - Configuration Dictionary Persistent Keys or Table 3 - Configuration dictionary non-persistent keys, then the module returns 'ERR11 UNKNOWN KEY'.

6.1.1.6 ERR4 PARAMETER ERROR

If the parameter (value) length exceeds the buffer size as defined in Table 2 - Configuration Dictionary Persistent Keys or Table 3 - Configuration dictionary non-persistent keys.

6.2 Dictionary data access - CONF command

6.2.1 CONF KEY=*{value}* »Assignment«

Assign a value to a configuration parameter present in the configuration dictionary. (See 9.11.2 CONF? *{certificate} pem* »Special certificate output formatting option«).

Returns:

6.2.1.1 OK*{EOL}*

If the write is successful, then the module returns 'OK'.

Example:

```
AT+CONF Topic1={EOL}  # Assign the empty string to Topic 1
OK
```

6.2.1.2 ERR9 INVALID KEY LENGTH

If the key is too long, then the module returns 'INVALID KEY LENGTH'.

6.2.1.3 ERR10 INVALID KEY NAME

If the key uses incorrect characters, then the module returns 'INVALID KEY NAME'.

6.2.1.4 ERR11 UNKNOWN KEY

If the key is not present in the dictionary, then the module returns 'UNKNOWN KEY'.

Example:

```
AT+CONF VERSION=1.0   # Incorrect capitalization
ERR11 UNKNOWN KEY     # The key is not recognized as spelled
```

6.2.1.5 ERR12 KEY READONLY

Some keys are read-only and cannot be written. If the key cannot be written to, then the module returns 'KEY READONLY' (for example, ThingName, Certificate, About).

Example

```
AT+CONF Version=1.0   # Attempt to manually modify the Version parameter
ERR12 KEY READONLY
```

6.2.1.6 ERR23 INVALID SIGNATURE

When updating a certificate (for example, Certificate, OTAcertificate, HOTAcertificate) if a required signature verification failed, then the module returns 'INVALID SIGNATURE'. (See 9.11 Host OTA certificate update for more detail on the signature verification rules that apply to different types of certificates.)

6.2.2 CONF? *key* »Read the value of a configuration parameter«

Returns:

6.2.2.1 OK *{value}*

If the read is successful, then the module returns 'OK'.

6.2.2.2 ERR9 INVALID KEY LENGTH

If the key is too long, then the module returns 'INVALID KEY LENGTH'.

6.2.2.3 ERR10 INVALID KEY NAME

If the key uses incorrect characters, then the module returns 'INVALID KEY NAME'.

6.2.2.4 ERR11 UNKNOWN KEY

If the key is not present in the system, then the module returns 'UNKNOWN KEY'.

6.2.2.5 ERR13 KEY WRITEONLY

Some keys are write-only and cannot be read. If the key cannot be read, then the module returns 'KEY WRITEONLY'.

Example:

```
AT+CONF? Passphrase
ERR13 KEY WRITEONLY
```

7 Status dictionary

The status dictionary has been removed since rev 0.6 of the ExpressLink Specifications. See 4.7.1 CONNECT? »Request the connection status« for the current connection status. See 9.2 OTA commands for the current OTA process status. The Overflow mechanism does not have a corresponding status bit associated anymore.

7.1 State commands

The *STAT* command is not available since version 0.6

8 Event handling

8.1 Introduction

Events are generated by internal or external occurrences that require the host controller's attention, such as the arrival of messages on one of the subscribed topics, but also error conditions that reflect an unexpected change in the module's internal state.

Events are appended to the module *event queue*. From there the host can fetch them in order of arrival (FIFO) by polling the event queue periodically (at regular intervals), or, if the EVENT pin is connected, after detecting a (rising edge) signal on the pin.

8.1.1.1 The event queue depth is an implementation dependent parameter that must be documented by the vendor in the module datasheet.

8.1.1.2 The EVENT pin is asserted (HIGH) when the event queue contains one or more events. The EVENT pin is automatically de-asserted as soon as the host processor has emptied the event queue.

8.1.1.3 When the event queue is full, and a new event occurs, the oldest event is discarded (circular buffer).

8.2 Event handling commands

8.2.1 EVENT? »Request the next event in the queue«

Returns:

8.2.1.1 `OK` *`[{event_identifier} {parameter} {mnemonic [detail]}]{EOL}`*

When the queue contains one or more events, the module response returns the first event in order of arrival (FIFO). See Table 4 below for the predefined event types.

8.2.1.2 `OK`*`{EOL}`*

Acknowledgement of the command. No events where pending, a timer is started counting down *time* seconds.

The following table contains the definition of common event identifiers and error codes implemented by all ExpressLink modules; they should be considered reserved:

Table 4 - ExpressLink event codes

Event Identifier	Parameter	Mnemonic	Description
0	-	TIMEOUT	A timout occurred.
1	Topic Index	MSG	A message was received on the topic #.
2	0	STARTUP	The module has entered the active state.
3	0	CONLOST	Connection unexpectedly lost.
4	0	OVERRUN	Receive buffer Overrun (topic in detail).
5	0	OTA	OTA event (see the OTA? command for details).
6	Connection Hint	CONNECT	A connection was established or failed.
7	0	CONFMODE	CONFMODE exit with success.
8	Topic Index	SUBACK	A subscription was accepted.
9	Topic Index	SUBNACK	A subscription was rejected.
10	Topic Index	PUBACK	A QoS1 PUBACK was received.
11..19	-	-	RESERVED
20	Shadow Index	SHADOW INIT	Shadow[Shadow Index] interface was initialized successfully.
21	Shadow Index	SHADOW INIT FAILED	The SHADOW[Shadow Index] interface initialization failed.

Event Identifier	Parameter	Mnemonic	Description
22	Shadow Index	SHADOW DOC	A Shadow document was received.
23	Shadow Index	SHADOW UPDATE	A Shadow update result was received.
24	Shadow Index	SHADOW DELTA	A Shadow delta update was received.
25	Shadow Index	SHADOW DELETE	A Shadow delete result was received.
26	Shadow Index	SHADOW SUBACK	A Shadow delta subscription was accepted.
27	Shadow Index	SHADOW SUBNACK	A Shadow delta subscription was rejected.
28...39	-	-	RESERVED
40	0	BLE CONNECTED	A BLE Connection was established peripheral role.
41	0 or Hint Code	BLE DISCOVER COMPLETE	0 for successful; >0 vendor defined Hint Codes.
42	0 or Central Index	BLE CONNECTION LOST	Connection was terminated or 0 if peripheral role.
43	GATT Index	BLE SUBSCRIBE START	Subscription started on BLEGATT# while on peripheral mode.
44	GATT Index	BLE SUBSCRIBE STOP	Subscription terminated on BLEGATT# while on peripheral mode.

Event Identifier	Parameter	Mnemonic	Description
45	GATT Index	BLE READ REQUEST	Read operation requested at BLEGATT# while on peripheral mode.
46	GATT Index	BLE WRITE REQUEST	Write operation requested at BLEGatt# while on peripheral mode.
47	Subscription Index	SUBSCRIPTION RECEIVED	Subscription was received on BLECentral# connection.
48	BLECentral#	BLE PAIR	Pairing event (see Table 3 - Configuration dictionary non-persistent keys)
49	Auth Index	BLE AUTHORIZE	Authorizing event (see 13.3.6 BLE AUTH*[#] [0/1]* »Authorize access to a characteristic«) Auth Index encoding is defined below in *8.2.1.5*.
50	Hint code	BLE EXCEPTION	An exception occurred in the BLE stack. Codes are vendor specific.
≤ 999	-		RESERVED.
≥1000	-		Available for custom implement ation.

8.2.1.3 Sleep, reset, and factory reset commands

Clears all pending events.

8.2.1.4 Subscription Indexes

combine a BLECentral Index and BLEGATT Index according to the following formula:

Subscription Index = BLECentral Index *100 + BLEGATT Index

8.2.1.5 Authorization indexes combine a BLEGATT Index with an Operation Value

combine a BLECentral Index and BLEGATT Index according to the following formula:

Subscription Index = BLECentral Index *100 + BLEGATT Index

Auth Index = BLEGATT Index *100 + Operation Value

where Operation Value is:

- 0 for Read
- 1 for Read

8.2.1.6 Asynchronous generation of BLE EXCEPTION

A BLE EXCEPTION is generated asynchronously by the BLE stack and the Hint Codes provided must be documented by the vendor in the specific module datasheet.

8.2.2 EVENT! *time* »Wait (blocking) for the next event or timeout«

This is a synchronous variant of the EVENT command meant to allow efficient event driven operation in systems that employ only the serial interface, without an additional event pin. The *time* parameter is mandatory and indicates the maximum amount of time (in seconds) the module will be waiting for an event. Setting *time* to 0 will set the timer to the maximum value allowed for the module (see 4.6.2 Response timeout).

The implementation of this command is optional.

Returns:

8.2.2.1 OK *{event identifier} {parameter} {mnemonic [detail]}{EOL}*

When the queue already contains one or more events, the module response returns immediately the first event in order of arrival (FIFO) identically to the asynchronous version of the command (EVENT?). See Table 4 - ExpressLink event codes for the decoding of event codes and corresponding event types. This ends the command, and a timer is not enabled.

If there is no event pending in the event queue, returns only an immediate acknowledgement.

8.2.2.2 OK *{EOL}*

Acknowledgement of the command. No events where pending, a timer is started counting down *time* seconds.

8.2.2.3 ERR4 PARAMETER ERROR*{EOL}*

If the time parameter is greater than the maximum duration allowed for any ExpressLink command (see 4.6.2 Response timeout).

When a new event is generated (for example: message received or BLE notification) the device returns the event identifier prefixed by a sequence of 4 white spaces {NewLine}.

8.2.2.4 \n\n\n\nOK *{event identifier} {parameter} {mnemonic [detail]}{EOL}*

An event was generated and reported, and the timer stopped as the command completed. The prefixing white spaces are meant to allow the host processor serial interface (UART) to resynchronize its clock, a feature that allows the host processor to enter a low power sleep mode and await for a new character to wake up. Note: it is common for such feature to cause the first character (or pair of) to be corrupted

8.2.2.5 To give the host time to prepare for a low power sleep phase and prevent race conditions, the first event generated is guaranteed to be reported by the ExpressLink module no sooner than 500ms after the acknowledgement response is delivered.

Should an event not be generated before the timer expires, a TIMEOUT event is generated instead.

8.2.2.6 \n\n\n\nOK *{EOL}*

The timer has expired, and an event is not generated. The event does not contain any parameter or description to reduce the length of the response and allow the host to re-issue immediately a new EVENT! command to maximize the power saving.

Example 1:

```
AT+EVENT! 0{EOL}    #Wait for an event up to the longest possible time (120s)
OK{EOL}       # Acknowledgement, no events pending, start timer

... after >500ms …

\n\n\n\nOK 1 5{EOL} #A message arrived on Topic #5, timer stopped
```

```
… or after 120s …

\n\n\n\nOK 0{EOL}  #No event arrived and the timer has expired
```

Example 2:

```
AT+EVENT! 10{EOL}        # wait up to 10s for an event
OK 1 6{EOL}  #An even was already in the queue, a message on Topic # 6
```

8.3 Diagnostic commands

8.3.1 DIAG *{command} [optional parameters]* »Perform a diagnostic command«

A number of custom diagnostic commands can be added to assist the developer in their debugging efforts. These commands are vendor and model specific. See the manufacturer's datasheet for specific details.

Diagnostic commands are not checked as part of the ExpressLink qualification test suite.

Diagnostic commands must be documented in the vendor device datasheet.

9 ExpressLink module Updates

ExpressLink modules natively support firmware updates utilizing the AWS IoT OTA service and, locally, using Over the Wire (OTW) updates.

To support the OTA feature, ExpressLink modules provide additional bulk storage space (non-volatile memory). The amount of non-volatile memory available is sufficient to store at least two full copies of the ExpressLink module's own firmware image – a current known-good copy and a new copy. This is intended to provide a backup in case of a fatal failure during the update process.

When an ExpressLink firmware update job is triggered (using the AWS IoT OTA console), the update process begins and takes place in five steps:

1. Without disrupting the Host processor communication, the module starts receiving chunks of the new firmware image.
2. Each chunk is checked for integrity and acknowledged, retried as necessary, and stored in bulk memory.
3. When all chunks are reassembled in bulk memory, the module performs a final signature check.

4. Only if successfully verified, the module notifies the Host processor.
5. *Upon receiving an explicit request*, the ExpressLink module initiates a reboot.

This process provides two types of security/safety assurance to the user:

- It makes sure that only valid memory images are accepted.
- The potentially disruptive process of rebooting is performed *in agreement with the host processor* to avoid impacting the overall product functionality and potential safety hazards.

The host processor is notified of the module's OTA ready/pending status by means of an event. (See the EVENT? command.)

The host processor can poll the OTA process state at any time using the OTA? Command. (See 9.2 OTA commands.)

> **Note**
>
> The OTA service is supported only when the device is in the *onboarded* state (see 21.3.2 ExpressLink onboarding states and transitions), that is, only when the module is connected to the customer's AWS account.

9.1 ExpressLink module support of Host Processor OTA

ExpressLink modules are designed to support Host processor updates Over the Air (HOTA). This is done in a shared responsibility model in collaboration with the host processor. The Bulk Storage memory capacity of the module might be shared between the module and host OTA images, so that only one of the two is *guaranteed* to be supported at any time, although manufacturers can choose to differentiate their products by offering a larger amount of non-volatile memory. Consult the manufacturer's datasheet to verify the amount of memory available on a specific model.

The HOTA feature is not limited to supporting only host processor firmware images but can also be used to transport, stage, and verify the delivery of any large payload including pictures, audio files, or any binary blobs that may potentially contain multiple files of different natures.

The mechanism utilized to trigger and perform the transfer of host processor images makes use of the same underlying services as the module OTA (namely, AWS IoT Jobs and AWS IoT OTA). It

utilizes a collaborative model based on the paradigm of a *mailbox*. ExpressLink devices act as the recipient of *envelopes* meant for the host. They can verify the envelope's integrity (checksum) and authenticity (signature) before notifying the host by raising a flag (event). It is up to the host to periodically check for flags, and when ready, to retrieve the contents of the mailbox. ExpressLink devices, much like actual mailboxes, are not concerned with the nature of the contents of the envelopes. Once the envelope is retrieved, and the flag lowered, they are ready (empty) to receive more mail. Successive attempts to deliver more updates to a host processor will be NACKed until the host either retrieves the update or rejects it and clears the flag without retrieving the contents.

The communication between the host processor and the ExpressLink module required to deliver an OTA payload is represented in the following diagram:

9.1 ExpressLink OTA/HOTA process

ExpressLink module	Host Processor
Receives an event indicating an OTA request and generates an event (also raising the EVENT Pin).	
	EVENT? polls the event queue.
Returns OK OTA indicating an OTA event.	
	OTA? checks the OTA state.
Returns an OTA or HOTA ready state.	
if OTA ready	
	When safe, issue an OTA APPLY command to allow the ExpressLink module to update its firmware and reboot (or OTA FLUSH to abort).
If HOTA ready	Retrieve the payload in chunks of appropriate size.
	READ 1024 – Requests the first chunk of payload data.

ExpressLink module	Host Processor
Delivers first chunk of payload data and advances pointer.	
The process repeats until the entire payload is transferred to the host processor. At any point, the Host processor can request a pointer reset or terminate the process altogether.	
The module returns a 0 sized chunk, indicating transfer complete.	
	CLOSE – indicate to the ExpressLink module that the buffer can now be freed and the process was completed successfully.
The ExpressLink module returns a Job complete notification to the AWS IoT OTA service.	

The Host processor is not required to retrieve the entire payload at once, nor to follow a strictly sequential process, the fetching pointer can be moved (seek) to allow random access to the payload contents. Also, the size of the chunks retrieved by the Host processor is independent from the chunking performed during the image download by the module. Instead, this is intended to be the most convenient value depending on the host processor's serial interface buffer size, the Host processor's own (flash) memory page size, and/or binary format decoding needs (for example, INTEL HEX...). Consequently, the host processor can choose the reboot directly from the ExpressLink module host OTA memory or can choose to transfer only parts of the payload to be consumed by other subsystems as necessary.

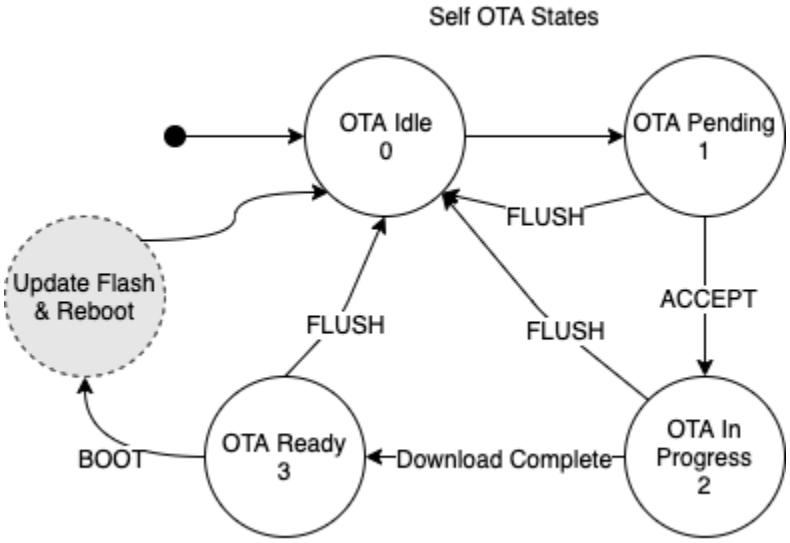


Figure 3 - ExpressLink module OTA state diagrams

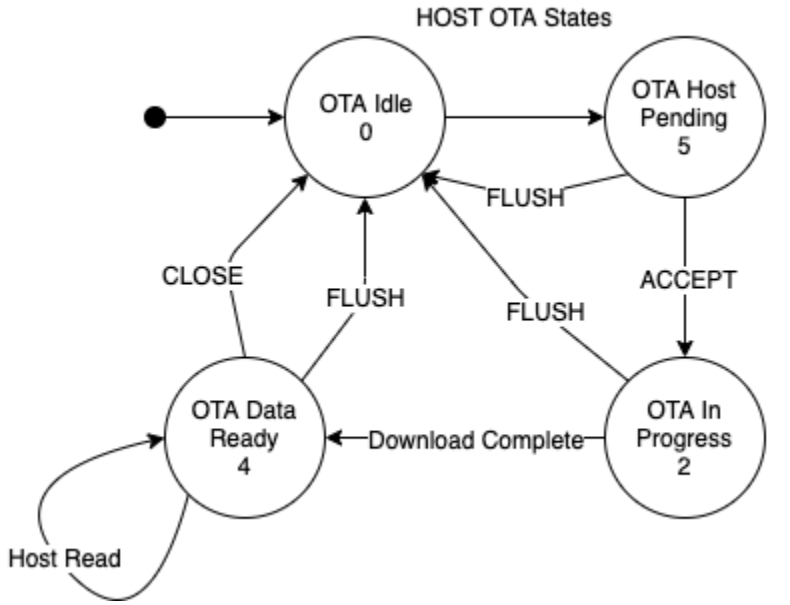


Figure 4 - ExpressLink Host OTA state diagrams

The serial interface commands involved in the implementation of the OTA and Host OTA features are summarized here:

9.2 OTA commands

9.2.1 OTA?[DOC] »Fetches the current state of the OTA process«

Fetch the current state of the OTA process. If the device is in one of the OTA pending states (1 and 3) and the optional parameter “DOC” is provided, the entire *Jobs OTA document* (a JSON object string) is provided describing in detail the incoming payload to allow the host processor determine when to accept or reject the operation. This command is intended to be used following an OTA event (see Table 4) or polling periodically.

Returns:

9.2.1.1 OK *{OTA code} {description} [JSON string] {EOL}*

If a valid request was pending and the host is allowing the OTA operation to commence, the host returns 'OK'.

9.2.1.2 *OTA code* see Table 5 - OTA codes and descriptions

Indicates OTA process states as illustrated in Figure 3 and Figure 4.

9.2.1.3 *Description*

This is a concise, humanly readable description of the state.

9.2.1.4

If in a pending state (OTA codes 1 and 3), and the optional ‘DOC‘ parameter is present, the entire OTA Jobs Document is presented to the host as a JSON object string to provide more detailed information on the nature of the incoming payload (For example, packageName, workingDirectory, launchCommand, fileName).

9.2.2 OTA codes

Table 5 - OTA codes and descriptions

0	IDLE	No OTA in progress.
1	OTA PENDING	A new module OTA update is being proposed. The host can optionally inspect the Jobs OTA DOC (JSON) and decide

		to accept or reject it (see OTA ACCEPT or OTA FLUSH).
2	HOTA PENDING	A new Host OTA update is being proposed. The host can inspect the provided metadata (JSON) details and decide to accept or reject it. (See OTA ACCEPT or OTA FLUSH).
3	OTA IN PROGRESS	The download and or verificat ion steps are not completed yet. Contents of the OTA buffer cannot be accessed at this time.
4	READY TO REBOOT	A new module firmware image has arrived. The signature has been verified and the ExpressLink module is ready to reboot. When ready, at the most appropriate time (depending on application state) the host can issue an OTA APPLY command to reboot the ExpressLink module and apply the new firmware version or abort the update with OTA FLUSH.

5	HOTA AVAILABLE	A new host image has arrived. The signature has been verified (if requested) and the contents are available for the host to retrieve using the OTA READ and OTA SEEK commands.

Example 1:

```
AT+OTA?    # check the OTA status
OK 3 HOTA PENDING     # a Host OTA file download was proposed.
```

Example 2:

```
AT+OTA? DOC       # check the OTA status and request a full JOBS DOC if available
OK 1 OTA PENDING {... "fileName": "FWupdate v2.5.7", ...} {EOL}   # a module OTA
 firmware update is pending
```

Note

The host has the ultimate say to allow this update to proceed (downloading) by sending the OTA ACCEPT command, or to reject it immediately (if it is deemed incompatible with the host version) by sending the OTA FLUSH command.

9.2.3 OTA ACCEPT »Allow the OTA operation to proceed«

The host allows the module to download a new image for the module or the host OTA.

Returns:

9.2.3.1 OK*{EOL}*

If a valid request was pending and the host is allowing the OTA operation to commence, the host returns 'OK'.

9.2.3.2 ERR21 INVALID OTA UPDATE

If no OTA update is pending, the host returns 'INVALID OTA UPDATE'.

Example:

```
AT+OTA?                   # Check the OTA state
OK 0 IDLE                 # No pending OTA request (host or module)
AT+OTA ACCEPT             # accept the OTA download
ERR21 INVALID OTA UPDATE  # No OTA pending, nothing there for the host to accept
```

9.2.4 OTA READ *#bytes* »Requests the next # bytes from the OTA buffer«

The read operation is designed to allow the host processor to retrieve the contents of the OTA buffer starting from the current position (0 initially). The # bytes must be provided as a decimal value.

Returns:

9.2.4.1 OK *{count} ABABABAB... {checksum}*

The byte count is expressed in hex (from 1 to 6 digits), each byte is then presented as a pair of hex digits (no spaces) for a total of count*2 characters followed by a checksum (4 hex digits).

The reading pointer is advanced by *count* bytes. *Count* can be less than requested or 0 if the end of the payload was reached. If the *count* is zero, the data and checksum portion are omitted.

The maximum *number of bytes* a module can read (MaxOTARead) is implementation specific and will be declared by the manufacturer in the device datasheet. If the requested value is greater than the maximum supported by the module, the module will return the maximum value possible.

The *checksum* is provided as a 16-bit (4 digit hex value) computed as the sum of all data (byte) values returned (modulo 2^16).

Example 1:

```
AT+OTA READ 2       # request 2 bytes of data from the OTA buffer
OK 02 ABAB CK
```

Example 2:

```
AT+OTA READ 256     # request 256 bytes of data from the OTA buffer
OK 100 ABABAB....AB CK
```

Example 3:

```
AT+OTA READ 16      # request 16 bytes of data from the OTA buffer
OK 0C ABABAB.. CK   # reached the end of the OTA buffer, only 12 bytes were
 available
```

9.2.4.2 ERR19 HOST OTA IMAGE NOT AVAILABLE

The module returns an error if the OTA buffer is empty, or if it is in use and the download or signature verification processes have not been completed. The host processor should first check the OTA status using the OTA? command.

9.2.5 OTA SEEK *{address}* »Moves the read pointer to an absolute address«

This command moves the read pointer to the specified address in the OTA buffer. If no address is specified, the read pointer is moved back to the beginning (0). The # bytes must be provided as a decimal value.

Returns:

OK *{address}*

If the pointer was successfully moved the module returns 'OK'. The address is returned in hex (from 1 to 6 digits).

Example 1:

```
AT+OTA SEEK 1024    # move the read pointer to location 1024
OK 400
```

Example 2:

```
AT+OTA SEEK         # move the read pointer back to location 0
OK 0
```

9.2.5.1 ERR20 INVALID ADDRESS

If the address provided was out of bounds (> OTA buffer content size), then the module returns 'INVALID ADDRESS'.

9.2.5.2 ERR19 HOST OTA IMAGE NOT AVAILABLE

An error is issued if the OTA buffer is empty or in use and the download or signature verification processes have not been completed. The host processor should first check the OTA status using the OTA? command.

9.2.6 OTA APPLY »Authorize the ExpressLink module to apply the new image«

When an ExpressLink module OTA image has been downloaded and is ready to be applied, the host processor is notified by an event. When it is appropriate (safe for the application), the host processor should activate the boot command to update its own firmware version. Upon completion, the OTA buffer is emptied, making it available for additional OTA operations. The OTA status is cleared.

Returns:

9.2.6.1 OK*{EOL}*

The module has initiated a boot sequence.

9.2.6.2 ERR19 HOST OTA IMAGE NOT AVAILABLE

An error is returned if the OTA buffer is empty or it is in use and the download or signature verification processes have not been completed. The host processor should first check the OTA status using the OTA? command.

9.2.6.3 ERR21 INVALID OTA UPDATE

The module is unable to apply the new module images (integrity issue or version incompatibility).

9.2.6.4 ERR23 INVALID SIGNATURE

The new image signature check failed.

9.2.6.5 Upon successful completion of the boot sequence, the ExpressLink module communicates the new status and firmware revision number to the AWS IoT OTA service.

9.2.6.6 The event queue is emptied and a STARTUP event is generated to inform the host processor that the process has completed.

9.2.6.7 The host processor should expect all state and configuration parameters of the module to be reset in a way similar to a Reset command (although additional changes may apply and are implementation and firmware version dependent).

9.2.7 OTA CLOSE »The host OTA operation is completed«

The host's use of the OTA buffer is terminated and the buffer can be released. The OTA flag is cleared and the operation is reported to the AWS IoT Core as successfully completed.

Returns:

9.2.7.1 `OK`*{EOL}*

When the ExpressLink module returns 'OK', it indicates that the command was received correctly, but the actual run sequence (that requires a handshake with the AWS IoT OTA service) can still fail later. In that case, an event is generated to inform the host and help diagnose the problem.

9.2.8 OTA FLUSH »The contents of the OTA buffer are emptied«

The OTA buffer is immediately released. The OTA flag is cleared. Any pending OTA operation is stopped. The OTA operation is reported as failed.

Returns:

9.2.8.1 `OK`*{EOL}*

When the ExpressLink module returns 'OK', it indicates the command was received correctly, but the actual run sequence (that requires a handshake with the AWS IoT OTA service) can still fail at a later time. In that case, an event will be generated to inform the host and help diagnose the problem.

9.3 OTA update jobs

OTA updates are meant to be issued by the customers' fleet managers through the AWS Cloud console using the AWS IoT OTA Update Manager service. This is built upon the AWS IoT Jobs

service and is designed to allow customers to send updates to selected groups of devices in a fleet. (For more information, see Prerequisites for OTA updates using MQTT in the *AWS FreeRTOS User Guide*.)

The OTA service has the following basic requirements:

- Each device must be associated with a policy allowing it to publish and subscribe to the AWS reserved topics for `streams/*` and `jobs/*`. This policy will be automatically added to the thing created in the staging account (see the JITP template) and later moved to the customer's account using the AWS IoT API.
- Firmware updates and certificates for ExpressLink modules will only be provided and signed by the module manufacturer. Firmware updates and certificates for the host can be provided and signed by the customer/developer. They will be uploaded to an Amazon S3 bucket before the process is initiated.
- The customer will create an OTA update role to allow the service to operate in the account
- The operator initiating the update process must have an OTA User policy that authorizes them to operate the service.

The OTA Job creation can be instantiated from the AWS CLI or from the AWS IoT Console.

The OTA Jobs service is generic and can transfer (stream) any type of file to a selected group of devices. Metadata is provided by the user (in the form of a Jobs OTA document, created manually and submitted by the AWS CLI, or created automatically with the AWS IoT Console OTA interface), formatted as a JSON string and providing target devices with information (metadata) about the nature of the incoming OTA payload, signing method (if used) and a number of additional attributes (key-value pairs).

Specifically, ExpressLink devices will require the `fileType` attribute to be set to values according to Table 6.

Table 6 - Reserved OTA file type codes (0-255)

fileType	Reserved for	Signature	Certificate	Request Host Permission
101	Module firmware update	Signed	Module OTA	Y

fileType	Reserved for	Signature	Certificate	Request Host Permission
103	Module OTA certificate update	Signed[1]	Module OTA	N
107	Server Root certificate update	Signed[1]	Server Root	N
202	Host firmware update	Optional	Host OTA	N
204	Host OTA certificate update	Certificates are already hashed and signed, no additiona l signing is required.	Host OTA	N

[1] Not required if the HostCertificate parameter is empty (factory default).

These codes allow the ExpressLink modules that receive them to determine and initiate the corresponding module or host update processes described in this chapter. Different signing rules apply to each type of update/file and the certificates used for the validation of the signatures can themselves be updated.

9.4 Module OTA image signing

ExpressLink module manufacturers may create a new profile with the AWS Code Signing service for each ExpressLink module *model* they qualify and introduce to production. This profile will then be used exclusively to sign images before distributing them to their customer base (publishing them on a dedicated manufacturer support web page).

For a complete workflow detailing all steps required for the generation of signed image, see Creating an OTA update with the AWS CLI in the *FreeRTOS User Guide*.

ExpressLink manufacturers are free to choose any signature and hashing algorithms compatible with AWS IoT Core specifications to best match the cryptographic capabilities of their modules. Contact the module manufacturer or check the module manufacturer's datasheet for the algorithms used.

9.5 Module OTA signature verification

In order for ExpressLink modules to validate module OTA updates, they are pre-provisioned by the manufacturer with an OTA certificate that will be used automatically after download to ensure the payload integrity and authenticity.

9.6 Module OTA certificate updates

The certificates used for the module OTA signature validation (not to be confused with the module birth certificate used to authenticate with the AWS cloud) may be updated using the OTA mechanism or using the serial API:

- Module OTA certificate updates performed using OTA use the fileType code indicated in Table 5 - OTA codes and descriptions (Module OTA certificate update).
- Module OTA certificate updates performed using the AT+CONF command use the key *OTAcertificate*.

Example:

```
AT+CONF OTAcertificate=<x509.pem>[2]
```

[2] Some escaping required to accommodate newlines may be present in the certificate (.pem) file.

Returns:

9.6.1.1 `OK{EOL}`

The module returns 'OK' if the new certificate was valid.

9.6.1.2 `ERR23 INVALID SIGNATURE`

The module returns 'INVALID SIGNATURE' if the new certificate could not be verified.

9.6.1.6 `ERR26 INVALID CERTIFICATE`

The module returns 'INVALID CERTIFICATE' if the new certificate provided was invalid or corrupted.

9.6.1.3 The new certificate must be signed with the private key corresponding to the previous valid module OTA certificate.

9.6.1.4 Module OTA certificate updates performed using the OTA mechanism do not require the host to accept the update nor to control its run timing.

9.6.1.5 Module OTA certificates are NOT affected by a factory reset.

9.7 Module OTA override

As described in **9.1** ExpressLink OTA/HOTA process, the host processor is given ultimate control over the ExpressLink module firmware update process, including whether to accept or reject an incoming image, and control over when the process starts. While this mechanism is meant to prevent scenarios where host and module firmware versions could become incompatible or the module reboot could happen at an inconvenient time (possibly affecting the device functional safety), we must consider cases where a poorly behaved (or too basic) host application might *indefinitely* prevent an ExpressLink module from being updated to fix a critical bug or an identified security threat. To this end, an additional piece of metadata that uses the attribute <force:YES> will be provided to bypass the host control and to activate an immediate module firmware update.

> **Note**
>
> A forced module OTA update cleans the module OTA buffer (bulk memory), and erases all its contents, potentially including a host payload previously occupying this memory. This is an extremely invasive operation and, as such, should be used only when strictly necessary and with the customer's full understanding of its implications for the host application.

9.8 Synchronized Module and Host update sequence

When new capabilities or API changes are introduced by a new ExpressLink module firmware version that potentially has backward compatibility issues (side-effects) affecting the host application, the following recommended update sequence should be applied:

1. The manufacturer publishes the new module image and documents the incompatibilities.
2. The customer evaluates the opportunity to apply the update to their fleet and its impact on the host application.
3. The customer develops a new host application with old and *new ExpressLink module* support.

4. A host firmware OTA update is sent to (and accepted by) the host.
5. After rebooting, the host can verify the module current version.
6. An OTA module update must then be offered to the (new) host.
7. The new host can validate the proposed new module version and "allow" the module update.
8. The new host can then switch to the new module API or start using the new feature.

If the host and module fail to stay in step with this sequence, it can be terminated at any point without irreversible consequences and restarted.

9.9 Host OTA updates

Host application updates can be sent to an ExpressLink module using the same OTA mechanisms used for the module's own OTA updates. Thanks to the host OTA feature, ExpressLink modules provide two important services:

- The ability to transport and reconstruct a potentially large payload into the OTA buffer (bulk memory space inside the module) making it available for retrieval by the host in small increments to optimize the host memory resources. The payload can be of any nature (for example, pictures, sounds, and video) and could in fact be a bundle itself, composed of multiple files concatenated together.
- The ability to perform an authenticity check, relieving the host of the heavy cryptographical effort required to hash and verify a cryptographical signature. This second feature is optional in this case, because a host application might perform integrity and authenticity checks on its own, using secrets not accessible to the ExpressLink module or using another custom defined protocol.

9.10 Host OTA Signature Verification

Host firmware updates can also *optionally* have a crypt signature verified by the ExpressLink module after download. Metadata provided during the OTA Job creation (using the AWS IoT Console or the AWS IoT API) informs the module whether the optional signature verification step is required. The developer must then ensure that the host (or other Automated Test Equipment at the end of the production line) sets the HostOTACertificate which provides the required decryption (public) key, otherwise undefined/empty by default.

9.11 Host OTA certificate update

The host OTA certificate can be updated by the customer (OEM) using the AT+CONF command at the end of the product assembly line or later using the OTA mechanism using the code indicated in Table 5 - OTA codes and descriptions. (See the "Host OTA certificate update" entry.)

9.11.1.1 Host OTA certificate updates performed using the OTA mechanism do not require the host to accept the update nor to control when it is run.

9.11.1.2 The host OTA certificate is a configuration parameter initially undefined (empty) and cleared at factory reset.

9.11.1.3 When the host OTA certificate is undefined, the signature verification of an incoming (first) host OTA certificate payload cannot and will NOT be verified.

9.11.2 CONF? *{certificate} pem* »Special certificate output formatting option«

The special qualifier *pem* (case insensitive) can be appended to read a certificate configuration dictionary key (Certificate, HOTAcertificate, RootCA) and produce output in a format that allows the developer to cut and paste the output directly into a standard .pem file for later upload to the AWS IoT dashboard.

> **Note**
>
> The response to this command is an exception to the general format described in 4.6.1 General response formats because it produces more than one output line.

Example:

```
AT+CONF? HOTAcertificate pem{EOL}
```

Returns:

9.11.2.1 `OK# pem{EOL}`

The command returns 'OK' with the number (#) of additional lines, followed by those additional lines composing the certificate, for example:

```
OK9 pem
```

```
-----BEGIN CERTIFICATE-----
MIIDWTCCAkGgAwIBAgIUeKvfYpklvnnattQF09ug9UULjZwwDQYJKoZIhvcNAQEL
BQAwTTFLMEkGA1UECwxCQW1hem9uIFdlYiBTZXJ2aWNlcyBPPUFtYXpvbi5jb20g
...
KHiN1yooauYJKaKr5eJilRAhdYsV2t9X3EFD60/eKmZyD+NE68jAwK/OvokhIGms
cZAj8m0QwqvPkZ0Y2Yc+hPSipQl/hLsg4W/GtbA2MPkTGcvkCBHLYgLBBGpe
-----END CERTIFICATE-----
```

9.11.3 CONF {certificate}=pem »Special certificate input formatting option«

The special value **pem** (case insensitive) can be used to input a certificate (OTAcertificate, HOTAcertificate, RootCA, Certificate) as a multi-line string to allow the developer to directly cut and paste the content of a standard .pem file.

Example:

```
AT+CONF HOTAcertificate=pem
-----BEGIN CERTIFICATE-----
MIIDWTCCAkGgAwIBAgIUeKvfYpklvnnattQF09ug9UULjZwwDQYJKoZIhvcNAQEL
BQAwTTFLMEkGA1UECwxCQW1hem9uIFdlYiBTZXJ2aWNlcyBPPUFtYXpvbi5jb20g
...
KHiN1yooauYJKaKr5eJilRAhdYsV2t9X3EFD60/eKmZyD+NE68jAwK/OvokhIGms
cZAj8m0QwqvPkZ0Y2Yc+hPSipQl/hLsg4W/GtbA2MPkTGcvkCBHLYgLBBGpe
-----END CERTIFICATE-----
```

Returns:

9.11.3.1 OK*{EOL}*

The module returns 'OK' if the new certificate was valid.

9.11.3.2 ERR23 INVALID SIGNATURE

The module returns 'INVALID SIGNATURE' if the new certificate could not be verified.

These command extensions are meant for the developer to use to manually input/output certificates from a terminal application without worrying about escaping the many newline characters contained in a typical .pem file. When a host processor reads or writes to the same certificates, the developer can easily implement the necessary escaping programmatically, resulting in single line (long) strings.

9.12 Server Root Certificate Update

All ExpressLink modules are pre-provisioned with a long-lived AWS server root certificate that is used to validate the endpoint (server) during the TLS connection setup. A new certificate can be provided by means of the AT command interface or the OTA mechanism, using the code indicated in Table 5 - OTA codes and descriptions (Server Root certificate update).

9.12.1.1 Server root certificate updates performed using the OTA mechanism do not require the host to accept the update nor to control its run timing.

9.12.1.2 Server Root certificates are NOT deleted upon a factory reset

9.13 Over the Wire (OTW) module firmware update command

A direct module firmware update mechanism is offered as a convenient alternative for customers that intend to update module firmware during, or immediately after, the assembly/testing line.

The OTW command allows the host to act as the conduit for a new firmware image to the module through the same interface used for the AT commands. Alternatively, a customer's Automated Testing Equipment can seize control of the interface and take over communication with the module (holding the host processor in RESET).

The implementation of this command is optional.

9.13.1 OTW »Enter firmware update mode«

When it receives this command, the module enters a custom bootloader interface that allows you to transfer a complete image to the reserved bulk storage memory.

Returns:

9.13.1.1 `OK`*`{EOL}`*

The module is in OTW mode and ready to receive the new firmware image.

9.13.1.2 The actual protocol used to negotiate the transfer of the file is implementation dependent (XMODEM) and must be documented by each vendor in the module datasheet.

9.13.1.3 The OTW process can be terminated at any point by issuing a hardware reset (pulling the RST pin low).

When the transfer is completed, the same firmware integrity, version compatibility and signature verification process described for the module OTA will be applied. At this point, the module returns one of the values shown here:

Returns:

9.13.1.4 OK*{EOL}*

The image was downloaded successfully. The module will now reboot from the new image in bulk storage.

9.13.1.5

The process will erase all volatile configuration parameters (Topics, PATHs) and re-initialize some of the non-volatile ones in the same way as a Reset command (actual details can be implementation and firmware version dependent).

9.13.1.6

When the boot process completes successfully, the event queue is emptied and a new STARTUP event is generated.

9.13.1.7 ERR21 INVALID OTA UPDATE

If the module is unable to apply the new module images (because of version incompatibility or an integrity check failure), the module returns 'INVALID OTA UPDATE'. The update process is stopped and any OTA memory used is freed.

9.13.1.8 ERR23 INVALID SIGNATURE

If the image signature check fails, the module returns 'INVALID SIGNATURE'. The update process is stopped and any OTA memory used is freed.

10 AWS IoT Services

10.1 AWS IoT Device Defender

(Support for this feature is required and tested for as of v1.1.1.)

ExpressLink devices support the AWS IoT Device Defender service. They can publish a basic set of metrics to AWS IoT Core at a configurable interval, including the table below:

Table 7 - ExpressLink Defender metrics

ExpressLink Custom Metric	Type	Description
Bytes Out	Count	Number of bytes sent since last update.
Messages sent	Count	Number of messages sent since last update.
Messages received	Count	Number of messages received since last update.
Hard Reset Event	Flag	Set to 1 if a hardware reset occurred since last update.
Reconnect Events	Flag	Set to 1 if a reconnect occurred since last update.
Flash Memory Writes	Count	Number of writes to flash memory since last update.
<Module-Name Prefix> Custom Metric(s)		One or more manufactu rer/module specific custom metrics...

All ExpressLink custom metrics are volatile in nature, as their values are reset to 0 after each periodic update (or set to 1 upon a device reset/reboot for the corresponding events).

The device defender feature is activated by setting the *DefenderPeriod* configuration parameter (see Table 2 - Configuration Dictionary Persistent Keys) to a value greater than 0 in the configuration dictionary (using the AT+CONF command).

The *DefenderPeriod* configuration parameter value indicates the *number of seconds* between successive updates of the Device Defender metrics. The maximum period value is an implementation detail that must be documented by the module manufacturer in the device data sheet. Note that the Device Defender service may choose to throttle down (reject) metric updates if they are too frequent.

The latest metrics collected are sent to the Device Defender service as soon as the device connects and at each successive interval. The internal timer continues counting even when the device is disconnected. The internal timer is reset when the Device Defender feature is turned off (when the *DefenderPeriod* configuration parameter is set to 0).

The *DefenderPeriod* parameter is non-volatile, so an ExpressLink device automatically resumes sending Device Defender metrics after a reset (using a RESET command, power cycle or the RST pin).

Module manufacturers can offer additional custom metrics specific to their model. They must prefix the metric with the distinctive **Model** name and document the feature in the device datasheet. (See the "About" Configuration parameter in Table 2 - Configuration Dictionary Persistent Keys.

> **Note**
>
> Access to the AWS IoT Device Defender service is available only when the device is in the *onboarded* state (see 21.3.2 ExpressLink onboarding states and transitions) and the customer/OEM AWS IoT account is properly configured.

Examples:

```
AT+CONF DefenderPeriod=0     # Device Defender metrics are disabled
                             # NOTE: this is the initialized value after a factory
 reset (see Table 2 - Configuration Dictionary Persistent Keys

AT+CONF DefenderPeriod=60    # Device Defender metrics are updated every minute

AT+CONF DefenderPeriod=3600  # Device Defender metrics are updated every hour
```

10.2 AWS IoT Device Shadow

SHADOW commands are provided to facilitate use of the AWS IoT Device Shadow service. Set the **EnableShadow** configuration parameter to 1 to enable support for these commands. (See Table 3 - Configuration dictionary non-persistent keys.) To provide Shadow support, a module automatically handles subscriptions to the various Device Shadow MQTT topics, and parses and reports the responses provided by the service to the device in the form of SHADOW events.

> **Note**
>
> Access to the AWS IoT Device Shadow service is allowed only when the module is in the *onboarded* state (see 21.3.2 ExpressLink onboarding states and transitions), that is, when the module is connected to the customer's AWS account.

The SHADOW INIT# command manages subscriptions to Shadow topics. It must be invoked, and its actions completed, before you invoke any of the other SHADOW commands.

As the interaction with the Device Shadow service often requires a long messaging round trip, the module implements an asynchronous API to avoid blocking the host. SHADOW commands generate requests to the Device Shadow service and return immediately, while SHADOW GET commands can be used later to poll, and eventually retrieve, the responses of the service.

10.2.1.1

Each ExpressLink manufacturer can choose to simultaneously support a maximum number of named shadow documents (**MaxShadow** ≥ 1). The chosen MaxShadow value will be documented in the manufacturer's module datasheet.

The corresponding list of non-persistent parameters, **Shadow1 .. Shadow***{MaxShadow}*, will be pre-populated in the Configuration Dictionary and initialized to empty strings. (See Table 3 - Configuration dictionary non-persistent keys.)

Device Shadow support requires a continuous connection with the AWS IoT Core Service. Such a connection must be established before any SHADOW commands are issued and must be maintained uninterrupted until the completion of any of the requests. If the connection is lost at any point, the host has responsibility to re-initialize the SHADOW interface, re-issue any interrupted commands and, eventually, re-subscribe to shadows for which delta update notifications are expected.

10.2.1.2

All SHADOW commands use a *client-token* defined by the non-volatile configuration parameter *ShadowToken* (factory default: "ExpressLink") to identify and manage requests and responses received on the relevant Device Shadow service topics. Any notifications received that do not match the client-token used in the request, and not generated by the SHADOW commands, are discarded.

10.2.1.3

If the *ShadowToken* configuration parameter is set to an empty string, ANY subsequent notifications received from the Shadow service will NOT be managed by the module but will be added to the messaging queue for the host to handle with the GET0 command (see 5.1.5 GET0 »Request next message pending on an unassigned topic«).

10.2.2 SHADOW*[#]* INIT »Initialize communication with the Device Shadow service«

Initialize the Device Shadow service communication interface for the specified shadow. This subscribes to various topics that are managed by other SHADOW commands such as SHADOW DOC, SHADOW UPDATE and SHADOW DELETE. Note that subscriptions to Shadow Deltas are controlled separately by the SHADOW SUBSCRIBE and SHADOW UNSUBSCRIBE commands.

10.2.2.1 When the INIT process is completed successfully, a SHADOW INIT event is generated (see Table 4 - ExpressLink event codes) and further SHADOW commands can be issued.

10.2.2.2 If the numerical shadow parameter (*[#]*) is not provided, the Unnamed Shadow interface is initialized.

10.2.2.3 Otherwise, the corresponding Shadow# entry in the Configuration Dictionary is used to specify one of the object's Named Shadows.

10.2.2.4 If a Shadow# entry is modified (AT+CONF) before the corresponding SHADOW# INIT process is completed, the initialization is aborted and no Shadow INIT event will be generated.

Returns:

10.2.2.5 `OK`

The Device Shadow service initialization process has started.

10.2.2.6 `ERR7 OUT OF RANGE`

The specified shadow (*[#]*) exceeds the maximum number of shadows supported by this module.

10.2.2.7 `ERR8 PARAMETER UNDEFINED`

The specified shadow (*[#]*) entry in the configuration dictionary is empty.

10.2.2.8 `ERR6 NO CONNECTION`

The device is currently not connected and the request cannot be performed.

10.2.2.9 `ERR24 SHADOW ERROR`

Shadow support is disabled (the *EnableShadow* configuration parameter set to 0 or the device is not in the *onboarded* state, see 21.3.2 ExpressLink onboarding states and transitions).

10.2.4 SHADOW*[#]* DOC »Request a Device Shadow document«

Send a request to the Device Shadow service to retrieve an entire shadow document for the device.

10.2.4.1 A SHADOW DOC event is generated when the request is accepted or rejected.

10.2.4.2 If the numerical shadow parameter (*[#]*) is not provided, the Unnamed Shadow document is requested.

10.2.4.3 Otherwise, the corresponding Shadow# entry in the Configuration Dictionary is used to specify one of the object's Named Shadows.

Returns:

10.2.4.4 `OK`

A shadow document request was sent to the Device Shadow service.

10.2.4.5 `ERR7 OUT OF RANGE`

The specified shadow (*[#]*) exceeds the maximum number of shadows supported by this module.

10.2.4.6 `ERR8 PARAMETER UNDEFINED`

The specified shadow (*[#]*) entry in the configuration dictionary is empty.

10.2.4.7 `ERR6 NO CONNECTION`

The device is currently not connected and the request cannot be performed.

10.2.4.8 `ERR24 SHADOW ERROR`

Shadow support is disabled (the *EnableShadow* configuration parameter set to 0), or the maximum number of simultaneous asynchronous requests was exceeded, or the device is not in the *onboarded* state (see 21.3.2 ExpressLink onboarding states and transitions).

10.2.5 SHADOW*[#]* GET DOC »Retrieve a device shadow document«

Check if a (requested) Device Shadow document has arrived and retrieve its contents.

10.2.5.1 If the numerical shadow parameter (*[#]*) is not provided, the Unnamed Shadow document is requested.

10.2.5.2 Otherwise, the corresponding Shadow# entry in the Configuration Dictionary is used to specify one of the object's Named Shadows.

Returns:

10.2.5.3 `OK`

The requested shadow document has not arrived yet.

10.2.5.4 `OK 1 {document}`

The requested shadow document has arrived.

10.2.5.5 `OK 0 {detail}`

The shadow document request was rejected (0), additional detail is provided.

10.2.5.6 `ERR7 OUT OF RANGE`

The specified shadow (*[#]*) exceeds the maximum number of shadows supported by this module.

10.2.5.7 `ERR8 PARAMETER UNDEFINED`

The specified shadow (*[#]*) entry in the configuration dictionary is empty.

10.2.5.8 `ERR24 SHADOW ERROR`

Shadow support is disabled (the *EnableShadow* configuration parameter set to 0), or the device is not in the *onboarded* state (see 21.3.2 ExpressLink onboarding states and transitions).

Example:

```
AT+SHADOW DOC{EOL}          # Request the entire (unnamed) device shadow document.
OK{EOL}                     # Request submitted.
AT+SHADOW GET DOC{EOL}      # Attempt to retrieve the entire device shadow document.
```

```
OK{EOL}                      # No document has arrived yet.

 ...later...

OK 1 {"state": { "lamp": { "switch": "ON" } }, "version": 11, "timestamp": 1234 }{EOL}
                             # The Device Shadow service response has arrived!
 ...or...
OK 0 {…} {EOL}               # The Device Shadow document request was rejected!
```

10.2.6 SHADOW*[#]* UPDATE *{new state}* »Request a device shadow document update«

Send a request to the Device Shadow service to update a device shadow. The *{new state}* is a JSON document and should NOT contain a "client-token" unless the *ShadowToken* configuration parameter is set to empty (see Table 2 - Configuration Dictionary Persistent Keys), in which case all shadow notifications are left for the host to manage.

10.2.6.1 If the numerical shadow parameter (*[#]*) is not provided, the Unnamed Shadow document is assumed.

10.2.6.2 Otherwise, the corresponding Shadow# entry in the Configuration Dictionary is used to specify one of the object's Named Shadows.

10.2.6.3 A SHADOW UPDATE event is generated when the request is accepted (or rejected).

Returns:

10.2.6.4 `OK`

A shadow document update request was sent to the Device Shadow service.

10.2.6.5 `ERR4 PARAMETER ERROR`

The *{new state}* parameter provided is not a valid JSON document.

10.2.6.6 `ERR7 OUT OF RANGE`

The specified shadow (*[#]*) exceeds the maximum number of shadows supported by this module.

10.2.6.7 `ERR8 PARAMETER UNDEFINED`

The specified shadow (*[#]*) entry in the configuration dictionary is empty.

10.2.6.8 `ERR6 NO CONNECTION`

The device is currently not connected and the request cannot be performed.

10.2.6.9 `ERR24 SHADOW ERROR`

Shadow support is disabled (the *EnableShadow* configuration parameter set to 0).

10.2.6.10 `ERR24 SHADOW ERROR`

If a client-token was present in the update document but *ShadowToken* is NOT empty (SHADOW notifications are managed), or if the device is not in the *onboarded* state (see 21.3.2 ExpressLink onboarding states and transitions), then the module returns 'SHADOW ERROR'.

10.2.7 SHADOW*[#]* GET UPDATE »Retrieve a device shadow update response«

Check if a response to a (requested) Device Shadow update has arrived and retrieve the returned value.

10.2.7.1 If the numerical shadow parameter (*[#]*) is not provided, the Unnamed Shadow document is assumed.

10.2.7.2 Otherwise, the corresponding Shadow# entry in the Configuration Dictionary is used to specify one of the object's Named Shadows.

Returns:

10.2.7.3 `OK`

A shadow document update response has not arrived yet.

10.2.7.4 `OK` *`{0/1} {document}`*

A response to the shadow document update request has arrived. A Boolean value indicates if it was accepted (1) or rejected (0). An additional document containing the update details is appended.

10.2.7.5 `ERR7 OUT OF RANGE`

The shadow specified (*[#]*) exceeds the maximum number of shadows supported by this module.

10.2.7.6 `ERR8 PARAMETER UNDEFINED`

The specified shadow (*[#]*) entry in the configuration dictionary is empty.

10.2.7.7 ERR24 SHADOW ERROR

Shadow support is disabled (the *EnableShadow* configuration parameter set to 0), or the device is not in the *onboarded* state (see 21.3.2 ExpressLink onboarding states and transitions).

Example:

```
AT+SHADOW1 UPDATE {"state":{"desired":{"switch": "off" } } }{EOL}
OK{EOL}                     # The request was sent.
AT+SHADOW1 GET UPDATE{EOL}  # Check if the update was accepted/rejected.
OK{EOL}                     # No response received yet.

 ...later...

AT+SHADOW1 GET UPDATE(EOL}  # Check if the update was accepted/rejected.
OK 1 {"switch": "off"){EOL} # The update was accepted.
 ...or...
OK 0 {..}{EOL}              # The update was rejected.
```

10.2.8 SHADOW*[#]* SUBSCRIBE »Subscribe to a device shadow document«

Send a request to the Device Shadow service to receive Delta updates for a shadow document.

10.2.8.1 If the numerical shadow parameter (*[#]*) is not provided, the Unnamed Shadow document is requested.

10.2.8.2 Otherwise, the corresponding Shadow# entry in the Configuration Dictionary is used to specify one of the object's Named Shadows.

10.2.8.3 A SHADOW SUBACK or SHADOW SUBNACK event are generated when the subscription is accepted or rejected. Note that if a Shadow# (configuration string) is modified before the subscription confirmation (or rejection) is received, the corresponding event will not be generated.

Returns:

10.2.8.4 OK

A shadow subscribe request was sent to the Device Shadow service.

10.2.8.5 ERR7 OUT OF RANGE

The specified shadow parameter (*[#]*) exceeds the maximum number of shadows supported by this module.

10.2.8.6 ERR8 PARAMETER UNDEFINED

The specified shadow (*[#]*) entry in the configuration dictionary is empty.

10.2.8.7 ERR6 NO CONNECTION

The device is not currently connected and the request cannot be performed.

10.2.8.8 ERR24 SHADOW ERROR

Shadow support is disabled (the *EnableShadow* configuration parameter set to 0), or the device is not in the *onboarded* state (see 21.3.2 ExpressLink onboarding states and transitions).

Example:

```
AT+SHADOW2 SUBSCRIBE{EOL}          # Request subscription to the device Shadow2.
OK{EOL}                            # Request submitted.

 ...later...

AT+EVENT?{EOL}                     # Check if the subscription was accepted.
OK{EOL}                            # No response has arrived yet.
 ...or...
OK 26 2{EOL}                       # SHADOW SUBACK The subscription to Shadow2 was
 accepted.
 ...or...
OK 27 2{EOL}                       # SHADOW SUBNACK The subscription to Shadow2 was
 rejected.
```

10.2.9 SHADOW*[#]* UNSUBSCRIBE »Unsubscribe from a device shadow document«

Send a request to the Device Shadow service to stop receiving Delta updates for a shadow document. Note that no SHADOW event is generated following this request.

10.2.9.1 If the numerical shadow parameter (*[#]*) is not provided, the Unnamed Shadow document is requested.

10.2.9.2 Otherwise, the corresponding Shadow# entry in the Configuration Dictionary is used to specify one of the object's Named Shadows.

Returns:

10.2.9.3 `OK`

A shadow document request was sent to the Device Shadow service.

10.2.9.4 `ERR7 OUT OF RANGE`

The shadow specified (*[#]*) exceeds the maximum number of shadows supported by this module.

10.2.9.5 `ERR8 PARAMETER UNDEFINED`

The specified shadow (*[#]*) entry in the configuration dictionary is empty.

10.2.9.6 `ERR6 NO CONNECTION`

The device is currently not connected and the request cannot be performed.

10.2.9.7 `ERR24 SHADOW ERROR`

Shadow support is disabled (the *EnableShadow* configuration parameter set to 0), or the device is not in the *onboarded* state (see 21.3.2 ExpressLink onboarding states and transitions).

Example:

```
AT+SHADOW3 UNSUBSCRIBE{EOL}    # Request subscription to the device Shadow2.
OK{EOL}                        # Request submitted.
```

10.2.10 SHADOW*[#]* GET DELTA »Retrieve a Shadow Delta message«

Query for the next pending shadow update message. This command can be used after receiving a DELTA event, or to poll the delta queue (the queue depth is implementation specific).

10.2.10.1 If the numerical shadow parameter (*[#]*) is not provided, the Unnamed Shadow document is requested.

10.2.10.2 Otherwise, the corresponding Shadow# entry in the Configuration Dictionary is used to specify one of the object's Named Shadows.

Returns:

10.2.10.3 `OK`

An empty string indicates no delta pending.

10.2.10.4 `OK {delta document}`

A shadow delta update has arrived. A document containing the details is appended.

10.2.10.5 `ERR7 OUT OF RANGE`

The shadow specified (*[#]*) exceeds the maximum number of shadows supported by this module.

10.2.10.6 `ERR8 PARAMETER UNDEFINED`

The specified shadow (*[#]*) entry in the configuration dictionary is empty.

10.2.10.7 `ERR24 SHADOW ERROR`

Shadow support is disabled (the *EnableShadow* configuration parameter set to 0), or the device is not in the *onboarded* state (see 21.3.2 ExpressLink onboarding states and transitions).

10.2.11 SHADOW*[#]* DELETE »Request the deletion of a Shadow document«

Requests the Device Shadow Service to delete a device shadow document.

10.2.11.1 If the numerical shadow parameter (*[#]*) is not provided, the Unnamed Shadow document is requested.

10.2.11.2 Otherwise, the corresponding Shadow# entry in the Configuration Dictionary is used to specify one of the object's Named Shadows.

10.2.11.3 A SHADOW DELETE event is generated when the request is accepted or rejected.

Returns:

10.2.11.4 `OK`

The request was sent to the Device Shadow service.

10.2.11.5 `ERR7 OUT OF RANGE`

The shadow specified (*[#]*) exceeds the maximum number of shadows supported by this module.

10.2.11.6 `ERR8 PARAMETER UNDEFINED`

The specified shadow (*[#]*) entry in the configuration dictionary is empty.

10.2.11.7 `ERR6 NO CONNECTION`

The device is currently not connected and the request cannot be performed.

10.2.11.8 `ERR24 SHADOW ERROR`

Shadow support is disabled (the *EnableShadow* configuration parameter set to 0), or the maximum number of simultaneous asynchronous requests was exceeded, or the device is not in the *onboarded* state (see 21.3.2 ExpressLink onboarding states and transitions).

10.2.12 SHADOW*[#]* GET DELETE »Request a Shadow delete response«

Check if a Device Shadow Delete request was accepted.

10.2.12.1 If the numerical shadow parameter (*[#]*) is not provided, the Unnamed Shadow document is requested.

10.2.12.2 Otherwise, the corresponding Shadow# entry in the Configuration Dictionary is used to specify one of the object's Named Shadows.

Returns:

10.2.12.3 `OK`

The request was sent to the Device Shadow service.

10.2.12.4 `OK` *`{0/1} {payload}`*

The shadow delete request was accepted (1) or rejected (0). Additional detail may be present in the *{payload}*.

10.2.12.5 `ERR7 OUT OF RANGE`

The shadow specified (*[#]*) exceeds the maximum number of shadows supported by this module.

10.2.12.6 `ERR8 PARAMETER UNDEFINED`

The specified shadow (*[#]*) entry in the configuration dictionary is empty.

10.2.12.7 `ERR24 SHADOW ERROR`

Shadow support is disabled (the *EnableShadow* configuration parameter set to 0), or the device is not in the *onboarded* state (see 21.3.2 ExpressLink onboarding states and transitions).

10.2.13 SHADOW EVENTS

When subscribing to shadow document messages, retrieving a shadow document, or requesting updates, the module communicates with the Device Shadow service using various MQTT topics. When it receives a response to a request or a shadow Delta update to which it has subscribed, the module reports this to the host *asynchronously* by generating a Shadow event (see Table 4 - ExpressLink event codes). Each Shadow event carries the *Shadow-Index* parameter:

- **0** indicates the unnamed Shadow
- **1..MaxShadow** indicates the corresponding Shadow# entry in the configuration table

Example 1:

```
AT+EVENT?{EOL}                # Query events pending.
OK 24 1{EOL}                  # A SHADOW DELTA event was received for the Shadow1.

AT+SHADOW1 GET DELTA{EOL}     # Fetch the delta message.
OK {delta document}{EOL}      # Returns the delta document received.
```

Example 2:

```
AT+SHADOW SUBSCRIBE{EOL}      # Request a subscription to DELTA updates for the unnamed
 Shadow
OK{EOL}                       # Request sent successfully.

 ...later...

AT+EVENT?{EOL}
OK  26 0{EOL}                 # SHADOW SUBACK The subscription request was accepted.
 ...or...
OK 27 0{EOL}                  # SHADOW SUBNACK The subscription request was rejected.
```

10.3 AWS IoT Jobs

The JOB command is not implemented. It is reserved for the support of the AWS IoT Jobs service.

11 Additional services

11.1.1 TIME? »Request current time information«

ExpressLink modules *must* provide time information as available from SNTP, GPS or cellular network sources. Devices can choose to maintain a time reference internally even when disconnected or in sleep mode, depending on implementation specific software or hardware capabilities.

Returns:

11.1.1.1 `OK` *`{date YYYY/MM/DD} {time hh:mm:ss.xx} {source}`*

If time information is available and recently obtained, the module returns 'OK' followed by that information.

11.1.1.2 `ERR15 TIME NOT AVAILABLE`

A recent time fix could not be obtained.

11.1.2 WHERE? »Request location information«

ExpressLink modules can optionally provide last location information as available from GPS, GNSS, cellular network or other triangulation method. A time stamp is provided to allow the host determine whether the information is current. The implementation of this command is optional.

Returns:

11.1.2.1 `OK` *`{date} {time} {lat} {long} {elev} {accuracy} {source}`*

If location coordinates could be obtained at date/time, the module returns 'OK' followed by the information.

11.1.2.2 `ERR16 LOCATION NOT AVAILABLE`

A location fix could not be obtained.

12 Provisioning and Onboarding

All ExpressLink modules will be equipped with a pre-provisioned hardware root of trust (on chip or external secure element, secure enclave, TPM, iSIM). This will provide the necessary unique identifier (UID) of the module, a key pair (public, private) and will hold a certificate that is signed by

a CA shared with AWS as part of ExpressLink program. This certificate will be used to transfer the module public key to the AWS endpoint upon activation.

12.1 ExpressLink Modules Activation

Upon first use, or following a complete factory reset, each ExpressLink module is ready to establish a connection according to the model's specific connectivity capabilities (Wi-Fi, Cellular, ...). In case of Wi-Fi modules, this is possible only after the end-user has provided the module with the proper Wi-Fi credentials for a local, compatible Wi-Fi Access Point (router).

12.1.1 ExpressLink Staging Account Authentication

Each ExpressLink module is ready to establish a connection with a default AWS IoT ExpressLink staging account. The connection is mutually authenticated using the ExpressLink birth certificate (and an AWS server certificate) and upgraded to a secure socket connection (Mutual TLS).

12.1.2 ExpressLink Staging Account Endpoint

During the qualification process, AWS assigns each ExpressLink manufacturing partner a dedicated staging account and the associated, unique AWS endpoint (URL).

12.1.2.1 The assigned staging account endpoint is set as the "factory default" for the Endpoint configuration parameter (see Table 2 - Configuration Dictionary Persistent Keys).

12.1.3 ExpressLink Birth Certificate

Each ExpressLink device must be provided with an X.509 certificate that conforms to the following specification:

12.1.3.1

The Serial Number must contain the device Unique ID (a unique, nonsequential 128-bit or larger number) also assigned as the ExpressLink module ThingName configuration.

12.1.3.2

The certificate signature is provided by a Certificate Authority that has been registered by the vendor with AWS IoT Core for the exclusive use of the vendor ExpressLink modules.

12.1.3.3

The expiration date is set to no less than 10 years from the device certificate issue.

12.1.4 ExpressLink staging account device registration

Using the staging account endpoint, the ExpressLink module proceeds to login to the AWS IoT Core MQTT broker. If successful, an automated process (JITP or similar) creates a thing and associated policies using an ExpressLink template and appends it to the staging account registry.

12.2 ExpressLink Evaluation Kits Quick Connect Flow

ExpressLink Evaluation Kits are able to use the ExpressLink staging account to deliver a fast, out-of-box experience. As soon as connected they are able to publish data to an ExpressLink MQTT topic ("data") and subscribe to any ExpressLink MQTT topic ("state"). AWS provides a simple web application (Quick Connect) to all ExpressLink users to visualize the data published by the Host processor (using animated graphs) and to send customizable commands back to their Host processors.

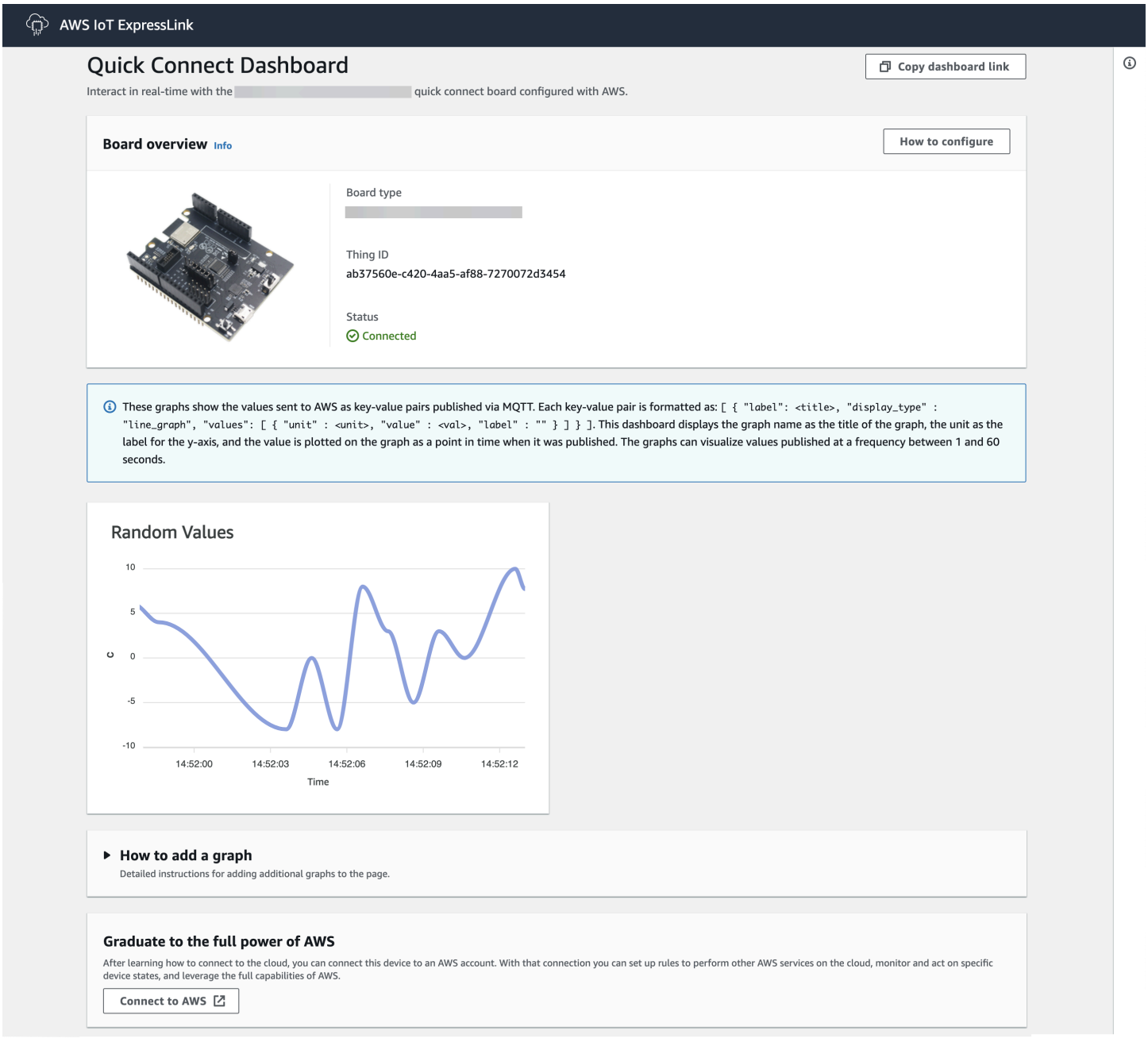


Figure 5 - ExpressLink evaluation kit Quick Connect flow

Developers are also able to register their ExpressLink modules to their private developer's accounts and proceed to application development with a few simple, manual steps, including:

- extracting the device certificate
- uploading it to their private accounts
- updating the ExpressLink endpoint

12.2.1 Workshop Default Wi-Fi Credentials (Optional)

To reduce the number of configuration steps (and time) required to establish a Wi-Fi connection, a default set of Wi-Fi credentials can be provided in the configuration dictionary at Factory Reset.

Using default Wi-Fi credentials can be convenient in workshop, classroom or seminar environments avoiding a number (10+) of user to attempt simultaneously to use a CONFMODE (Bluetooth) connection and greatly simplifying the room set up.

If implemented, the manufacturer will document such credentials in the module datasheet.

12.3 ExpressLink Production Onboarding Flow

Onboarding is the process of creating a "thing" corresponding to each physical device in the customer account registry in order to provide access to the account's IoT core services. Each thing must be associated with a valid certificate and access policy document.

In a production flow, ExpressLink customers can use any of a number of automated onboarding techniques as required by their application, including:

- Pre-registration, where the modules' certificates are obtained before assembly and uploaded to the customer account in advance.
- End of (assembly) Line registration, where module certificates are collected after product assembly and individually uploaded to the customer's AWS account.
- End of Line batch registration, where module certificates are collected after product assembly and shipped in batches to the customer for upload into the AWS account.
- Just in Time Registration, where the device onboards to the customer account at first connection. (This requires pre-registration of the module manufacturer's CA to the customer account.)
- Late-binding, where the end product user performs the product onboarding (often simultaneously with the user's own registration, although the two steps should not be confused).

12.3.1 ExpressLink Late Binding (Onboarding by Claim) flow example

A late binding onboarding flow can be initiated by the end-user after purchasing the finished product when they connect it for the first time and register the product. The end-user can be directed to a web application devised by the OEM/customer (for example, a toaster manufacturer) that will guide the user through the following steps:

1. Enter Wi-Fi credentials (only for Wi-Fi modules)– this is required to access the AWS cloud. To accomplish this, the host can activate a CONFMODE for credential entry or the host can directly manipulate the configuration dictionary (SSID, Passphrase).
2. Access the ExpressLink staging account for the first time.
3. Claim the ExpressLink module (identified by ThingName) from the staging account.
4. Transfer the certificate to the OEM account registry (thing creation).
5. Update the ExpressLink module Endpoint (to point to the OEM account).
6. Disconnect and reconnect the ExpressLink Device to the OEM account.

Steps 1 and 2 are facilitated by the staging account assigned to each manufacturer and managed by AWS. Steps 3 and 4 require the customer to implement a claim mechanism that interacts with the AWS managed staging account. Step 5 is facilitated by a specific device feature as described in 21.3.2 ExpressLink onboarding states and transitions.

Additional steps to register the user, create an end-user (application) account, collect user identifiable information (user name and password) and bind it to the ExpressLink thing are left to the OEM application.

21.3.2 ExpressLink onboarding states and transitions

The configuration parameter Endpoint (see Table 2 - Configuration Dictionary Persistent Keys) controls the onboarding state of the device. The device is in the *staging* state when the Endpoint parameter (string) matches the factory default value that corresponds to an AWS-managed staging account assigned to each manufacturer. The device is in the *onboarded* state when the Endpoint parameter has been modified to point to a customer account (endpoint) by a host that directly updated the configuration dictionary using a CONF command (see 6.2.1 CONF KEY=*{value}* »Assignment«) or by means of the following remote update process:

12.3.2.1 When (and only when) in the *staging* state, a connected ExpressLink module automatically subscribes ONLY to the endpoint-update topic: ***ThingName*/expresslink_config**. Then, when it receives a message on the update topic with the following format: **{"Endpoint" : "value"}**, the module updates the Endpoint configuration parameter with the requested new value.

12.3.2.2 The host can retrieve the MSG event produced (GET0) and use it to implement additional optional features, such as to alert the user of the device of a successful onboarding (registration).

12.3.2.3 The module will also automatically disconnect. The related CONLOST event will inform the host that it must re-establish a new connection, this time to the newly assigned endpoint.

12.3.2.4 The host can query the state of the module using the CONNECT? command and inspecting the second numerical parameter provided in the response (see 4.7.1 CONNECT? »Request the connection status«) without having to inspect the contents of the Endpoint configuration parameter (or knowing or assuming the default Endpoint value to compare against).

12.3.2.5 When (and only when) in the *onboarded* state, a connected ExpressLink module subscribes automatically to several AWS-reserved topics as required to support OTA and other core ExpressLink functionality. In the same way, features dependent on the AWS IoT Device Defender and AWS IoT Device Shadow services are supported only when a module is in the *onboarded* state.

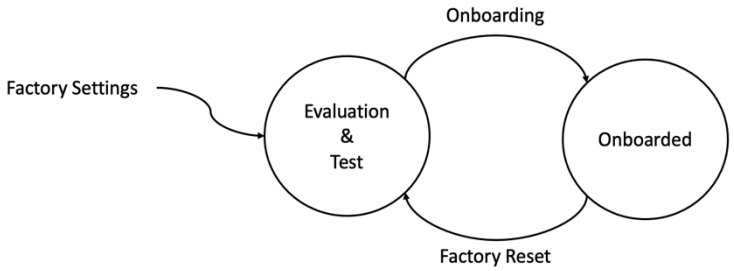


Figure 6 - ExpressLink onboarding states diagram

Once onboarded, all ExpressLink modules behave as fully owned devices and connect to the customer/OEM account as the ExpressLink things are transferred to the chosen OEM registry. It is the responsibility of the OEM to manage the product life cycle, use the OTA services to apply module updates (with images provided by the ExpressLink module vendor) and apply host processor application updates as needed.

12.4 End-User change, product re-registration

This is a normal occurrence in the life of many products when they are resold or disposed of. If required, ExpressLink modules can be reset to factory settings to reactivate the onboarding process from the beginning and eliminate any previous owner association.

12.5 Handling onboarding failures

The onboarding process can fail at various points due to end-user, host application, or network errors. We envision the following scenarios:

- Onboarding process failure: if the OEM misconfigures the account policies this would prevent the device certificate from being moved into the target account. The AWS IoT API will report this type of error to OEM developers during testing.
- Onboarding process failure: if the ExpressLink claim and removal from the staging account fails this would leave it in the staging account while a new thing is created in the OEM account and the ExpressLink module is redirected to the new endpoint. Staging account periodic cleaning and fraud detection sweeps will clear the anomaly in a short time.
- Endpoint Update failure: if the device does not receive the ExpressLink endpoint update message it remains in the staging account and fails to connect to the target OEM account within a given amount of time. The binding process (web application) can be designed to timeout and guide the user to repeat the procedure until successful.
- Accidental product factory reset: in this case, the ExpressLink device will rejoin the staging account as soon as connectivity is regained, and the onboarding process can be restarted at any time. The OEM application will be able to detect that an already registered device is re-applying to onboarding and could possibly help to restore the product status and/or report the error to developers.

13 Bluetooth Low Energy (BLE)

ExpressLink modules can take advantage of additional (local) connectivity capabilities, optionally available on selected SoCs. Bluetooth Low Energy (BLE) is one prominent example of such capabilities used to communicate with accessories and other modules or gateways over a local (personal) area network.

BLE interfaces can be configured by means of *profiles* which describe a collection of *attributes* used to transfer individual pieces of information between two devices. Devices can assume one of two roles:

1. Central – a BLE device which initiates an outgoing connection request to an advertising peripheral device.
2. Peripheral - the BLE device which accepts an incoming connection request after advertising its presence and capabilities.

A host processor can access BLE features of a capable ExpressLink module through the set of commands described in this chapter. Additionally, a set of events is available to support asynchronous communication with the host (see the BLE specific events in Table 4 - ExpressLink event codes) and the configuration parameters in the ExpressLink configuration dictionary (see Table 3 - Configuration dictionary non-persistent keys).

BLE Central Device Configuration

In particular, BLE devices that adopt a central role require the host to initialize common configuration using the **BLECentral** parameter, and one or more **BLECentral#** parameters in order to describe the (peripheral) devices they wish to connect to. The numerical suffix # is an integer between 1 and MaxBLECentral - a value dependent on the module capabilities. (Manufacturers of ExpressLink modules are required to publish the MaxBLECentral value in the module datasheet).

BLECentral configuration parameters use JSON notation and expect the following keys:

- **filterDups** (1/0) – filters duplicate broadcasts by the same device.
- **address** is specified as:

  "address": {"**type**": *type*, "**BDAddress**":"AA:BB:CC:DD:EE:FF"}

  Where *type* is:
  - **"Public"**: Public Address (Fixed IEEE MAC address assigned by vendor - default).
  - **"Random"**: Static Random Address (Manually configured static/random address).
  - **"RPAPublic"**: Public Identity Address (Public address but used with identity resolution).
  - **"RPARandom”**: Random Identity Address (Automatically generated random static address with identity resolution).

  BDAddress is specified only when type is Random or RPARandom.
- **IOCapabilities**: Specifies I/O Capability of the host, used during pairing (see 13.4 Pairing, Bonding and Filtering).
  - Setting "IOCapability":{"display":0,"keyboard": 0,"YesNo": 0} implies Just Works method.
- **customFilters** (JSON object) – additional vendor-defined filtering options (must be documented by the vendor on the module's datasheet).

Individual BLECentral# configuration parameters use JSON notation and expect the following keys:

- **peer** mac address of a target peripheral device is specified as:

"**peer**": "AA:BB:CC:DD:EE:FF"

Where the address type is “Public” by default, or:

"**peer**": *type*, "**BDAddress**":"AA:BB:CC:DD:EE:FF"}

Where *type* is:

- **"Public"**: Public Address (Fixed IEEE MAC address assigned by vendor - default).
- **"Random"**: Static Random Address (Manually configured static/random address).
- **"RPAPublic"**: Public Identity Address (Public address but used with identity resolution).
- **"RPARandom”**: Random Identity Address (Automatically generated random static address with identity resolution).

> **Note**
>
> If the peer address was found using AT+BLE GET DISCOVER (see 13.2.2 BLE GET DISCOVER »Retrieve the collected advertising information«), the address type should match the type reported as the last response item..

- **customFilters** (JSON object) – additional vendor-defined filtering options (must be documented by the vendor on the module's datasheet).

BLE Peripheral Device Configuration

BLE devices adopting a peripheral role require the host to initialize the BLEPeripheral (single) configuration parameter in order to describe the device's connection requirements. BLEPeripheral configuration parameters use JSON notation and expect the following keys:

- **mode**: Indicates advertising mode. Possible values are:
  - "**LEGACY**": refers to BLE specifications before 5.0. (default).
  - "**EXTENDED**": refers to the extended advertising as define in BLE specification 5.0 and later.
- **IOCapability**: Specifies I/O Capability of the peripheral, used during pairing (see 13.4 Pairing, Bonding and Filtering), possible values are:
  - "**display**": 0/1 the device can interact with a user via a display (default 0)
  - "**keyboard**": 0/1 the device can interact with the user via a keyboard to enter numerical values (default 0)

- "**YesNo**": 0/1 the device has buttons for the user to accept or reject a connection (default 0)

> **Note**
>
> "IOCapability": {"display":0,"keyboard": 0,"YesNo": 0} implies “Just Works” pairing method.

- **filterPolicy**: Used to apply a filter policy using the following keys:
  - "**ALLOW_LIST_SCAN**": to grant scan requests to devices in the AllowList.
  - "**ALLOW_LIST_CONNECT**": to grant connection requests to devices in the AllowList.
- **address**: is specified as:

  "address": {"**type**": *type*, "**BDAddress**":"AA:BB:CC:DD:EE:FF"}

  Where *type* is:
  - **"Public"**: Public Address (Fixed IEEE MAC address assigned by vendor - default).
  - **"Random"**: Static Random Address (Manually configured static/random address).
  - **"RPAPublic"**: Public Identity Address (Public address but used with identity resolution).
  - **"RPARandom”**: Random Identity Address (Automatically generated random static address with identity resolution).

  BDAddress is specified only when type is Random or RPARandom.

> **Note**
>
> While it is possible to cancel an ongoing advertisement, by editing the advertisement configuration filterPolicy and re-advertising, it is not possible to do the same with the address type. For address type changes to take effect, you need to AT+RESET the device.

Table 8 - BLEPeripheral filterPolicy configuration options

ALLOW_LIST_CONNECT	ALLOW_LIST_SCAN	Description
0	0	Allow both scan and connection request from anyone

ALLOW_LIST_CONNECT	ALLOW_LIST_SCAN	Description
0	1	Allow scan requests from AllowList devices only and connection requests from anyone
1	0	Allow scan requests from anyone and connection requests from AllowList devices only
1	1	Allow scan and connectio n requests from AllowList devices only

- **raw** is used to assign pre-encoded complete (hex) strings for the advertisement and response. (The raw data is expected as per the BLE link layer specification. It is directly sent in the same form over the air without any further processing. Thus, certain multi-octet fields need to be reversed. Please refer to the Bluetooth Core Specification: Vol 6, Part B, § 1.2 “Bit ordering” and Vol 1, Part E, § 2.9 “Type Names”).
- **appearance** (string) – a 4 digit hex number that represents the type of device. (See page 28 of the Bluetooth Specification).

Example 1

```
AT+CONF BLEPeripheral={"mode": "LEGACY",
    "IOCapability":{"display":0,"keyboard":0,"YesNo": 0},
    "filterPolicy": {"ALLOW_LIST_SCAN":0,"ALLOW_LIST_CONNECT": 0},
    "raw":{"advertisement":"02010603030F18",
        "response":"0C09457870726573734C696E6B"},
     "address":{"type":"RPARandom"}
}
```

> **Note**
>
> Basic parsing of the JSON string happens when the CONF command is executed, hence formatting errors (ERR 4 – PARAMETER ERROR) can be immediately reported. Configuration keys/values are validated only at connection time, when a BLE peripheral initialization is attempted, and may cause it to fail (ERR27 – BLE ERROR or ERR28 – CONFIGURATION ERROR).

BLE Characteristics Configuration

Both Central and Peripheral devices require the host to initialize one or more BLEGATT# parameters in order to describe individual characteristics they wish to publish or access. The numerical suffix # is an integer between 1 and MaxBLEGATT - a value dependent on the module's capabilities. (Manufacturers of ExpressLink modules are required to publish the MaxBLEGATT value in the module datasheet).

BLEGATT# configuration parameters use JSON notation and expect the following keys:

- **service** (UUID string) – the UUID of a BLE service.
- **chr** (UUID string) – the UUID of a BLE characteristic.
- **write** (permission bitmask in decimal) – optional
- **read** (permission bitmask in decimal) – optional
- **indicate** (permission bitmask in decimal) – optional
- **notify** (permission bitmask in decimal) – optional

UUID strings can be in short form 4 hex-digits (for example, "20AD") for short 16-bit services and characteristics, or long form 128-bit for custom service and characteristic identifiers (for example, "00000000-0000-1000-8000-00805F9B34FB"). Long form UUID separators ("-") can be omitted. The default value for optional keys is 0. The maximum size of a characteristic value is MaxBLECharLen bytes, where *MaxBLECharLen* is a value (>=31) defined by the vendor and documented in the module datasheet.

Example 2: Two services composed of one characteristic each

```
AT+CONF BLEGATT1={"service": "1809", "chr": "2A1C" }
```

```
    # Thermometer service, Temperature characteristic.

AT+CONF BLEGATT2={"service": "180F", "chr": "2A19" }
    # Battery Level service and level(%) characteristic.
```

Since BLEGATT# configuration parameters describe only one characteristic each, multiple parameters are required to describe a complex service composed of a number of characteristics.

Example 3: A service composed of two characteristics.

```
AT+CONF BLEGATT1={"service": "180D", "chr": "2A37" }
    # Heart rate service, heart rate measurement characteristic.

AT+CONF BLEGATT2={"service": "180D", "chr": "2A38" }
    # Heart rate service, body sensor location characteristic
```

Permission bitmasks are defined as decimal values computed as follows:

Table 9 - Characteristics Permissions Bitmask

Bit Position (lsb to msb)	Permission
0	On / Enabled
1	Encrypt
2	Authenticate
3	Authorize
4	Write-Without-Response
5 and higher	Reserved (must be 0)

Details:

- **Encrypted**: If set, the link will be required to be encrypted.
- **Athenticate**: If set, implies that Man-In-the-Middle (MITM) protection must be enabled. If the *Just Works* pairing method is used, any attempt to use a characteristic that sets this flag will be rejected since this pairing method does not provide MITM protection.

- **Authorize**: If set, will generate an Authorize event (see Table 4 - ExpressLink event codes - ExpressLink event codes). The host is required to approve or deny access to the characteristic (see 13.3.6 BLE AUTH*[#] [0/1]* »Authorize access to a characteristic«)
- **Write-Without-Response**: (this applies only to the write property) If set, a write command will not generate a response (shortening the transaction).

When a module is acting as a Central, if the Peripheral on the other side has both Write and Write-Without- Response flag set on a characteristic, the module acting as BLE Central will carry out the full 'write' operation, overriding the 'write-without-response' option. Only if the peripheral has exclusively set the Write-Without-Response flag, a 'write-without-response' operation is carried out.

Table 10 - Example Values for Write property

Value (Decimal)	Binary Representation	Decoding	Notes
0	00000		Write Disable
1	00001	On	Write Enabled
3	00011	On + Encrypt	Encrypted link required for write operation
7	00111	On + Encrypt + Authenticate	Encrypted and Authenticated (MITM protected) link required for write operation
9	01001	On + Authorize	Trigger authorization event before allowing write operation for this attribute
18	10010	Encrypt + Write-Without-Resp	Encrypted but only write without response operation

Value (Decimal)	Binary Representation	Decoding	Notes
			are allowed for this attribute

BLE Descriptors Configuration

BLE descriptors are attributes that provide additional information about a BLE characteristic, helping to describe its value and how it should be used. They essentially act as metadata for characteristics, offering details about their format, meaning, and how they can be interacted with. BLE descriptors can be configured using the same BLEGATT# configuration dictionary entries used for normal characteristics but with the following distinctions:

- A **desc** key is added to provide the descriptor's uuid-string
- No **notify, indicate** or **write-without-response** (permission bitmask) are allowed

Example: Define the Environmental Sensing Service (0x181A) with Temperature characteristic (0x2F6A), and Presentation Format Descriptor (0x2904). Then assign a value like below to the descriptor:

- Flags: sint16 (0x00 0x00)
- Sampling Function: uint8 (0x02) Arithmetic Mean
- Measurement Period: uint24 (0x000096) -> Little endian (0x96 0x00 0x00) = 150s
- Update Interval: uint24(0x000384) -> (0x84 0x03, 0x00) = 900s
- Application: uint8(0x04) = soil
- Measurement Uncertainty: uint8(0xFF) = Unknown uncertainty

```
AT+CONF BLEGATT1={"service":"181A", "chr":"2A6E", "read":"1"}
AT+CONF BLEGATT2={"service":"181A", "chr":"2A6E", "desc":"290C", "read":"1"}
AT+BLE SET1 6009Big endian: 0x0960, Actual: 2400 in C/100 = 24 C)
AT+BLE SET2 0000029600008403000 4FF
```

Complete Peripheral Configuration Examples

Example

Example 1:

1. Configure BLE in Peripheral mode:

```
AT+CONF BLEPeripheral={
 "mode": "LEGACY",
 "IOCapability":{"display":0,"keyboard":0,"YesNo": 0},
 "filterPolicy":{"ALLOW_LIST_SCAN":0,"ALLOW_LIST_CONNECT":0},
 "raw":{"advertisement":"02010603030F18","response":"0C09457870726573734C696E6B"},
 "address":{"type":"RPARandom"}
}
```

Where:

- **"mode": "LEGACY"** implies legacy advertisement mode (before BLE spec 5.0).
- **"IOCapability":{"display":0,"keyboard": 0,"YesNo": 0}** the host has no special I/O capability but can perform "Just Works" pairing.
- **"filterPolicy": {"ALLOW_LIST_SCAN":0,"ALLOW_LIST_CONNECT": 0}** allow both scan and connection requests from anyone
- **"raw"** allows full configurability of the advertisement and response data.
- **"address":{"type":"RPARandom"}** Random Identity Address (automatically generated random static address) with identity resolution.

2. Define service 180D, characteristic 2A37:

```
AT+CONF BLEGATT1={"service":"180D", "chr":"2A37", "write":18, "read":3}
```

The write bitmask (18) enables write operations but allows only write-without-response operations. The read bitmask (3) enables read operations but requires an encrypted link.

3. Define service 180D, characteristic 2A38:

```
AT+CONF BLEGATT2={"service":"180D", "chr":"2A38", "write":3, "read":1,"indicate":1,
 "notify":1}
```

The write bitmask (3) enables write (with response) operations but requires an encrypted link. The read bitmask (1) enables read operations and does not require an encrypted link. The indicate bitmask (1) enables indicate operations and does not require an encrypted link. The notify bitmask (1) enables notify operations and does not require an encrypted link.

4. Initialize the peripheral:

```
AT+BLE INIT PERIPHERAL
```

5. Advertise the peripheral:

```
AT+BLE ADVERTISE
```

After successful connection, you receive the event: **40 0 BLE CONNECTED**. Any Central device (for example, smartphone) can now see the device and connect to it.

Example

Example 2: Configuring a device as peripheral and retrieve its BDAddress.

1. Configure the device as a peripheral with automatically assigned address and identity resolution:

```
AT+CONF BLEPeripheral = {"address": {"type":"RPARandom"}}
OK
```

2. Configure characteristics and descriptors:

```
AT+CONF BLEGATT1={"service":"180D", "chr":"2A37", "write":1, "read":1, "indicate":1}
OK
AT+CONF BLEGATT2={"service":"180D", "chr":"FFF1", "write":1, "read":1, "notify":1}
OK
AT+CONF BLEGATT3={"service":"180D", "desc":"FFF2", "write":1, "read":1}
OK
```

3. Initialize the peripheral:

```
AT+BLE INIT PERIPHERAL
OK
```

4. Retrieve the automatically assigned address:

```
AT+CONF? BLEPeripheral
OK
{"mode":"LEGACY", "address": {"type":"RPARandom","BDAddress":"FD:15:18:24:FC:6D"}}
```

13.1 BLE initialization

13.1.1 BLE INIT *[*CENTRAL|PERIPHERAL*]* »Initializing the device role«

Initialize the BLE interface to operate in the selected (GAP) role. Note how this version of the ExpressLink specification allows a device to be configured as Central or Peripheral but not both. Once a BLE interface is initialized, the only way to terminate it or change its mode of operation is to use the RESET command. However, doing so will disconnect the device (if it is connected) and will reset all internal state. Non-persistent configuration parameters (see Table 3 - Configuration dictionary non-persistent keys) will be reinitialized, all subscriptions will be terminated, and the message queues will also be emptied.

For Central Mode:

Issuing the INIT CENTRAL command does not require (yet) any of the BLE configuration parameters to be set. One or more BLECentral# and BLEGATT# configuration parameters will be required later, before issuing an actual connection request (see the BLE CONNECT command). BLECentral# and BLEGATT# parameters are used in central mode to act as filters to identify suitable connection target devices.

For Peripheral Mode:

Issuing the INIT PERIPHERAL command requires the BLEPeripheral and one (or more) BLEGATT# configuration parameters to be set. BLEGATT keys do not need to be set in numerical order. All BLE parameters found (initialized) in the configuration dictionary will be used to define the peripheral's GATT service. After initialization, the host may not change the BLEGATT# service composition (characteristics) without first resetting the device. The host can instead update or retrieve the latest characteristic values using the appropriate SET/GET commands (see BLE SET and BLE GET commands).

Returns:

13.1.1.1 `OK`*{EOL}*

If the command was accepted and the requested central or peripheral role is set.

13.1.1.2 ERR4 PARAMETER ERROR*{EOL}*

If the role parameter was not CENTRAL or PERIPHERAL.

13.1.1.3 ERR28 CONFIGURATION ERROR*{EOL}*

If the peripheral role was selected and the configuration dictionary does not contain the BLEPeripheral and at least one BLEGATT# parameter.

13.1.1.4 ERR25 NOT ALLOWED*{EOL}*

If the BLE role was already initialized.

13.1.1.5 ERR27 BLE ERROR*{EOL}*

Failed to initialize the BLE Stack.

13.1.1.6

filterDups is 0 by default.

Example 1:

```
AT+BLE INIT CENTRAL{EOL}
OK
AT+CONF BLECENTRAL1={"peer": "a4:c1:38:12:56:5d"}{EOL}
OK
```

Example 2:

```
AT+CONF BLEPeripheral={"appearance": "4142"}{EOL}
OK
AT+CONF BLEGATT2={"service": "1809" ,"chr": "2A1C" }{EOL}
OK
AT+BLE INIT PERIPHERAL{EOL}
OK
```

13.2 BLE CENTRAL role commands

13.2.1 BLE*[#]* DISCOVER *[duration|*CANCEL*]* »Scanning and Advertisement«

BLE capable devices can communicate with accessories without establishing permanent connections by means of a scanning and advertising protocol. This protocol is commonly used for

characteristics whose value is required infrequently (device battery level, or ambient temperature sensors).

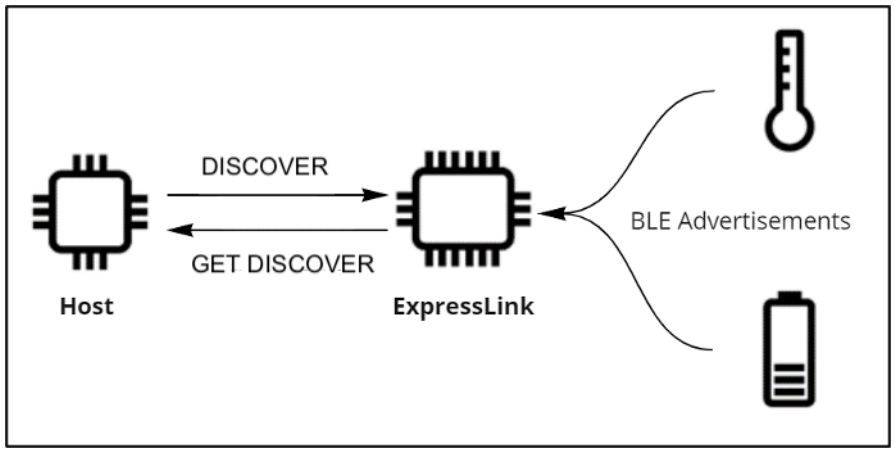


Figure 7 - BLE scanning for devices

The ExpressLink module scans for advertisements from peripherals that match criteria defined in the corresponding configuration string BLECentral#. This is an asynchronous command. It returns an immediate response to confirm the process has started (or an error prevented it). During the scanning time, any advertised data received will be queued. The scanning process will stop after a set amount of time optionally configured by the duration parameter (30 seconds by default), and a BLE DISCOVER COMPLETE event (see Table 4 - ExpressLink event codes) will be produced. Queued data can then be retrieved using the BLE GET DISCOVER command.

The CANCEL parameter is used to terminate an ongoing scanning process.

13.2.1.1 Scans for all the nearby devices if BLECentral# is set to {} (an empty JSON object).

13.2.1.2 If the command is issued while a scanning process is in progress, any queued data is discarded and a new scanning process is started.

13.2.1.3 If *{duration}* is provided, only devices found within *{duration}* seconds will be captured.

13.2.1.4 Enqueues a BLE DISCOVER COMPLETE event on successful Discover complete or cancellation (see Table 4 - ExpressLink event codes).

13.2.1.5 Enqueues a BLE DISCOVER ERROR event when the discovery fails for any reason. Hint codes can be defined by the vendor to provide additional insight on the reason for the failure. (Hint codes must be documented in the module datasheet).

Returns:

13.2.1.6 OK*{EOL}*

If the command was accepted and the scanning sequence started.

13.2.1.7 OK*{EOL}*

If the CANCEL parameter is given and the cancellation request was successfully submitted. CANCEL will terminate any scanning activity in progress regardless of the # index given. However, a suffix index is required for the command to execute.

13.2.1.8 ERR4 PARAMETER ERROR*{EOL}*

If the parameter is 0 seconds.

13.2.1.9 ERR4 PARAMETER ERROR*{EOL}*

If the numerical suffix # is omitted.

13.2.1.10 ERR7 OUT OF RANGE*{EOL}*

If the numerical suffix # is out of bounds (0 or greater than MaxBLECentral).

13.2.1.11 ERR8 PARAMETER UNDEFINED*{EOL}*

If the numerical suffix # points to an empty BLECentral# string.

13.2.1.12 ERR25 NOT ALLOWED*{EOL}*

If Central role is not initialized or if the message queue is full.

Example 1 - Scan for any advertising device in range for a default timeout of 30 seconds:

```
AT+BLE DISCOVER{EOL}

   ...a BLE DISCOVER COMPLETE event occurs (see Table 4 - ExpressLink event codes)...

AT+EVENT?{EOL}    # check the event queue
OK 31 0 DISCOVER COMPLETE{EOL}  # a BLE DISCOVER event was received
```

Example 2 - Scan for a specific peripheral UUID and timeout after 20 seconds:

```
AT+CONF BLECentral1={ "peer": "a4:c1:38:12:56:5d", "filterDups": 1}{EOL}
```

```
AT+BLE1 DISCOVER 20{EOL}

   ...20 seconds later a BLE DISCOVER COMPLETE event occurs (see Table 4 - ExpressLink
 event codes...
```

Example 3 - Discover until cancelled:

```
AT+BLE DISCOVER{EOL}
OK

   ...a few seconds later...

AT+BLE DISCOVER CANCEL{EOL}
OK

   ...a BLE DISCOVER COMPLETE event occurs (see Table 4 - ExpressLink event codes...
```

13.2.2 BLE GET DISCOVER »Retrieve the collected advertising information«

Retrieve the advertising information collected during the last discovery process.

Advertising information is stored in memory shared with MQTT messages. Collected information is also cleared when a new discovery is started. Hence, GET DISCOVER only fetches information for the devices that match the filter defined when the last DISCOVER command was issued.

Returns:

13.2.2.1 OK*{EOL}*

No new advertising information was collected.

13.2.2.2 OK *{collected information}{EOL}*

Discovered information was collected and is presented following the 'OK'.

13.2.2.3 ERR25 NOT ALLOWED*{EOL}*

If the BLE Central role is not initialized.

Collected information is fetched from the queue and returned as a record containing the following space-separated fields:

```
<peer> <connectable> <Scannable> <rssi> <advertisedData> <type>{EOL}
```

```
mac address    0/1          0/1       int     hex string   string
```

Example 1 - Check the BLE receive queue for any advertising data received, returning two records:

```
AT+BLE GET DISCOVER{EOL}
OK 3a:1a:f7:e4:11:38 0 1 -30 02011A0BFF4C00090603420A3F588B Public{EOL}
```

Example 2 - Repeat the GET DISCOVER request until the message queue is empty:

```
AT+BLE GET DISCOVER{EOL}
OK{EOL}
```

13.2.3 BLE*[#]* CONNECT »Connect to a peripheral«

In some use cases, instead of just reading sensor data provided in the advertisement message, the host may want to inspect what type of services and characteristics a peripheral exposes. To do this, the ExpressLink module must first establish a direct connection to the BLE peripheral device (using the BLE GAP protocol).

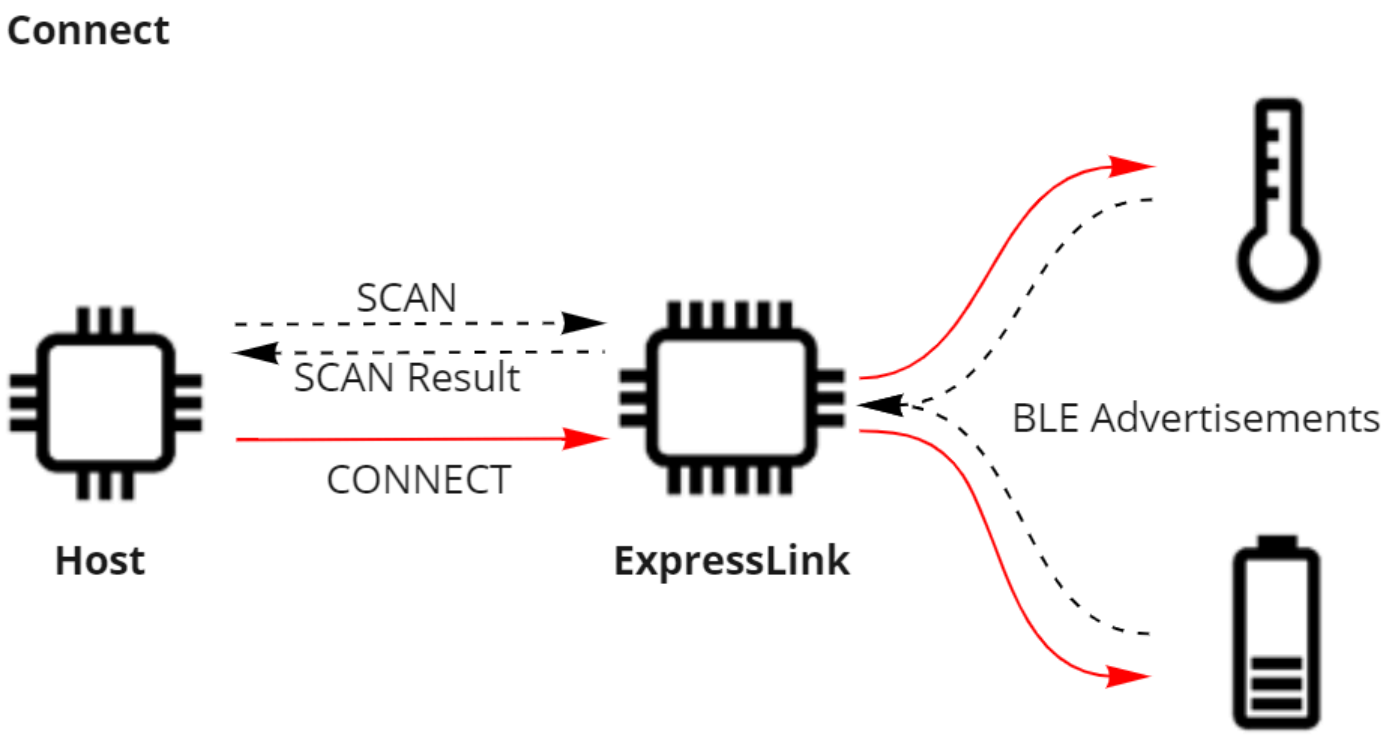


Figure 8 - Connecting to a BLE device

When you use the BLE{#} CONNECT command, the ExpressLink module attempts to connect to a peripheral based on a specific configuration defined int the BLECentral# parameter.

Returns:

13.2.3.1 OK*{EOL}*

If the command was accepted and the connection request was successful.

13.2.3.2 ERR7 OUT OF RANGE*{EOL}*

The numerical suffix is out of bounds (0 or greater than MaxBLECentral index).

13.2.3.3 ERR28 CONFIGURATION ERROR*{EOL}*

The numerical suffix points to a {}, or is missing the "peer" key.

13.2.3.4 ERR8 PARAMETER UNDEFINED*{EOL}*

The numerical suffix points to an empty BLECentral# string or {}.

13.2.3.5 ERR25 NOT ALLOWED*{EOL}*

If already connected to a device.

13.2.3.6 ERR25 NOT ALLOWED*{EOL}*

If BLE INIT is not set to CENTRAL mode.

13.2.3.7 ERR14 UNABLE TO CONNECT*{EOL}*

Discovery attempted for 30 seconds before timing out and returning error.

Example 1 - Connect to a specific peripheral according to the configuration. The attempt will stop if the timeout of 30 seconds is reached. The configuration information included in this example is for demonstration purposes only:

```
AT+CONF BLECENTRAL1={"peer": "a4:c1:38:12:56:5d", "filterDups": 1 }{EOL}
OK{EOL}
AT+BLE INIT CENTRAL{EOL}
OK{EOL}
AT+BLE1 CONNECT{EOL}
OK{EOL}
```

```
   ...a BLE CONNECTED event occurs...
```

Example 2 - Connect to a second peripheral according to the configuration. The attempt will stop if the timeout of 30 seconds is reached. The configuration information included in this example is for demonstration purposes only:

```
AT+CONF BLECENTRAL2={"peer": "a4:c1:38:12:56:5d", "filterDups":1}{EOL}
OK{EOL}
AT+BLE2 CONNECT{EOL}
OK{EOL}

   ...a BLE CONNECTED event occurs...
```

13.2.4 BLE*[#]*CONNECT? »Connection status query«

This query command allows the host device to check if the status of the specified connection is still active. This command can also be used to confirm a successful connection after the AT+CONNECT command. Note that if the numerical suffix is not specified then the Peripheral command 13.3.1 BLE CONNECT? »Connection status query« is invoked.

Returns:

13.2.4.1 `OK 1 CONNECTED`*`{EOL}`*

When connected to a Peripheral.

13.2.4.2 `OK 0 DISCONNECTED`*`{EOL}`*

When disconnected from a Peripheral.

13.2.4.3 `ERR7 OUT OF RANGE`*`{EOL}`*

The numerical suffix is out of bounds (0 or greater than MaxBLECentral index).

13.2.4.4 `ERR25 NOT ALLOWED`*`{EOL}`*

When the module role was not initialized as Central.

Example 1 - When connected over central config 2:

```
AT+BLE INIT CENTRAL{EOL}
```

```
OK{EOL}
AT+BLE2 CONNECT{EOL}
OK{EOL}

   ...a BLE CONNECTED event occurs...

AT+BLE2 CONNECT?{EOL}
OK 1 CONNECTED{EOL}
```

13.2.5 BLE*[#]*DISCONNECT »Connection termination request«

Terminate a BLE device connection.

13.2.5.1 Disconnects from the given index's connection.

Returns:

13.2.5.2 OK*{EOL}*

On successful termination of connection or if there is no connection to terminate.

13.2.5.3 ERR7 OUT OF RANGE*{EOL}*

The numerical suffix is out of bounds (0 or greater than MaxBLECentral index).

13.2.5.4 ERR27 BLE ERROR*{EOL}*

If the command fails to terminate the connection.

13.2.5.5 ERR25 NOT ALLOWED*{EOL}*

If BLE INIT is not set to CENTRAL mode.

Example 1 - Connect to a specific peripheral according to the configuration and then disconnect. The configuration information included in this example is for demonstration purposes only:

```
AT+CONF BLECentral1={"peer": "a4:c1:38:12:56:5d", "filterDups": 1}{EOL}
OK{EOL}
AT+BLE INIT CENTRAL{EOL}
OK{EOL}
AT+BLE1 CONNECT{EOL}
OK{EOL}
```

```
   ...a BLE CONNECTED EVENT occurs...

AT+BLE1 DISCONNECT{EOL}
OK{EOL}
```

13.2.6 BLE*[#]* READ*[#]* »Synchronous Read of a Characteristic«

The READ command allows the host to request the value of a characteristic when connected to a peripheral. The BLE (first) numerical suffix # identifies the connected device by the corresponding BLECentral# parameter. The READ (second) numerical suffix # identifies the characteristics by the corresponding BLEGATT# parameter. The maximum value that can be read from a characteristic is 31 bytes.

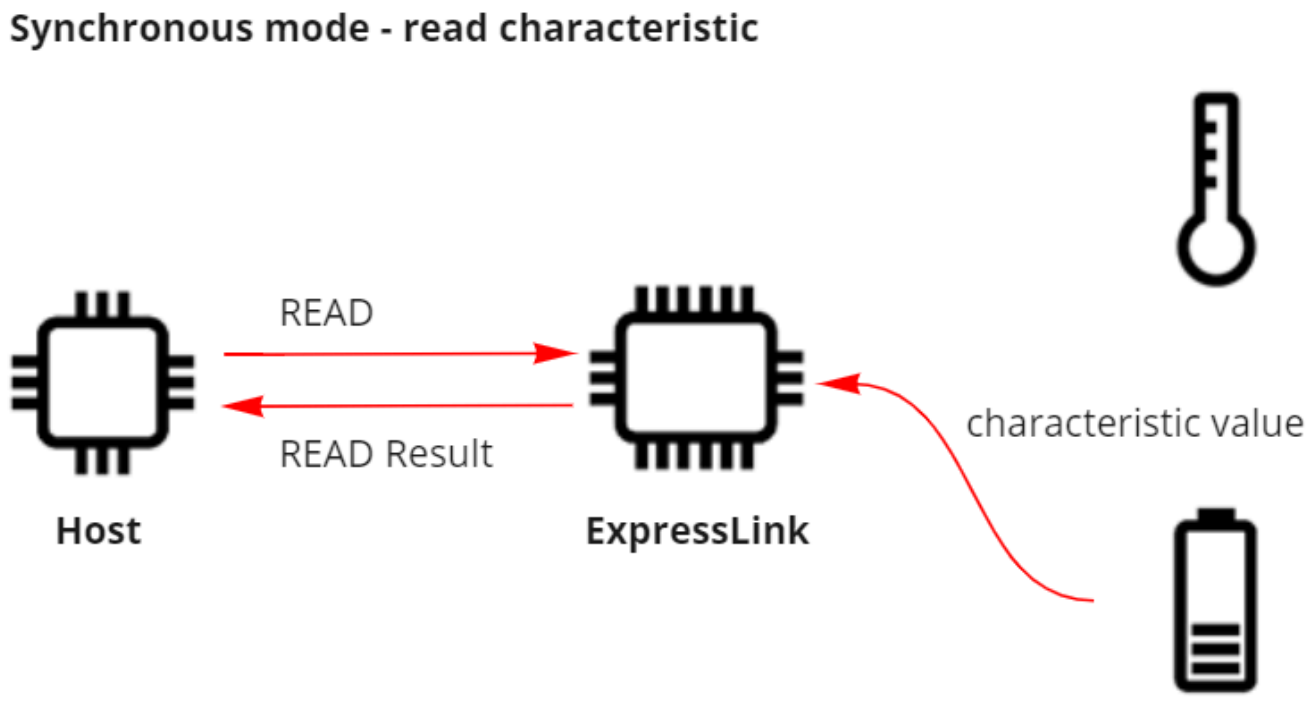


Figure 9 - Reading a connected BLE peripheral

Returns:

13.2.6.1 OK *{value}{EOL}*

On a successful read, returns the characteristic value as a hex string.

13.2.6.2 OK *{EOL}*

On a successful read, when the characteristic value is an empty string.

13.2.6.3 ERR8 PARAMETER UNDEFINED*{EOL}*

When a BLECentral# configuration is not set.

13.2.6.4 ERR8 PARAMETER UNDEFINED*{EOL}*

When a BLEGATT# configuration is not set.

13.2.6.5 ERR7 OUT OF RANGE*{EOL}*

The first numerical suffix is out of bounds (0 or greater than MaxBLECentral).

13.2.6.6 ERR7 OUT OF RANGE*{EOL}*

The second numerical suffix is out of bounds (0 or greater than MaxBLEGATT).

13.2.6.7 ERR6 NO CONNECTION*{EOL}*

When not connected to a **peripheral** device.

13.2.6.8 ERR27 BLE ERROR*{EOL}*

When the read request fails.

13.2.6.9 ERR25 NOT ALLOWED*{EOL}*

If BLE INIT is not set to CENTRAL mode.

Example:

```
    # Assuming the configuration:
AT+CONF BLECentral1={"peer": "a4:c1:38:12:56:5d", "filterDups": 1}{EOL}
OK{EOL}

AT+CONF BLEGATT5={"service": "1f10", "chr": "1f1f"}{EOL}
OK{EOL}

    # BLE is initialized and a connection to a peripheral matching peer
 a4:c1:38:12:56:5d is established:
AT+BLE INIT CENTRAL{EOL}
OK{EOL}
```

```
AT+BLE1 CONNECT{EOL}
OK{EOL}

    # Requst the value of characteristic 0x1F1F (part of the service 0x1F10):
AT+BLE1 READ5{EOL}
OK 48656C6C6F20576F726C64{EOL}

    # Successfully retrieved the value "Hello World"!
```

13.2.7 BLE*[#]* WRITE*[#]* *{value}* »Write to a characteristic«

The WRITE command allows the host to update the value of (writable) characteristics of a connected peripheral device. The BLE (first) numerical suffix # identifies the connected device by the corresponding BLECentral# parameter. The WRITE (second) numerical suffix #, identifies a characteristic by its corresponding BLEGATT# parameter.

The maximum value that can be written to a characteristic is *MaxBLECharLen* bytes, where *MaxBLECharLen* is a value (>=31) defined by the vendor and documented in the module datasheet.

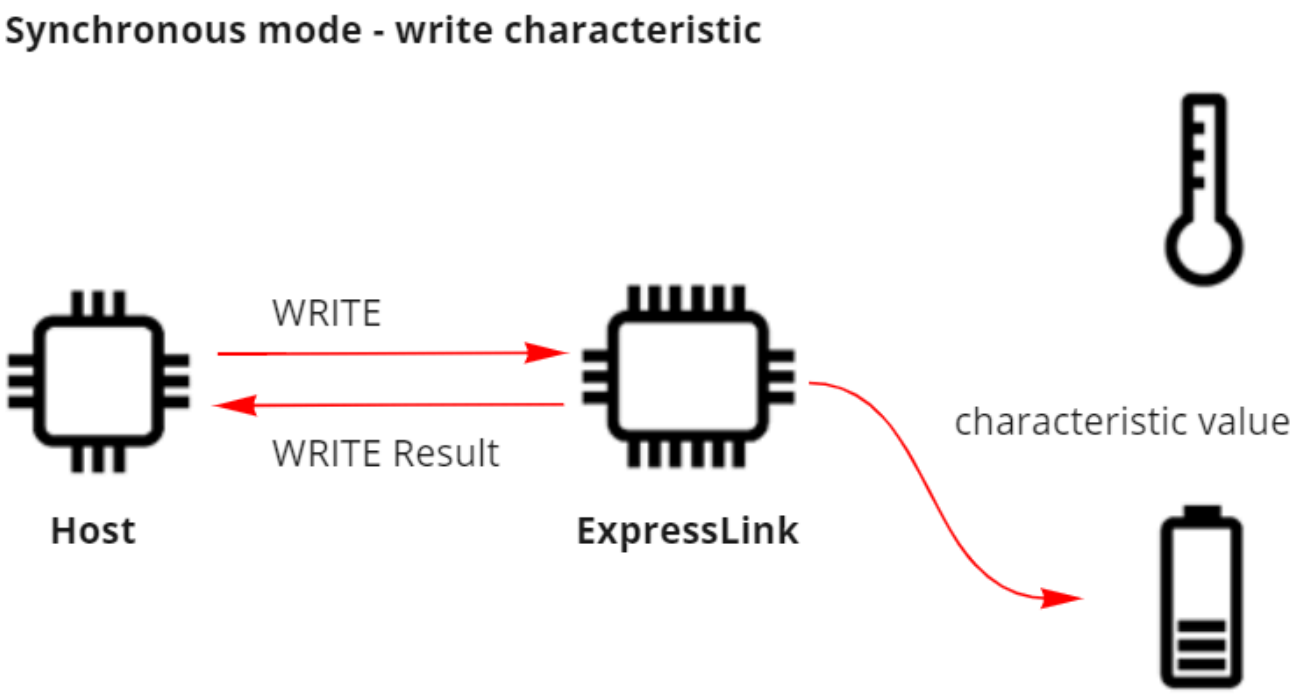


Figure 10 - Writing to a connected BLE device

Returns:

13.2.7.1 OK*{EOL}*

On a successful update of the peripheral characteristic.

13.2.7.2 ERR4 PARAMETER ERROR*{EOL}*

Must always take valid byte array encoded in hex as a valid parameter.

13.2.7.3 ERR8 PARAMETER UNDEFINED*{EOL}*

A BLEGATT# configuration was not set.

13.2.7.4 ERR7 OUT OF RANGE*{EOL}*

The BLE Central numerical suffix is out of bounds (0 or greater than MaxBLECentral).

13.2.7.5 ERR7 OUT OF RANGE*{EOL}*

The BLEGATT numerical suffix is out of bounds (0 or greater than MaxBLEGATT).

13.2.7.6 ERR6 NO CONNECTION*{EOL}*

When not connected to a peripheral device.

13.2.7.7 ERR25 NOT ALLOWED*{EOL}*

The device was not initialized as a Central.

13.2.7.8 ERR27 BLE ERROR*{EOL}*

When the command fails to update the characteristic.

Example:

```
    # Assuming the configuration:
AT+CONF BLECentral2={"peer": "a4:c1:38:12:56:5d", "filterDups": 1}{EOL}
OK{EOL}

AT+CONF BLEGATT6={"service": "1f10", "chr": "1f1f"}{EOL}
OK{EOL}
```

```
    # After initializing and connecting to a peripheral device:
AT+BLE INIT CENTRAL{EOL}
OK{EOL}

AT+BLE1 CONNECT{EOL}
OK{EOL}

    # Request to update the characteristic 0x1F1F with the new value "01A3":
AT+BLE2 WRITE6 01A3{EOL}
OK{EOL}
```

13.2.8 BLE*[#]* SUBSCRIBE*[#]* »Subscribe to a connected peripheral«

The host can subscribe to receive notifications (see Table 4 - ExpressLink event codes) when connected to a peripheral and the selected characteristic is updated (it must be configured as notify or indicate). The BLE (first) numerical suffix # identifies the connected device by the corresponding BLECentral# parameter. The SUBSCRIBE (second) numerical suffix #, identifies the characteristic by its corresponding BLEGATT# parameter.

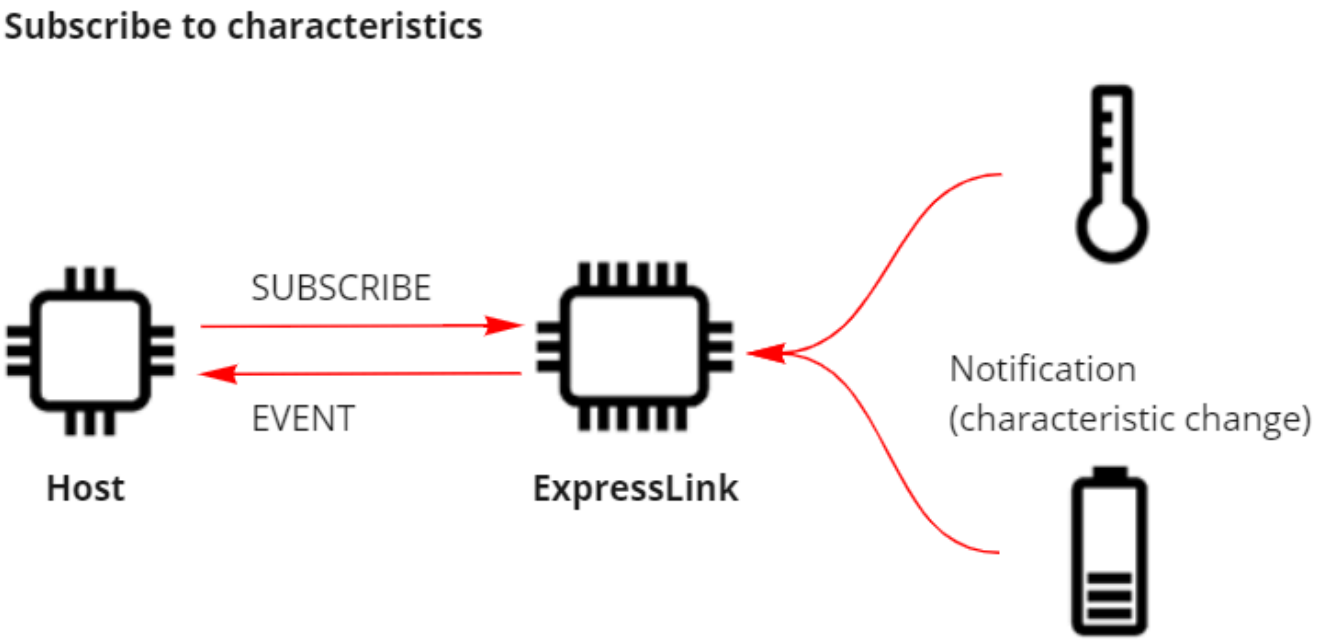


Figure 11 - Subscribing to receive peripheral notifications

Returns:

13.2.8.1 OK*{EOL}*

On a successful subscription.

13.2.8.2 ERR8 PARAMETER UNDEFINED*{EOL}*

A BLEGATT# configuration was not set.

13.2.8.3 ERR7 OUT OF RANGE*{EOL}*

The first numerical suffix is out of bounds (0 or greater than MaxBLECentral).

13.2.8.4 ERR7 OUT OF RANGE*{EOL}*

The second numerical suffix is out of bounds (0 or greater than MaxBLEGatt).

13.2.8.5 ERR6 NO CONNECTION*{EOL}*

When not connected to a peripheral device.

13.2.8.6 ERR25 NOT ALLOWED*{EOL}*

The command can only be executed when initialized as a Central.

13.2.8.7 ERR27 BLE ERROR*{EOL}*

When the command fails to successfully subscribe to the indexed characteristics. ExpressLink will support a limited number of subscriptions (minimum 2). The max number of supported subscriptions (*MaxBLECharLen*) must be documented by the vendor in the module datasheet.

Example:

```
    # Assuming the configuration:
AT+CONF BLECentral1={"peer": "a4:c1:38:12:56:5d", "filterDups": 1}{EOL}
OK{EOL}
AT+CONF BLEGATT7={"service":"1809", "chr":"1F1F"}{EOL}
OK{EOL}

    # After initializing and connecting to a peripheral device:
AT+BLE INIT CENTRAL{EOL}
OK{EOL}
AT+BLE1 CONNECT{EOL}
OK{EOL}
```

```
    # Request to be notified to
AT+BLE1 SUBSCRIBE7{EOL} OK

   ...EVENT 50 101 SUBSCRIPTION RECEIVED...
```

13.2.9 BLE*[#]* GET SUBSCRIBE*[#]* »Get information on Subscriptions«

After receiving the event subscribed to, the host can retrieve additional detail about the notification. The first BLE numerical suffix # identifies the connected device by the corresponding BLECentral# parameter. The SUBSCRIBE (second) numerical suffix # identifies the characteristic by its corresponding BLEGATT# parameter. The second numerical index is optional- if not specified, the most recent subscription detail will be retrieved.

Returns:

13.2.9.1 OK*{EOL}*

No notification detail record was found.

13.2.9.2 OK *{BLEGATT#}* [N|I] *{detail}{EOL}*

A notification detail record was found. Where “N” stands for Notification and “I” for Indication. When requesting a subscription to a characteristic that has both Notification and Indication, the module will subscribe only to Notification.

13.2.9.3 ERR7 OUT OF RANGE*{EOL}*

The first numerical suffix is out of bounds (0 or greater than MaxBLECentral).

13.2.9.4 ERR7 OUT OF RANGE*{EOL}*

The second numerical suffix is out of bounds (0 or greater than MaxBLEGatt).

13.2.9.5 ERR8 PARAMETER UNDEFINED*{EOL}*

A BLECentral# configuration was not set.

13.2.9.6 ERR25 NOT ALLOWED*{EOL}*

If the device is not initialized in the Central role.

Example 1:

```
    # Assuming the configuration:
```

```
AT+CONF BLECentral1={"peer": "a4:c1:38:12:56:5d", "filterDups": 1}{EOL}
OK{EOL}
AT+CONF BLEGATT2={"service":"1809", "chr":"1F1F"}{EOL}
OK{EOL}

    # After initializing and connecting to a peripheral device:
AT+BLE INIT CENTRAL{EOL}
OK{EOL}
AT+BLE1 CONNECT{EOL}
OK{EOL}

   # Subscribe to notifications/indication provided by characteristic 0x1F1F
AT+BLE1 SUBSCRIBE2{EOL}
OK{EOL}

   ...EVENT 48 102 SUBSCRIPTION received...

    # Request additional information:
AT+BLE1 GET SUBSCRIBE2{EOL}
OK 2 N 021A45{EOL}
```

13.2.10 BLE*[#]* UNSUBSCRIBE*[#]* »Unsubscribe to characteristics«

Terminate a notify or indicate subscription to a peripheral device characteristic.

Returns:

13.2.10.1 OK*{EOL}*

On successfully terminating the subscription to the peripheral characteristic.

13.2.10.2 ERR8 PARAMETER UNDEFINED*{EOL}*

A BLECentral# configuration was not set.

13.2.10.3 ERR8 PARAMETER UNDEFINED*{EOL}*

A BLEGATT# configuration was not set.

13.2.10.4 ERR7 OUT OF RANGE*{EOL}*

The BLECentral numerical suffix is out of bounds (0 or greater than MaxBLECentral).

13.2.10.5 ERR7 OUT OF RANGE*{EOL}*

The BLEGATT numerical suffix is out of bounds (0 or greater than MaxBLEGATT).

13.2.10.6 ERR6 NO CONNECTION*{EOL}*

When not connected to a peripheral device.

13.2.10.7 ERR25 NOT ALLOWED*{EOL}*

The command can only be executed when initialized as a Central.

13.2.10.8 ERR27 BLE ERROR*{EOL}*

When the command fails to successfully unsubscribe from the selected characteristic.

Example:

```
    # Assuming the configuration:
AT+CONF BLECentral1={"peer": "a4:c1:38:12:56:5d", "filterDups": 1}{EOL}
OK{EOL}
AT+CONF BLEGATT8={"service":"1809", "chr":"1F1F"}{EOL}
OK{EOL}

    # After initializing and connecting to a peripheral device and subscribing to
 notifications:
AT+BLE INIT CENTRAL{EOL}
OK{EOL}
AT+BLE1 CONNECT{EOL}
OK{EOL}
AT+BLE1 SUBSCRIBE8{EOL}
OK{EOL}

 ...EVENT 50 108 SUBSCRIPTION received...

AT+BLE1 UNSUBSCRIBE8{EOL}
OK{EOL}

    # No more subscription events will be generated for the selected characteristic.
```

13.3 BLE PERIPHERAL role commands

When initialized as a Peripheral device, an ExpressLink module waits for connections initiated by central devices. Only one Central device at a time can connect to a module initialized as Peripheral.

13.3.1 BLE CONNECT? »Connection status query«

Request the current Peripheral device connection status.

Returns:

13.3.1.1 OK 0 NOT CONNECTED*{EOL}*

When not connected to a Central device.

13.3.1.2 OK 1 CONNECTED*{EOL}*

When connected to a Central device.

13.3.1.3 OK 2 ADVERTISING*{EOL}*

When advertising is in progress.

13.3.1.4 ERR25 NOT ALLOWED*{EOL}*

When the module role was not initialized as Peripheral.

Example:

```
AT+CONF BLEPeripheral={"appearance": "4142"}{EOL}
OK{EOL}
AT+CONF BLEGATT1=={"service":"1809", "chr":"1F1F"}{EOL}
OK{EOL}
AT+BLE INIT PERIPHERAL{EOL}
OK{EOL}
AT+BLE CONNECT?{EOL}
OK 0 NOT CONNECTED{EOL}

    # Currently initialized in peripheral role but not connected to a central device.
```

13.3.2 BLE DISCONNECT »Connection termination request«

Terminate the current connection, if one was established by a central device.

Returns:

13.3.2.1 OK*{EOL}*

On successfully terminating the connection of if there was no connection to terminate.

13.3.2.2 ERR25 NOT ALLOWED*{EOL}*

When the module role was not initialized as Peripheral.

13.3.2.3 ERR27 BLE ERROR*{EOL}*

If the command fails to terminate the connection.

Example:

```
    # When initialized as a peripheral and wishes to disconnect from central device.
AT+CONF BLEPeripheral={“appearance”: “4142”}{EOL}
OK{EOL}
AT+CONF BLEGATT1={“service”:”1809”, ”chr”:”1F1F”}{EOL}
OK{EOL}
AT+BLE INIT PERIPHERAL{EOL}
OK{EOL}

   ...A central device connects...
   ...a BLE CONNECTED EVENT occurs...

    # The host wishes to terminate the connection with the central device:
AT+BLE DISCONNECT{EOL}
OK{EOL}
```

13.3.3 BLE ADVERTISE *{CANCEL}* »Advertise to nearby devices«

Start the advertising process, making the module a connectable, scannable device. The process will continue until connected or cancelled by AT+BLE ADVERTISE CANCEL.

Returns:

13.3.3.1 OK*{EOL}*

When the ExpressLink module successfully starts the Advertising process.

13.3.3.2 OK*{EOL}*

On successfully canceling advertisements, or if not currently advertising.

13.3.3.3 ERR4 PARAMETER ERROR*{EOL}*

When initialized as a peripheral and the command is provided with an index #.

13.3.3.4 ERR27 BLE ERROR*{EOL}*

When the Advertising process fails to start.

13.3.3.5 ERR27 BLE ERROR*{EOL}*

When the Advertising process fails to terminate (with the CANCEL parameter).

13.3.3.6 ERR25 NOT ALLOWED*{EOL}*

If BLE INIT is not set to PERIPHERAL mode.

Example 1 - A successful startup of the advertising process:

```
AT+CONF BLEPeripheral={“appearance”: “4142”}{EOL}
OK{EOL}
AT+CONF BLEGATT1={ "service" : "72bdd8d118874a3fbedaf6c22d45cfa0", "chr":
 "72bdd8d218874a3fbedaf6c22d45cfa0"}{EOL}
OK{EOL}
AT+BLE INIT PERIPHERAL{EOL}
OK{EOL}
AT+BLE ADVERTISE{EOL}
OK{EOL}

    # Now other devices can discover the ExpressLink device
```

Example 2 - When the necessary configuration CONF BLEPeripheral and/or BLEGATT# are missing:

```
AT+CONF BLEGATT1={ "service" : "CFAO", "chr": "72BD"}{EOL}
OK{EOL}
AT+BLE INIT PERIPHERAL{EOL}
ERR28 CONFIGURATION ERROR{EOL}
```

Example 3 - Cancelling an ongoing Advertisement:

```
AT+CONF BLEPeripheral={“appearance”: “4142”}{EOL}
OK{EOL}
AT+CONF BLEGATT1={"service" : "CFAO", "chr": "72BD"}{EOL}
OK{EOL}
AT+BLE INIT PERIPHERAL{EOL}
OK{EOL}
AT+BLE ADVERTISE{EOL}
OK{EOL}

    # Now other devices can discover the ExpressLink device
```

```
AT+BLE ADVERTISE CANCEL{EOL}
OK{EOL}
```

13.3.4 BLE GET*[#]* »Synchronous read of a local characteristic«

The BLE GET command allows the host to perform a synchronous read of the value of a local peripheral characteristic. The maximum value that can be retrieved from a characteristic is *MaxBLECharLen* bytes, where *MaxBLECharLen* is a value (>=31) defined by the vendor and documented in the module datasheet.

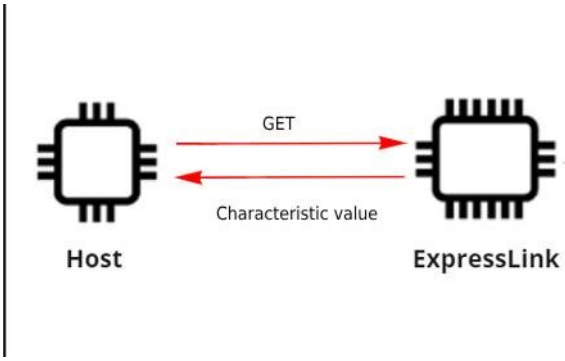


Figure 12 - Reading from a local characteristic in BLE peripheral mode

Returns:

13.3.4.1 `OK` *`{value}{EOL}`*

The GET request was successful, the characteristic value is returned as a hex string.

13.3.4.2 `OK` *`{EOL}`*

On a successful get read, when the characteristic value is an empty string.

13.3.4.3 `ERR8 PARAMETER UNDEFINED`*`{EOL}`*

A BLEGATT# configuration is not set.

13.3.4.4 `ERR7 OUT OF RANGE`*`{EOL}`*

The BLEGATT# numerical suffix is out of bounds (0 or greater than MaxBLEGATT).

13.3.4.5 `ERR25 NOT ALLOWED`*`{EOL}`*

The characteristic can be only be read when the peripheral is initialized.

13.3.4.6 ERR27 BLE ERROR*{EOL}*

When the command fails to successfully read from a characteristic.

Example: Read value of local characteristics configured at BLEGATT1 index.

```
AT+BLE GET1{EOL}
OK 014A{EOL}
```

13.3.5 BLE SET*[#] [payload]* »Write to a local characteristic«

The BLE SET command allows the host to perform a synchronous write to the value of a local peripheral characteristic. TThe maximum value that can be retrieved from a characteristic is *MaxBLECharLen* bytes, where *MaxBLECharLen* is a value (>=31) defined by the vendor and documented in the module datasheet.

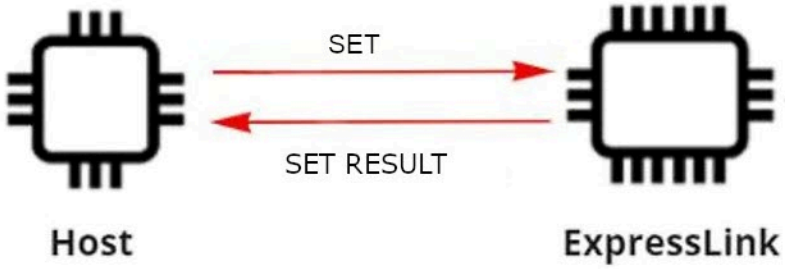


Figure 13 - Writing to a connected BLE device

Returns:

13.3.5.1 OK*{EOL}*

On a successful write to the local characteristic.

13.3.5.2 ERR4 PARAMETER ERROR*{EOL}*

If the payload is not valid hex array.

13.3.5.3 ERR8 PARAMETER UNDEFINED*{EOL}*

If the BLEGATT# configuration is not set.

13.3.5.4 ERR7 OUT OF RANGE*{EOL}*

If the numerical suffix is out of bounds (0 or greater than MaxBLEGATT).

13.3.5.5 ERR25 NOT ALLOWED*{EOL}*

The command can only be executed when initialized as a Peripheral.

13.3.5.6 ERR27 BLE ERROR*{EOL}*

When the command fails to set the value of a local characteristic.

Example- Update the value of local characteristics configured at BLEGATT2 index to 014A:

```
AT+BLE SET2 014A{EOL}
OK{EOL}
```

13.3.6 BLE AUTH*[#] [0/1]* »Authorize access to a characteristic«

The BLE AUTH command allows the host to authorize or deny access to a peripheral characteristic. This is used when a characteristic is configured to require a read or write authorization from the host. When a central device attempts to perform an authorization protected operation, the module configured as peripheral generates a BLE_AUTHORIZE event to notify the host. The host can use this command to authorize or deny access as appropriate for application logic.

Returns:

13.3.6.1 OK*{EOL}*

Acknowledge authorization accepted/denied.

13.3.6.2 ERR4 PARAMETER ERROR*{EOL}*

If the auth code is not Boolean, only 0/1 are valid values.

13.3.6.3 ERR8 PARAMETER UNDEFINED*{EOL}*

BLEGATT# characteristic configuration is not found.

13.3.6.4 ERR7 OUT OF RANGE*{EOL}*

The numerical suffix is out of bounds (0 or greater than MaxBLEGATT).

13.3.6.5 ERR25 NOT ALLOWED*{EOL}*

The command can only be executed when initialized as a Peripheral.

13.3.6.6 ERR27 BLE ERROR*{EOL}*

The command cannot be executed (e.g., the BLE connection was lost).

13.4 Pairing, Bonding and Filtering

13.4.1 AT+BLE PAIR »Initiate pairing as peripheral«

Pairing can be initiated from either end, peripheral and/or central. This command initiates the pairing from the ExpressLink device when configured in Peripheral mode. Successful connection with the central is a pre-requisite. This adds the device to the module's Bond list, i.e. bonding (saving the BLE pairing information) happens automatically after pairing. Setting the correct peripheral advertising configuration before issuing a BLE INIT PERIPHERAL command, is essential. (See BLEPeripheral configuration in Paragraph 13 introduction).

Returns:

13.4.1.1 OK*{EOL}*

Pairing was successful, the central device was added to the Bond list.

13.4.1.2 ERR6 NO CONNECTION*{EOL}*

Pairing not possible, device not connected or not initialized.

13.4.1.3 ERR27 BLE ERROR*{EOL}*

Pairing failed, device could not be paired.

13.4.2 AT+BLE*[#]* PAIR »Initiate pairing as central«

Pairing can be initiated from either end, peripheral and/or central. This command initiates the pairing from the ExpressLink device when configured in Central mode. Successful connection with

a peripheral device identified by the numerical parameter (#) is a pre-requisite. This adds the device to the module's Bond list, i.e. bonding (saving the BLE pairing information) happens automatically after pairing.

Pairing is a process that can require several steps optionally including user input over a long period of time. Once initiated using the AT+PAIR{#} commands, it continues asynchronously. The host is notified of progress via the BLE PAIR event (see Table 4 - ExpressLink event codes). The pairing process is aborted if no response is received within 30 seconds. A BLE PAIR event with a PAIR_FAIL code followed by a BLE DISCONNECT event will be generated.

Returns:

13.4.2.1 `OK`*{EOL}*

Pairing was successful, the peripheral device was added to the Allow list and Bond list.

13.4.2.2 `ERR6 NO CONNECTION`*{EOL}*

Pairing not possible, device not connected or not initialized.

13.4.2.3 `ERR27 BLE ERROR`*{EOL}*

Pairing failed, device could not be paired.

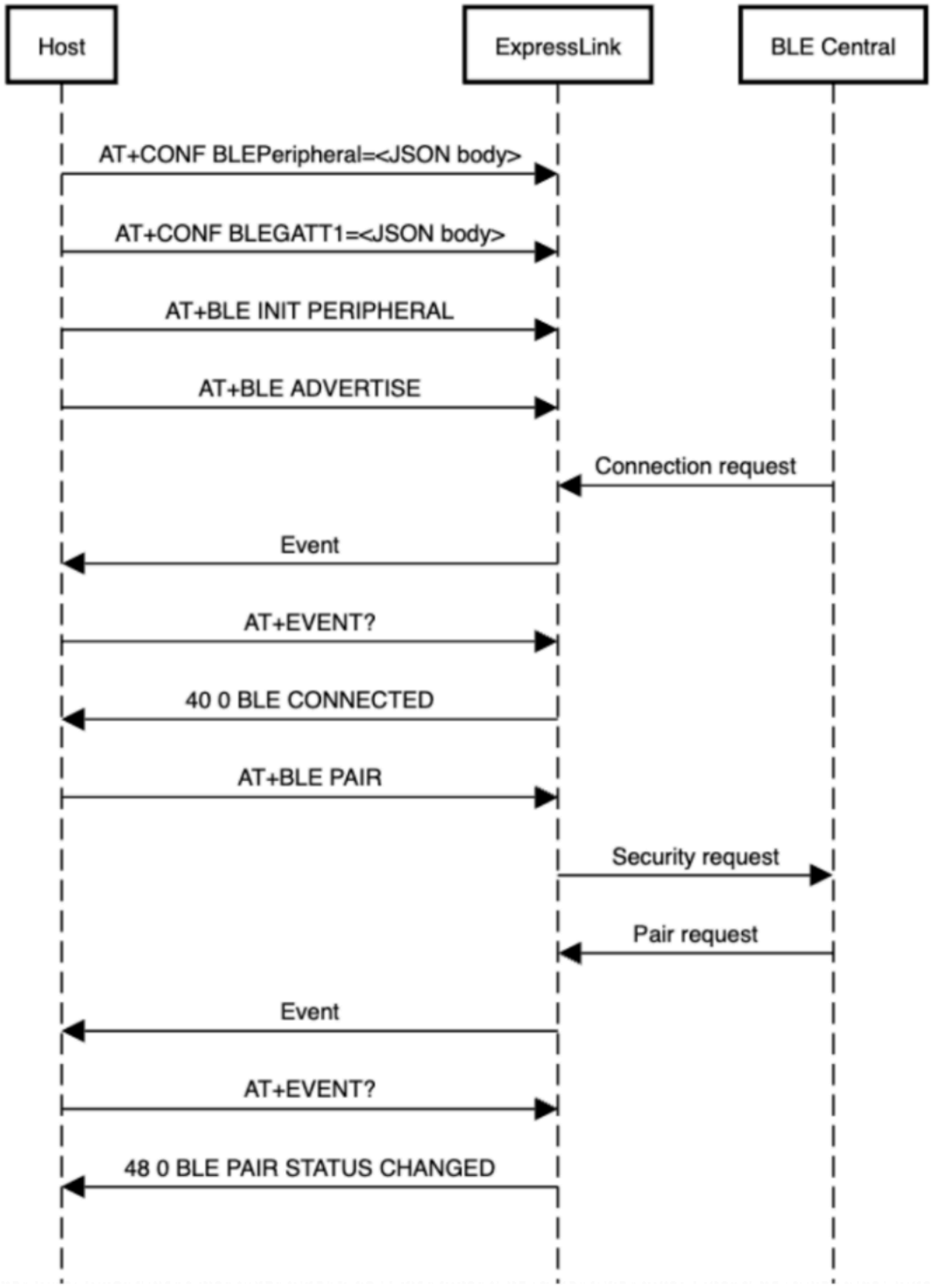


Figure 14 - Typical BLE Peripheral initialization and pairing flow

13.4.3 BLE PAIR? »Query device pairing status«

Fetch the status of the pairing process. This can be polled at any time or in response to a specific BLE PAIR event. If successful, the host can then respond by issuing appropriate PAIR_ACTION(s).

Returns:

13.4.3.1 `OK {code} [detail]`*{EOL}*

See Table 11 – Pairing Codes to interpret the response.

13.4.3.2 `ERR6 NO CONNECTION`*{EOL}*

Device not connected or not initialized.

Table 11 – Pairing Codes

Code	Mnemonic	Description	Detail
0	NONE	No pairing in progress	(empty)
1	PASSKEY_REQ	Request host to provide a 6-digit passkey	(empty)
2	PASSKEY_RECV	Received a 6-digit passkey for host to display	(empty)
3	NC_VALUE	Numeric-comparison value ready, host must display and accept/reject	(empty)
4	PAIR_OK	Pairing completed successfully	(empty)
5	PAIR_FAIL	Security Manager Error – see error code in detail	(error code, hex)

The detailed error codes (hex values) definitions are found in the Bluetooth Core Specification: Vol 3, Part H, § 3.5.5 in the Table 3.7: Pairing Failed reason codes.

13.4.4 BLE PAIR_ACTION SET_PASSKEY {6-digit code} »Supply passkey to controller«

Use this action to supply the 6-digit passkey to the controller after you receive a BLE PAIRING event with pairing code: PASSKEY_REQ (2).

Returns:

13.4.4.1 `OK`*{EOL}*

The controller accepts the passkey and continues the pairing process.

13.4.4.2 `ERR4 PARAMETER ERROR`*{EOL}*

The parameter is missing or malformed (expects a 6-digit decimal code).

13.4.4.3 `ERR17 MODE NOT AVAILABLE`*{EOL}*

No passkey request is pending.

13.4.6 BLE PAIR_ACTION NC {ACCEPT | REJECT} »Accept or reject numerical code«

This command accepts or rejects the numerical code that you receive during the pairing process. Use this in response to pairing code: NC_VALUE(4).

Returns:

13.4.6.1 `OK`*{EOL}*

The controller accepts or rejects the numerical code confirmation.

13.4.6.2 `ERR17 MODE NOT AVAILABLE`*{EOL}*

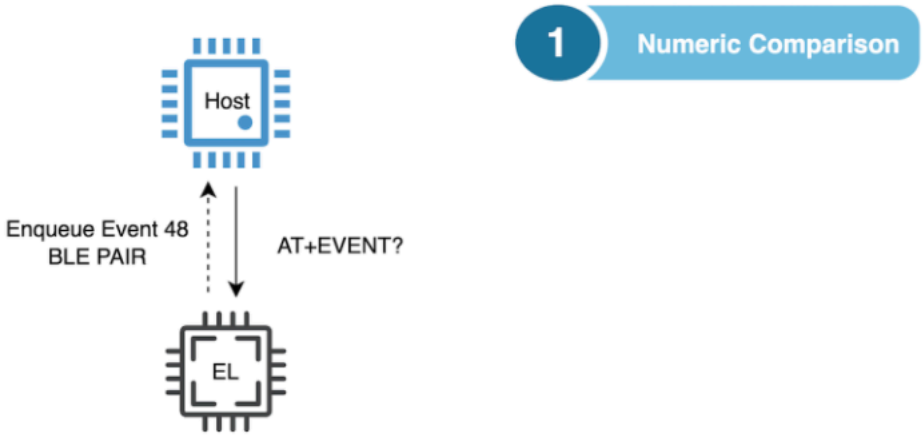
No NC confirmation is pending.

End-to-end Example (Peripheral role)

1. A smartphone initiates pairing and the IoT ExpressLink module raises an event:

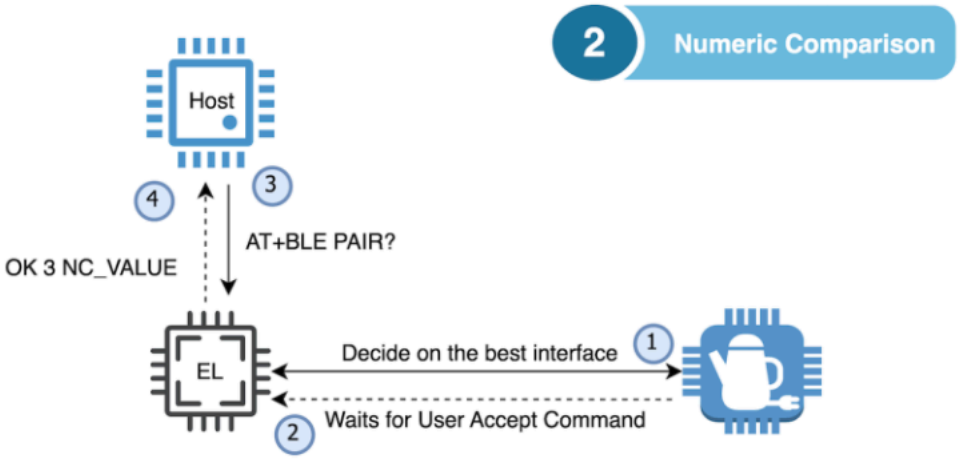
```
> AT+EVENT?
< OK 48 0 BLE PAIR
```

> **Note**
>
> In Central role, the second parameter returned (0) is replaced by the BLEGATT index(#).



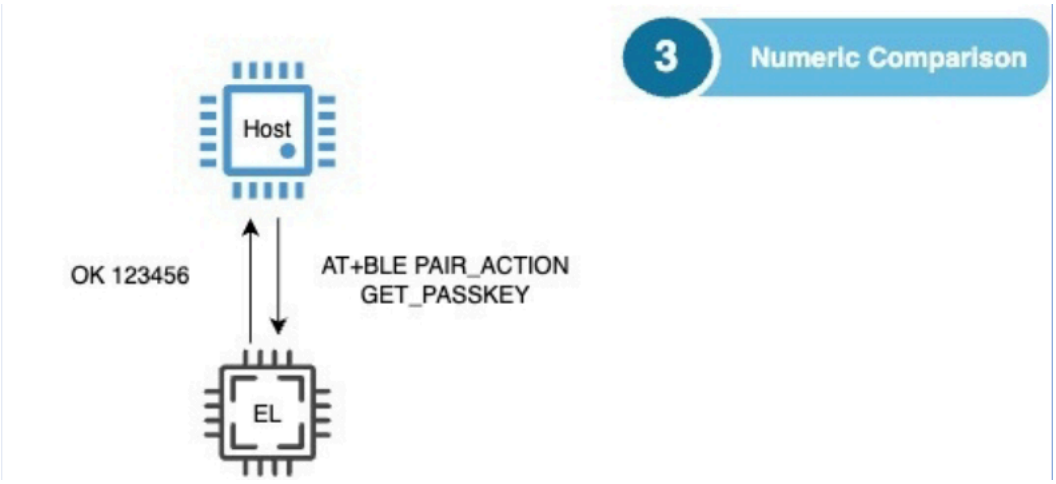
2. The host queries the sub-state:

```
> AT+BLE PAIR?
< OK 3 NC_VALUE
```



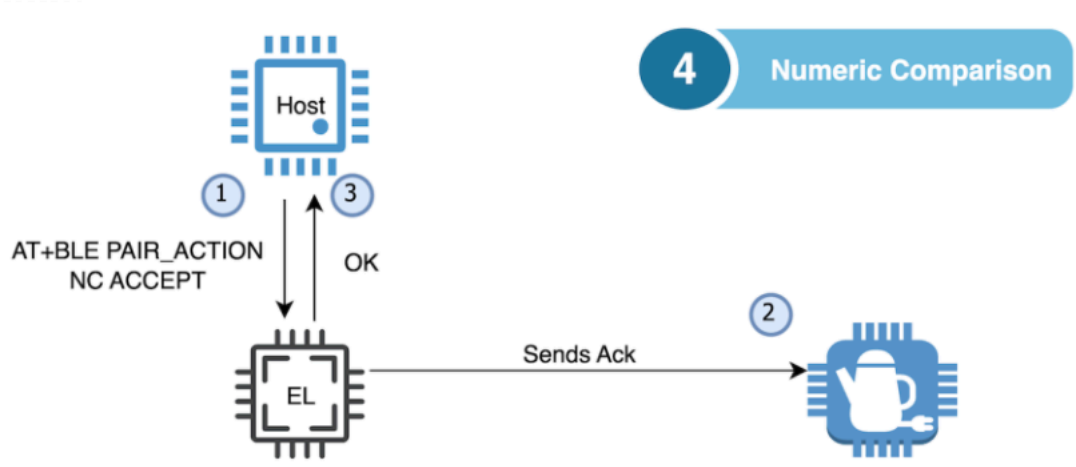
3. Fetch the six-digit number and display it:

```
> AT+BLE PAIR_ACTION GET_PASSKEY
< OK 123456
```



4. The user taps "Yes" on the device to accept pairing:

```
> AT+BLE PAIR_ACTION NC ACCEPT
< OK
```



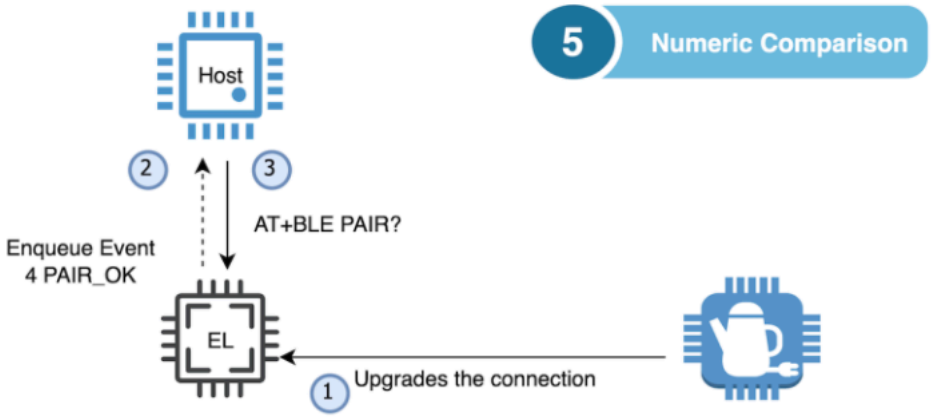
5. Pairing completes and you receive a final notification event:

```
> AT+EVENT?
< OK 48 0 BLE PAIR
```

It's a pairing event!

```
> AT+BLE PAIR?
< OK 4 PAIR_OK
```

Pairing was successful!



13.4.7 AT+BLE UNPAIR »Removes device from the Allow list and Bond list«

This command removes the currently connected and paired device from the Bond list and Allow list. The device must be paired and connected to the ExpressLink device before you can issue this command.

Returns:

13.4.7.1 `OK`*{EOL}*

Unpairing was successful. The device was removed from the Allow list and Bond list.

13.4.7.2 `ERR6 NO CONNECTION`*{EOL}*

Unpairing not possible. Device not connected or not initialized.

13.4.7.3 ERR27 BLE ERROR*{EOL}*

Unpairing failed. Device not paired.

13.4.8 AT+BLE FORGET »Clears the Allow list and Bond list«

This command clears all contents of the Bond list and Allow list.

Returns:

13.4.8.1 OK*{EOL}*

Clearing was successful. All devices were removed from the Allow list and Bond list.

13.4.8.2 ERR25 NOT_ALLOWED*{EOL}*

BLE not initialized.

13.4.8.3 ERR27 BLE ERROR*{EOL}*

Clearing failed.

13.4.9 AT+BLE ALLOW_ADD »Add a device to the Allow list«

Adds the connected device to the Allow list.

Returns:

13.4.9.1 OK*{EOL}*

The device was successfully added to the Allow list.

13.4.9.2 ERR6 NO CONNECTION*{EOL}*

No device connected or BLE not initialized.

13.4.9.3 ERR27 BLE ERROR*{EOL}*

Command failed. Could not add device to AllowList.

13.4.10 AT+BLE ALLOW_REMOVE »Remove a device from the Allow list«

Remove the connected device from the Allow list.

Returns:

13.4.10.1 `OK{EOL}`

The device was successfully removed from the Allow list.

13.4.10.2 `ERR6 NO CONNECTION{EOL}`

No device connected or BLE not initialized.

13.4.10.3 `ERR27 BLE ERROR{EOL}`

Command failed. Could not remove device from AllowList.

13.4.11 AT+BLE ALLOW_CLEAR »Remove all devices from the Allow list«

Remove all devices from the Allow list.

Returns:

13.4.11.1 `OK{EOL}`

All devices were successfully removed from the Allow list.

13.4.11.2 `ERR6 NO CONNECTION{EOL}`

No device connected or BLE not initialized.

13.4.11.3 `ERR27 BLE ERROR{EOL}`

Command failed. Could not clear the AllowList.

> **Note**
>
> The state of the BLE Allow and Bond lists can be queried at any time using the AT CONF? command, and respectively the **>BLEBondList** and **BLEAllowList** configuration parameters.

Querying BLE Lists

You can query the state of the BLE Allow and Bond lists at any time using the AT CONF? command and the BLEBondList and BLEAllowList configuration parameters.

Example 1 - Query Bond List:

```
AT+CONF? BLEBondList{EOL}

Returns:
OK [{"type":"Public","address":"CC:F9:F0:54:11:3F"},
 {"type":"Public","address":"06:05:04:03:02:01"}]{EOL}
# The BLEBondList contains two Public items.

OK []{EOL}
# The BLEBondList is empty.

ERR25 NOT ALLOWED{EOL}
# The BLE functionality has not been initialized.
```

Example 2 - Query Allow List:

```
AT+CONF? BLEAllowList{EOL}

Returns:
OK [{"type":"Public","address":"CC:F9:F0:54:11:3F"},
 {"type":"Public","address":"06:05:04:03:02:01"}]{EOL}
# The BLEAllowList contains two public items.

OK []{EOL}
# The BLEAllowList is empty.

ERR25 NOT ALLOWED{EOL}
# The BLE functionality has not been initialized.
```

14 GPIO control (new with v1.3)

General Purpose I/O control is provided to allow ExpressLink modules to act as I/O expanders for the host processor by allowing control with serial interface of additional pins beyond the basic set defined in section 2.2. This command group is optional, and dependent on the capabilities of the connectivity module and underlying SoC. GPIOs will be numbered 0..MaxIO and made available individually to operate in digital and analog input/output modes, as available. Each I/O has an associated output register which can be set independently of the control mode. Reading I/O inputs will always report the current state of the pin.

14.1.1.1 The number of GPIO pins available (MaxIO), their physical location and DC characteristics (current drive, voltage thresholds) is dependent on the specific module (and underlying SoC) capabilities and must be detailed in the vendor module datasheet.

14.1.1.2 Testing of the GPIO functionality is limited to a few pins during the device qualification, leaving to the partner the responsibility to ensure its proper operation on all pins supported.

14.1.1.3 Pin configuration is volatile and is initially set to all digital inputs after a power cycle or a reset command.

14.1.2 GPIO# SET »Set pin output value to high«

Set the GPIO# pin to a logic high value.

Returns:

14.1.2.1 `OK 1`*`{EOL}`*

Returns the new output register value.

14.1.2.2 `ERR7 OUT OF RANGE`*`{EOL}`*

If the # index is greater than the available number of GPIOs.

14.1.2.3 `ERR25 NOT ALLOWED`*`{EOL}`*

If the GPIO is configured in an analog mode (input or output).

14.1.3 GPIO# CLR »Set pin output value to low«

Sets the GPIO# pin to a logic low value.

Returns:

14.1.3.1 `OK 0`*`{EOL}`*

Returns the new output register value.

14.1.3.2 `ERR7 OUT OF RANGE`*`{EOL}`*

The # index exceeds the available number of GPIOs.

14.1.3.3 ERR25 NOT ALLOWED*{EOL}*

The GPIO is configured in analog mode (input or output).

14.1.4 GPIO# TOGGLE »Toggle output value«

Inverts the output logic value of the GPIO# pin.

Returns:

14.1.4.1 OK *0/1{EOL}*

Returns the new output register value.

14.1.4.2 ERR7 OUT OF RANGE*{EOL}*

The # index exceeds the available number of GPIOs.

14.1.4.3 ERR25 NOT ALLOWED*{EOL}*

The GPIO is configured in analog mode (input or output).

14.1.5 GPIO# OUTPUT »Enable output mode«

Enables the GPIO# pin digital output mode and publishes the current output register value to the pin.

Returns:

14.1.5.1 OK *0/1{EOL}*

Returns the current output register binary value.

14.1.5.2 ERR7 OUT OF RANGE*{EOL}*

The # index exceeds the available number of GPIOs.

14.1.5.3 ERR17 MODE NOT AVAILABLE*{EOL}*

The GPIO cannot be configured as a digital output.

Example:

```
AT+GPIO1 SET{EOL}                    # Set the output register of pin1 to high
OK 1{EOL}

AT+GPIO1 OUTPUT{EOL}                 # Set pin1 to output mode (i.e., to turn on an
 LED)
OK 1{EOL}

AT+SLEEP 1{EOL}                      # Wait for 1 second
OK{EOL}

AT+GPIO1 TOGGLE{EOL}                 # Toggle the pin (turn the LED off)
OK 0{EOL}
```

14.1.6 GPIO# INPUT »Enable input mode«

Sets the pin mode to digital input mode and releases the pin output control.

Returns:

14.1.6.1 OK *0/*1*{EOL}*

Returns the current digital input value.

14.1.6.2 ERR7 OUT OF RANGE*{EOL}*

The # index exceeds the available number of GPIOs.

14.1.6.3 ERR17 MODE NOT AVAILABLE*{EOL}*

The GPIO cannot be configured as a digital input.

14.1.7 GPIO# READ »Read current pin value«

Reads the input value of the GPIO. If the pin is configured for digital input or output modes, this returns the current logic value present on the actual pin. If the GPIO# is configured for analog input or output modes, this returns the current ADC reading or the output register (decimal integer) value.

Returns:

14.1.7.1 OK *{value}{EOL}*

Returns the current GPIO value.

14.1.7.2 ERR7 OUT OF RANGE*{EOL}*

The # index exceeds the available number of GPIOs.

Example:

```
AT+GPIO2 INPUT{EOL}                # Make pin2 an input (i.e., sensing a push button)
OK 1{EOL}                          # Returns current input value (push button not
 pressed)

...

AT+GPIO2 READ{EOL}                 # Read the current value of pin2
OK 0{EOL}                          # Input is low (the push button is being pressed)
```

14.1.8 GPIO# ANALOG »Enable analog input mode«

The maximum integer (IOMaxInt) value is provided in the response to allow the host processor to scale the following readings. This command implementation is optional and must be documented on the device datasheet if available.

14.1.8.1 The IOMaxInt value depends on the module ADC resolution (IOMaxInt = 2^resolution) and must be documented in the module datasheet.

Returns:

14.1.8.2 OK *{IOMaxInt}{EOL}*

Returns the maximum input integer value.

14.1.8.3 ERR7 OUT OF RANGE*{EOL}*

The # index exceeds the available number of GPIOs.

14.1.8.4 ERR17 MODE NOT AVAILABLE*{EOL}*

The GPIO cannot be configured as an analog input.

Example:

```
AT+GPIO3 ANALOG{EOL}               # Configure pin3 as analog input (i.e.,
 potentiometer)
```

```
OK 1024{EOL}                          # A 10-bit resolution ADC is available on this
 module

AT+GPIO3 READ{EOL}                    # Sample pin3 and convert the analog input
OK 512{EOL}                           # The analog input is at mid-scale (potentiometer
 middle)
```

14.1.9 GPIO# DAC »Enable analog output mode«

Disconnects the digital output logic and enables the digital to analog (DAC) feature. Publishes the output value. The maximum integer (IOMaxInt) value is provided in the response to allow the host processor to scale the output values. This command implementation is optional and must be documented on the device datasheet if available.

Returns:

14.1.9.1 OK *{IOMaxInt}{EOL}*

Returns the maximum output integer value.

14.1.9.2 ERR7 OUT OF RANGE*{EOL}*

The # index exceeds the available number of GPIOs.

14.1.9.3 ERR17 MODE NOT AVAILABLE*{EOL}*

The GPIO cannot be configured as an analog output.

14.1.10 GPIO# WRITE {value} »Set a new output register value«

Assigns a new value to GPIO# pin output register. You can use this command to assign non-binary values to the output register (e.g., to assign analog outputs). This command implementation is optional and must be documented on the device datasheet if available.

Returns:

14.1.10.1 OK *{value}{EOL}*

Returns the new output register value.

14.1.10.2 ERR7 OUT OF RANGE*{EOL}*

The # index exceeds the available number of GPIOs.

14.1.10.3

If the assigned new value is greater than IOMaxInt for the GPIO current mode, the new value is *reduced modulo IOMaxInt*.

Example 1 - Analog output write:

```
AT+GPIO4 DAC{EOL}                    # Enable analog output mode
OK 256{EOL}                          # An 8-bit DAC is available on this pin

AT+GPIO4 WRITE 64{EOL}               # Set the output to ¼ of the scale
OK 64{EOL}                           # The actual output value
```

Example 2 - Digital output write:

```
AT+GPIO4 OUTPUT{EOL}                 # Enable digital output
OK 0{EOL}                            # Current output register value

AT+GPIO4 WRITE 33{EOL}               # Write a new value to the output register
OK 0{EOL}                            # The actual output value (33 mod 1)
```

Example 3 - Analog output write, exceeding IOMaxInt:

```
AT+GPIO4 DAC{EOL}                    # Enable analog output mode
OK 256{EOL}                          # An 8-bit DAC is available on this pin

AT+GPIO4 WRITE 356{EOL}              # Set the output to a value larger than IOMaxInt
OK 100{EOL}                          # The actual output value (356 mod IOMaxInt)
```

Appendix – Manufacturer Module Datasheet Requirements

The following is a summary of commands and features that are listed as optional for implementation by the manufacturer. Their availability must be explicitly documented on the device datasheet:

- *EVENT!* – long polling, support host communication purely over the serial interface (see 8.2.2 EVENT! *time* »Wait (blocking) for the next event or timeout«)
- *DIAG* – command group for enhanced diagnostic and debugging purposes (see 8.3.1 DIAG *{command} [optional parameters]* »Perform a diagnostic command«)

- *OTW* – over the wire, proprietary firmware update protocol (see 9.13.1 OTW »Enter firmware update mode«)
- *WHERE?* – request module location information (see 11.1.2 WHERE? »Request location information«)
- *Workshop Wi-Fi Credentials* – default Workshop connectivity credential (see 12.2.1 Workshop Default Wi-Fi Credentials (Optional))
- *BLE* – command group exposing additional Bluetooth Low Energy wireless connectivity (see section 13)

  Modules implementing this command group should also document in the device datasheet the following parameters:

  - MaxBLECentral – the maximum number of Central role configurations supported
  - MaxBLEGATT – the maximum number of Peripheral role configurations supported
  - MaxBLECharLen – length of the BLE characteristic buffer
  - MaxBLESubscriptions – the maximum number of subscriptions
- *GPIO* – command group exposing I/O control for modules with digital and analog I/O expansion (see 14 GPIO control (new with v1.3))

  Modules implementing this command group should also document in the device datasheet the following parameters:

  - MaxIO – the number of I/O pins available, their physical location and DC electrical characteristics (for example, sink and source current and voltage thresholds)
  - IOMaxInt – largest integer value representation of the pin value

The following implementation parameters must be documented by all ExpressLink modules as part of the mandatory command set:

- MaxExec – Maximum command execution time (<= 120 seconds)
- MaxEvent – Depth of the Event queue
- MaxOTARead – Maximum number of bytes that can be requested at once with an OTA READ command
- MaxShadow – Maximum number of Shadow objects the module can handle

Additional documentation is provided in the device datasheet for the following commands:

- SLEEP# – a list of available low power modes, their operating conditions and expected power consumption (see 4.7.4 SLEEP*[#] [duration]* »Request to enter a low power mode«)
- CONNECT – a list of Hint Codes (see 4.7.2 CONNECT »Establish a connection to an AWS IoT Core Endpoint«)
- Device Defender – list of Custom Metrics reported (see 10.1 AWS IoT Device Defender)
- BLE EXCEPTION – list of Hint Codes (see section 8.2.1 EVENT? »Request the next event in the queue«)
- BLE DISCOVER ERROR – list of Hint Codes (see section 13.2.1 BLE*[#]* DISCOVER *[duration|*CANCEL*]* »Scanning and Advertisement«)

Document history

The following table describes important changes to the AWS IoT ExpressLink Programmer's Guide starting with v1.0. We also update the documentation to address any errors found or feedback received.

Change	Description	Date

version 1.3	The following sections were added:
	• 13 Bluetooth Low Energy (BLE) Section 13 was updated with:
	  • BLE Central Device Configuration was updated.
	  • BLE Peripheral Device Configuration was updated.
	  • BLE Characteristics Configuration was updated.
	  • BLE Descriptors Configuration was updated.
	  • 13.3.6 BLE AUTH*[#] [0/1]* »Authorize access to a characteristic« was introduced.
	  • 13.4.11 AT+BLE ALLOW_CLEAR »Remove all devices from the Allow list« was introduced.
	• A new section 14 GPIO control (new with v1.3) was added.
		August 27, 2025

Change	Description	Date

version 1.2	The following sections and tables were updated:
	• Table 1 - Error codes - New error codes were introduced:
	  • 27 : BLE ERROR
	  • 28 : CONFIGURATION ERROR
	• Table 2 - Configuration Dictionary Persistent Keys - Added new non-persistent configuration parameters:
	  • BLECentral#
	  • BLEPeripheral
	  • BLEGATT#
	• Table 4 - ExpressLink event codes - New BLE Events introduced.
	• 13 Bluetooth Low Energy (BLE) - New BLE commands introduced.
		October 27, 2023

Change	Description	Date

version 1.1.2	The following sections and tables were updated:
	• Table 1 - Error codes
	  • Introduction of ERROR 26 INVALID CERTIFICATE.
	• Table 2 - Configuration Dictionary Persistent Keys
	  • 'Version' now allows for a suffix (for example, 'X.Y.Z beta2').
	  • OTACertificate is now write-only.
	• 9 ExpressLink module Updates (formerly Chapter 9)
	  • 9.5 Module OTA signature verification
	  • Underlined sentence requiring OTA Certificate pre-provisioning.
	  • Reworded last sentence for clarity.
	  • 9.6 Module OTA certificate updates
	  • OTACertificate is now write-only.
	  • ERR26 is produced when a corrupted or otherwise invalid certificate is presented.
	  • 9.10 Host OTA Signature Verification
	  • 9.11 Host OTA certificate update
	  • 9.11.2 CONF? *{certificate} pem* »Special certificate output formatting option«
	  • Special 'pem' qualifier is now case insensitive (for example, 'pem' | 'PEM').
	  • 9.11.3 CONF {certificate}=pem »Special certificate input formatting option«
	  • Special 'pem' qualifier is now case insensitive (for example, 'pem' | 'PEM').
		July 25, 2023

Change	Description	Date
version 1.1.1	The following sections were added: • 10 AWS IoT Services The following sections and tables were updated: • 4.3 Delimiters and escaping • 4.4 Maximum values • Table 1 - Error codes • 4.7.2 CONNECT »Establish a connection to an AWS IoT Core Endpoint« • 4.7.7 FACTORY_RESET »Request a factory reset of the ExpressLink module« • 5.1.7 SUBSCRIBE*[#]* »Subscribe to Topic#« • 5.1.8 UNSUBSCRIBE*[#]* »Unsubscribe from Topic#« • Table 2 - Configuration Dictionary Persistent Keys • Table 3 - Configuration dictionary non-persistent keys • Table 4 - ExpressLink event codes • 12 Provisioning and Onboarding The following sections were removed: • 4.1.3 SEND *{topic} message* »Publish msg on the specified topic« • 6 Status dictionary note that subsequent sections were renumbered • 9 Additional services	November 17, 2022

Change	Description	Date
version 1.0	Initial release.	June 20, 2022

Archive

The following archive versions of this Programmer's Guide are available:

- AWS IoT ExpressLink Programmer's Guide **v1.2**
- AWS IoT ExpressLink Programmer's Guide **v1.1.2**
- AWS IoT ExpressLink Programmer's Guide **v1.1.1**
- AWS IoT ExpressLink Programmer's Guide **v1.0**
- AWS IoT ExpressLink Programmer's Guide **v0.5**