aws

Programmer's Guide

# AWS IoT ExpressLink

# AWS IoT ExpressLink: Programmer's Guide

# Table of Contents

# AWS IoT ExpressLink programmer's guide v1.1.2

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**

- CONF {certificate}=pem »Special certificate input formatting option«
- CONF? *key* »Read the value of a configuration parameter«
- CONF? *{certificate} pem* »Special certificate output formatting option«
- CONFMODE *[parameter]* »Activate modal credential entry«
- CONNECT »Establish a connection to an AWS IoT Core Endpoint«
- CONNECT! »Non-blocking request to connect to IoT Core«
- CONNECT? »Request the connection status«
- DIAG *{command} [optional parameters]* »Perform a diagnostic command«
- DISCONNECT »Leave the connected state and enter the active state«
- EVENT? »Request the next event in the queue«
- FACTORY_RESET »Request a factory reset of the ExpressLink module«
- GET »Request next message pending on any topic«
- GET0 »Request next message pending on an unassigned topic«
- GET*[#]* »Request next message pending on the indicated topic«
- OTA ACCEPT »Allow the OTA operation to proceed«
- OTA APPLY »Authorize the ExpressLink module to apply the new image.«
- OTA CLOSE »The host OTA operation is completed«
- OTA FLUSH »The contents of the OTA buffer are emptied«
- OTA READ *#bytes* »Requests the next # bytes from the OTA buffer«
- OTA SEEK *{address}* »Moves the read pointer to an absolute address«
- OTA? »Fetches the current state of the OTA process«
- OTW »Enter firmware update mode«
- RESET »Request a full reset of the ExpressLink internal state«
- SEND*[#] message* »Publish msg on a topic selected from topic list«
- SHADOW*[#]* DELETE »Request the deletion of a Shadow document«
- SHADOW*[#]* DOC »Request a Device Shadow document«
- SHADOW*[#]* GET DELETE »Request a Shadow delete response«
- SHADOW*[#]* GET DELTA »Retrieve a Shadow Delta message«
- SHADOW*[#]* GET DOC »Retrieve a device shadow document«

**Tables**



# 1 Overview

An ExpressLink module is a connectivity module connected via serial interface (initially UART based) that uses an abstracted Application Programming Interface (API) to connect any host application to 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 heterogenous fleets.

## 1.2 Specifications

The specifications in this document are the minimum set of requirements needed by the ExpressLink module to provide connectivity and basic support for AWS services. The ExpressLink manufacturer (third party) can provide additional pins and commands that can present key differentiators of their unique product or technology within the guidelines provided in this document. Testing and AWS qualification of the device will be based on this specification and not on any additional features or functionality.

# 2 Hardware

The ideal hardware requirements are based on:

- a FreeRTOS qualified MCU, capable of network connectivity (ethernet, Wi-Fi, cellular)
- a minimum of eight I/Os
- a pre-provisioned secure element or equivalent secure enclave providing cryptographical hardware acceleration, random number generation and secure key storage
- a non-volatile memory providing bulk storage sufficient to support the module's own over-the-air updates (OTA) and host processor OTA (HOTA)

## 2.1 Block diagram

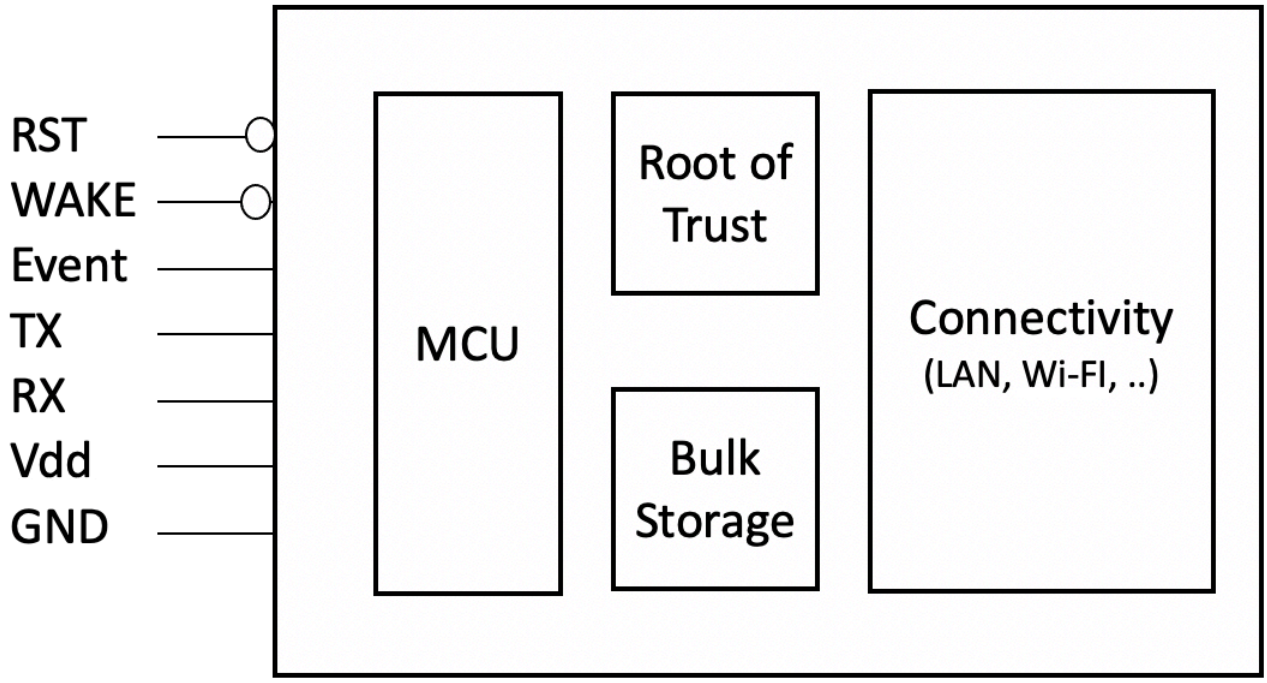


Figure 1 - Simplified block diagram

## 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 7.2.1 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 2 for a partial representation).

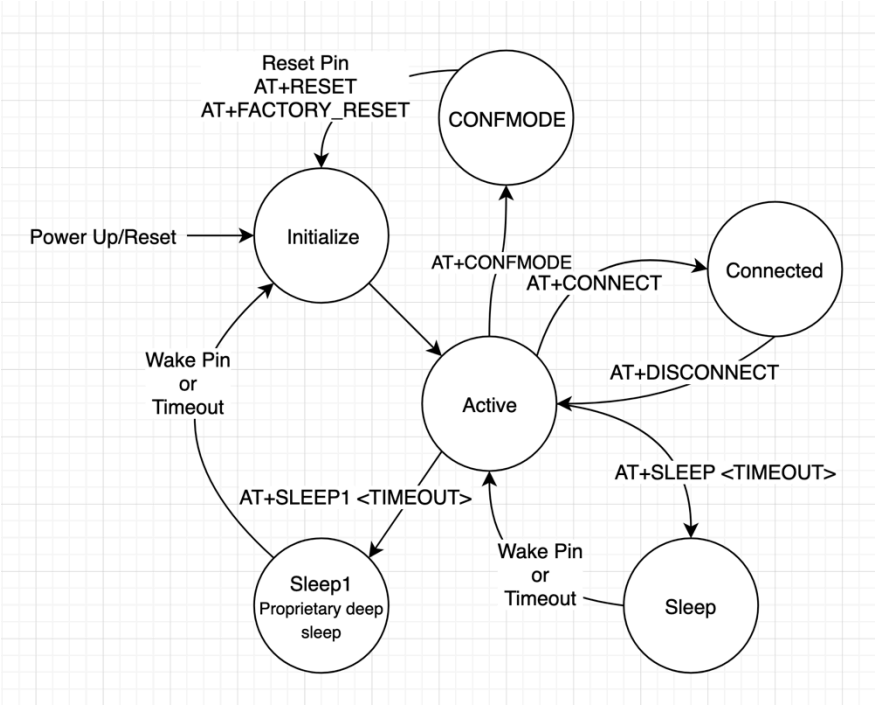


Figure 2 - ExpressLink internal states diagram (partial)

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 7.2 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 *[#]*

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'.

### 4.2.3 *{separator}*

A single ASCII space character (0x20).

**Returns:**

#### 4.2.3.1 ERR2 PARSE ERROR

If ANY character other than 0x20 is present after the numerical suffix or '?' in the command string, then the module returns 'ERR2 PARSE ERROR'.

### 4.2.4 *[parameter]*

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

**Returns:**

#### 4.2.4.1 ERR4 PARAMETER ERROR

If the command is unable to process the parameter supplied, then the module returns 'ERR4 PARAMETER ERROR'.

### 4.2.5 *{EOL}*

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

### 4.2.6 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

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

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

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

## 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 has been 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).

**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 reference s 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. |

| Code | ExpressLink text | Description |
|---|---|---|
| 12 | KEY READONLY | The key cannot be written. |
| 13 | KEY WRITEONLY | The key cannot be read. |
| 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. |

| Code | ExpressLink text | Description |
|---|---|---|
| 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. |

> **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 11.3.2 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 Cloud, bringing an active device into a higher power consumption mode where it is able to communicate with the AWS IoT Core endpoint.

> **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.<br><br>**Tip**<br>Check SSID/passphrase or local router state.<br><br>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.<br><br>**Tip**<br>Check the endpoint configuration parameter (URL)<br><br>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).<br><br>**Tip**<br>Check if the AWS root certificate might have expired.<br><br>After this step, a secure socket is established with AWS. |

| | |
|---|---|
| 5. | **Failed to login AWS (MQTT) broker** – reported when the MQTT login is unsuccess ful<br><br>**Tip**<br>Check if the device certificate is present in the customer account registry.<br><br>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)<br><br>**Tip**<br>Check policies attached to device certificate.<br><br>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').

### 4.7.4 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 increasing values (*[#]*) that correspond to deeper sleep states (as capable) to provide the lowest power consumption and longest possible battery life. 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** 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 7 Event handling.)

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«

ExpressLink modules that require additional user credentials can be set by the host to enter CONFMODE (see Figure 2) to enable or repurpose an interface to receive additional connection credentials from user input. Consult 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 cloud, bringing an active device into a higher power consumption mode where it is able to communicate with the AWS IoT Core endpoint.

> **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 (only 0 and 1 are supported) at the time the command is issued determines the applicable Quality of Service.

**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 9.2 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«) 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.

**5.1.8.1** Use the topic at the specified index.

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
```

**Returns:**

#### 5.1.8.2 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.

#### 5.1.8.3 ERR6 NO CONNECTION

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

#### 5.1.8.4 ERR8 PARAMETER UNDEFINED

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

#### 5.1.8.5 ERR7 OUT OF RANGE

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

# 6 Configuration Dictionary

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

| Configuration Parameter | Type | Initial Value | Factory Reset | Buff Size | Description |
|---|---|---|---|---|---|
| | | | | | the HW root of trust and present in the device certificate (also see 11.1.3.1) 11.1.3 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 11.1.3 ExpressLink Birth Certificate). |

| Configuration Parameter | Type | Initial Value | Factory Reset | Buff Size | Description |
|---|---|---|---|---|---|
| CustomName | R/W | {empty} | Y | ≥128 | Custom Product Name, can be set by the host (also see 11.1.5 ExpressLink MQTT Login signature. |
| Endpoint | R/W | Staging account endpoint | Y | ≥128 | The endpoint of the AWS account to which the ExpressLink module connects (also see 11.3.2 ExpressLink onboarding states and transitions). |

| Configuration Parameter | Type | Initial Value | Factory Reset | Buff Size | Description |
|---|---|---|---|---|---|
| RootCA | R/W | AWS root CA | N | ≥4KB | The server root certificate that will be used to authenticate the cloud Endpoint (also see 8.12 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 8.10 Host OTA Signature Verification). |
| OTAcertificate | W | Vendor OTA Certificate | N | ≥4KB | Module OTA certificate. Vendor and Model specific (see 8.5 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). |

| Configuration Parameter | Type | Initial Value | Factory Reset | Buff Size | Description |
|---|---|---|---|---|---|
| APN | R/W | {default} | Y | 128 | Access Point Name (Cellular modules only). |

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**

| Configuration 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<MaxTopic> | R/W | {Empty} | | Custom defined topic MaxTopic |
| EnableShadow | R/W | 0 | 1 | 0 - disabled, or 1 - enabled |

| Configuration Parameter | Type | Initial Value | Buff Size | Description |
|---|---|---|---|---|
| Shadow1 | R/W | {Empty} | 64 | Custom defined named shadow |
| ... | | | | |
| Shadow<MaxShadow> | R/W | {Empty} | | Custom defined named shadow |

# 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.

**Returns:**

#### 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.

**Returns:**

#### 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 Data accessed through the CONF command

### 6.2.1 CONF KEY=*{value}* »Assignment«

Assign a value to a configuration parameter present in the configuration dictionary. (See 8.11.2 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 8.11 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 Event handling

## 7.1 Introduction

Events are asynchronous messages on one of the subscribed topics that the ExpressLink module has received and queued. They can also be error messages that reflect an unexpected change in the module's internal state.

Events are appended to the module event queue (FIFO). The host can poll the event queue periodically. Or, if connected, it can poll the event queue following an interrupt (rising edge) on the EVENT pin.

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

**7.1.1.2** 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.

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

## 7.2 Event handling commands

### 7.2.1 EVENT? »Request the next event in the queue«

**Returns:**

**7.2.1.1** `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.

**7.2.1.2** `OK`*`{EOL}`*

If the event queue is empty, then the 'OK' response is followed immediately by *{EOL}*.

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 |
|---|---|---|---|
| 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..19 | - | - | RESERVED |
| 20 | Shadow Index | SHADOW INIT | Shadow[Shadow Index] interface was |

| Event Identifier | Parameter | Mnemonic | Description |
|---|---|---|---|
| | | | initialized successfu lly. |
| 21 | Shadow Index | SHADOW INIT FAILED | The SHADOW[Sh adow Index] interface initialization failed. |
| 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. |
| ≤ 999 | - | | RESERVED. |
| ≥1000 | - | | Available for custom implementation. |

**7.2.1.3** Sleep, reset, and factory reset commands automatically clear all events pending.

## 7.3 Diagnostic commands (not covered by test)

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

A number of diagnostic commands can be added to assist the developer in their debugging efforts. These commands are implementation specific and depend on the media and type of module. 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.

The following are examples of possible diagnostic commands for a Wi-Fi module:

Example 1:

```
AT+DIAG PING xxx.xxx.xxx.xxx    # Initiate a Ping of the IP address provided
```

Example 2:

```
AT+DIAG SCAN seconds    # Initiates a SCAN of nearby Wi-Fi access points with a
 timeout
```

# 8 ExpressLink module updates

> **Note**
>
> The OTA service is supported only when the device is in the *onboarded* state (see 11.3.2 ExpressLink onboarding states and transitions), that is, only when the module is connected to the customer's AWS account.

ExpressLink modules natively support firmware updates utilizing the AWS IoT OTA service (as currently implemented in the AWS Embedded C-SDK v.202103.00) and Over the Wire (OTW). 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 8.2 OTA commands.)

## 8.1 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 can be represented in the following diagram:

#### 8.1.1.1 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. |

| ExpressLink module | Host Processor |
| --- | --- |
| | READ 1024 – Requests the first chunk of payload data. |
| | |
| Delivers first chunk of payload data and advances pointer. | |
| The process repeats until the entire payload is transferred to the host processor.<br><br>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.

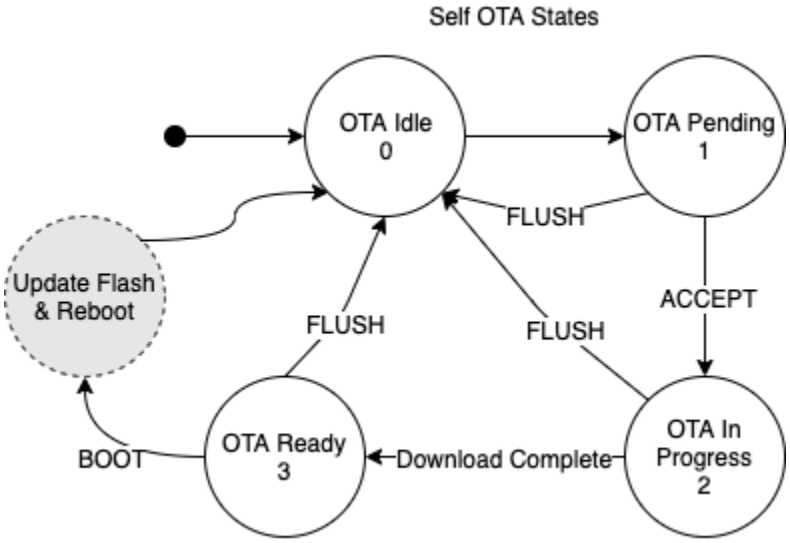


Figure 3 - ExpressLink module OTA state diagram

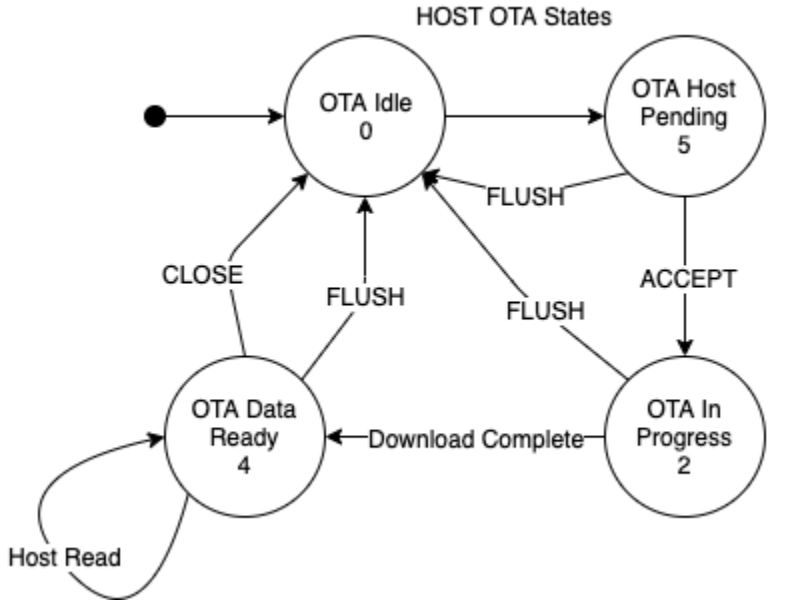


Figure 4 - ExpressLink Host OTA state diagram

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

# 8.2 OTA commands

## 8.2.1 OTA? »Fetches the current state of the OTA process«

**Returns:**

OK *{code} {detail}*

## 8.2.2 OTA codes

| | |
|---|---|
| 0 | No OTA in progress. |
| 1 | A new module OTA update is being proposed. The host can inspect the version number and decide to accept or reject it. The *{detail}* field provides the version information (string). |
| 2 | A new Host OTA update is being proposed. The host can inspect the version details and decide to accept or reject it. The *{detail}* field provides the metadata that is entered by the operator (string). |
| 3 | OTA in progress. The download and signature verification steps have not been completed yet. |
| 4 | A new module firmware image has arrived. The signature has been verified and the ExpressLink module is ready to reboot. (Also, an event was generated.) |
| 5 | A new host image has arrived. The signature has been verified and the ExpressLink module is ready to read its contents to the host. The size of the file is indicated in the response detail. (Also, an event was generated.) |

Example 1:

```
AT+OTA?    # check the OTA status
OK 3       # an OTA operation is in progress, the module OTA buffer is in use
```

Example 2:

```
AT+OTA?        # check the OTA status
OK 1 v2.5.7    # a module OTA firmware update is proposed
```

**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.

### 8.2.3 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:**

#### 8.2.3.1 OK*{EOL}*

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

#### 8.2.3.2 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                     # 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
```

### 8.2.4 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:**

#### **8.2.4.1** 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 *#bytes* a module can read 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
```

#### 8.2.4.2 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.

### 8.2.5 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
```

#### 8.2.5.1 ERR20 INVALID ADDRESS

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

#### 8.2.5.2 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.

### 8.2.6 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:**

**8.2.6.1** `OK`*`{EOL}`*

The module has initiated a boot sequence.

**8.2.6.2** `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.

**8.2.6.3** `ERR21 INVALID OTA UPDATE`

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

**8.2.6.4** `ERR23 INVALID SIGNATURE`

The new image signature check failed.

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

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

**8.2.6.7** 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).

### 8.2.7 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:**

#### 8.2.7.1 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.

### 8.2.8 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:**

#### 8.2.8.1 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.

## 8.3 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 that communicates the nature of the incoming OTA payload, the file signing method (if used), and a number of additional options are provided by the user and transferred to the ExpressLink module in the form a JSON string. ExpressLink devices require the fileType attribute to be set to values according to Table 5:

**Table 5 - Reserved OTA file type codes (0-255)**

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

| fileType | Reserved for | Signature | Certificate | Request Host Permission |
|---|---|---|---|---|
| | | l signing is required. | | |

[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.

## 8.4 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.

## 8.5 Module OTA signature verification

In order for ExpressLink modules to validate module OTA updates, they must be provided at the time they are manufactured with an OTA certificate that contains the public key that corresponds to the private key that will be used by the manufacturer to sign the update image files.

Module OTA certificates are not secret and therefore are not required to reside in the secure key storage of the module, but they must be stored "independently" of the module binary image. They can be stored in a separate non-volatile *memory partition*, or simply at a reserved address that is *not overwritten* during the OTA update process. If the module OTA certificate were statically linked

in (that is, were part of) the binary module image, it would become impossible for customers to *skip updates*, forcing a strictly continuous sequence because each new signature might depend on a new certificate (possibly) included in one (or each) of the previous updates.

Module OTA jobs will include additional metadata to clearly identify the module's *manufacturer, model, and version (major and minor)* to allow the ExpressLink executive to discard incorrect OTA images *ahead of time*, and so prevent the costly and lengthy download altogether.

Note how this metadata is input by the customer's dev-op at the time the Job is created and could therefore be incorrectly associated with a firmware update file meant for an incompatible model from the same vendor. The use of a unique *signature profile (and key) per model* acts as an additional safeguard to ensure that incompatible binary images will be rejected during the secure-boot process.

## 8.6 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 the serial API:

- Module OTA certificate updates performed using OTA use the fileType code indicated in Table 5 - Reserved OTA file type codes (0-255) (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:**

#### 8.6.1.1 OK*{EOL}*

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

#### 8.6.1.2 ERR23 INVALID SIGNATURE

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

#### 8.6.1.6 ERR26 INVALID CERTIFICATE

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

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

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

**8.6.1.5** Module OTA certificates are NOT affected by a factory reset.

## 8.7 Module OTA override

As described in **8.1.1.1** 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.

## 8.8 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.

## 8.9 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.

## 8.10 Host OTA Signature Verification

Metadata provided during OTA Job creation indicates to the module whether the optional signature verification is required.

A *host OTA certificate* that contains the public key that corresponds to the customer's private host OTA signing key may be provided by the customer.

The certificates used for the host OTA signature validation are accessible for reading by means of the serial API (see the CONF? command).

## 8.11 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 - Reserved OTA file type codes (0-255). (See the "Host OTA certificate update" entry.)

**8.11.1.1** 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.

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

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

### 8.11.2 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:**

**8.11.2.1** `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-----
```

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

The special value **pem** (case insensitive) can be used to input a certificate (OTAcertificate, HOTAcertificate, RootCA) 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:**

### 8.11.3.1 OK*{EOL}*

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

### 8.11.3.2 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.

## 8.12 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 - Reserved OTA file type codes (0-255) (Server Root certificate update).

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

**8.12.1.2** Server Root certificates are NOT deleted upon a factory reset

## 8.13 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).

### 8.13.1 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:**

**8.13.1.1** `OK`*`{EOL}`*

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

**8.13.1.2** 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.

**8.13.1.3** 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:**

**8.13.1.4** OK*{EOL}*

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

**8.13.1.5** 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).

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

**8.13.1.7** 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.

**8.13.1.8** 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.

# 9 AWS IoT Services

## 9.1 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 those shown in Table 6.

**Table 6 - 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 and the Module name in 11.1.5.3).

> **Note**
>
> Access to the AWS IoT Device Defender service is available only when the device is in the *onboarded* state (see 11.3.2 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
```

## 9.2 AWS IoT Device Shadow

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

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 11.3.2 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.

Because 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.

**9.2.1.1** 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 prepopulated 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.

**9.2.1.2** 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.

**9.2.1.3** 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«).

### 9.2.2 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.

**9.2.2.1** 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.

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

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

**9.2.2.4** 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:**

**9.2.2.5** `OK`

The Device Shadow service initialization process has started.

**9.2.2.6** `ERR7 OUT OF RANGE`

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

**9.2.2.7** `ERR8 PARAMETER UNDEFINED`

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

**9.2.2.8** `ERR6 NO CONNECTION`

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

**9.2.2.9** `ERR24 SHADOW ERROR`

Shadow support is disabled (the *EnableShadow* configuration parameter set to 0 or the device is not in the *onboarded* state, see 11.3.2 ExpressLink onboarding states and transitions).

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

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

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

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

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

**Returns:**

**9.2.3.4** `OK`

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

**9.2.3.5** `ERR7 OUT OF RANGE`

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

**9.2.3.6** `ERR8 PARAMETER UNDEFINED`

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

**9.2.3.7** `ERR6 NO CONNECTION`

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

**9.2.3.8** `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 11.3.2 ExpressLink onboarding states and transitions).

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

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

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

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

**Returns:**

**9.2.4.3** `OK`

The requested shadow document has not arrived yet.

**9.2.4.4** `OK 1 {document}`

The requested shadow document has arrived.

**9.2.4.5** `OK 0 {detail}`

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

**9.2.4.6** `ERR7 OUT OF RANGE`

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

**9.2.4.7** `ERR8 PARAMETER UNDEFINED`

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

**9.2.4.8** `ERR24 SHADOW ERROR`

Shadow support is disabled (the *EnableShadow* configuration parameter set to 0), or the device is not in the *onboarded* state (see 11.3.2 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!
```

### 9.2.5 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.

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

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

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

**Returns:**

**9.2.5.4** `OK`

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

**9.2.5.5** `ERR4 PARAMETER ERROR`

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

**9.2.5.6** `ERR7 OUT OF RANGE`

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

**9.2.5.7** `ERR8 PARAMETER UNDEFINED`

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

**9.2.5.8** `ERR6 NO CONNECTION`

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

**9.2.5.9** `ERR24 SHADOW ERROR`

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

**9.2.5.10** `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 11.3.2 ExpressLink onboarding states and transitions), then the module returns 'SHADOW ERROR'.

### 9.2.6 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.

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

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

**Returns:**

**9.2.6.3** `OK`

A shadow document update response has not arrived yet.

**9.2.6.4** `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.

**9.2.6.5** `ERR7 OUT OF RANGE`

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

**9.2.6.6** `ERR8 PARAMETER UNDEFINED`

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

**9.2.6.7** ERR24 SHADOW ERROR

Shadow support is disabled (the *EnableShadow* configuration parameter set to 0), or the device is not in the *onboarded* state (see 11.3.2 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.
```

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

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

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

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

**9.2.7.3** 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:**

**9.2.7.4** OK

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

**9.2.7.5** ERR7 OUT OF RANGE

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

#### 9.2.7.6 ERR8 PARAMETER UNDEFINED

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

#### 9.2.7.7 ERR6 NO CONNECTION

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

#### 9.2.7.8 ERR24 SHADOW ERROR

Shadow support is disabled (the *EnableShadow* configuration parameter set to 0), or the device is not in the *onboarded* state (see 11.3.2 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.
```

### 9.2.8 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.

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

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

**Returns:**

**9.2.8.3** OK

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

**9.2.8.4** ERR7 OUT OF RANGE

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

**9.2.8.5** ERR8 PARAMETER UNDEFINED

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

**9.2.8.6** ERR6 NO CONNECTION

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

**9.2.8.7** ERR24 SHADOW ERROR

Shadow support is disabled (the *EnableShadow* configuration parameter set to 0), or the device is not in the *onboarded* state (see 11.3.2 ExpressLink onboarding states and transitions).

Example:

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

### 9.2.9 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).

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

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

**Returns:**

**9.2.9.3** OK

An empty string indicates no delta pending.

**9.2.9.4** OK *{delta document}*

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

**9.2.9.5** ERR7 OUT OF RANGE

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

**9.2.9.6** ERR8 PARAMETER UNDEFINED

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

**9.2.9.7** ERR24 SHADOW ERROR

Shadow support is disabled (the *EnableShadow* configuration parameter set to 0), or the device is not in the *onboarded* state (see 11.3.2 ExpressLink onboarding states and transitions).

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

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

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

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

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

**Returns:**

**9.2.10.4** OK

The request was sent to the Device Shadow service.

**9.2.10.5** ERR7 OUT OF RANGE

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

**9.2.10.6** ERR8 PARAMETER UNDEFINED

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

**9.2.10.7** `ERR6 NO CONNECTION`

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

**9.2.10.8** `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 11.3.2 ExpressLink onboarding states and transitions).

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

Check if a Device Shadow Delete request was accepted.

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

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

**Returns:**

**9.2.11.3** `OK`

The request was sent to the Device Shadow service.

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

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

**9.2.11.5** `ERR7 OUT OF RANGE`

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

**9.2.11.6** `ERR8 PARAMETER UNDEFINED`

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

**9.2.11.7** `ERR24 SHADOW ERROR`

Shadow support is disabled (the *EnableShadow* configuration parameter set to 0), or the device is not in the *onboarded* state (see 11.3.2 ExpressLink onboarding states and transitions).

### 9.2.12 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

Example1:

```
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.
```

Example2:

```
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 Additional services

### 10.1.1 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:**

**10.1.1.1** 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.

**10.1.1.2** ERR15 TIME NOT AVAILABLE

A recent time fix could not be obtained.

### 10.1.2 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:**

**10.1.2.1** 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.

**10.1.2.2** ERR16 LOCATION NOT AVAILABLE

A location fix could not be obtained.

# 11 Provisioning

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.

# 11.1 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).

## 11.1.1 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).

## 11.1.2 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).

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

## 11.1.3 ExpressLink Birth Certificate

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

- **`11.1.3.1`** 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.
- **`11.1.3.2`** 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.
- **`11.1.3.3`** The expiration date is set to no less than 10 years from the device certificate issue.

## 11.1.4 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.

### 11.1.5 ExpressLink MQTT Login signature

The ExpressLink module presents an "MQTT login string" formatted as follows (as a single line string):

```
?SDK=RTOS &SDKVersion=x.y.z &Platform=ExpressLink &Metadata=( Vendor=<vendor-name>;
 Model=<model-name>; FWversion=X.Y.Z; TechSpec=0.9.1; CustomName=<custom-product-name>)
```

This string is logged by AWS IoT Core and allows the collection of meta data that is used to assess the effective usage of ExpressLink modules, provide diagnostic information and measure the successful completion of the onboarding process.

- **SDK**: is the RTOS used by the ExpressLink module (for example, "FreeRTOS").
- **SDKVersion**: is the semantic version of the RTOS kernel used (for example, "v10.4.2").
- **Platform**: must match the string "ExpressLink".
- **Metadata**: provides further detail as a list of additional key-value pairs using the following grammar (`<key>=<value> *`). The Metadata value string enclosed in parentheses is formatted as follows:
  - **11.1.5.1** Keys and values are expressed using ASCII alphanumeric characters (0-9, A-Z, a-z), including the dash (-), period (.) and space.
  - **11.1.5.2** Leading/trailing spaces are ignored and automatically trimmed from keys and values.
  - **11.1.5.3** The following key/value pairs are required:
    - **Vendor**: manufacturer of the module.
    - **Model**: uniquely identifies the specific model. NOTE: Vendor and Model combine to form the persistent configuration parameter "About" (see Table 2 - Configuration Dictionary Persistent Keys) with a max length of 64 characters.
    - **FWversion**: semantic version of the manufacturer module firmware (for example, "1.1.13"). Also see the persistent configuration parameter "Version" in Table 2 - Configuration Dictionary Persistent Keys.
    - **TechSpec:** semantic version of the module technical specifications target (for example, "0.9.1"). Also see the persistent configuration parameter "TechSpec" in Table 2 - Configuration Dictionary Persistent Keys.

- **CustomName**: a custom field as configured by the host processor (see Table 2 - Configuration Dictionary Persistent Keys). Also see the persistent configuration parameter "CustomName" in that same table.

- **11.1.5.4** The Metadata string (contained in parentheses) will not exceed a maximum length of 256 characters. Should the concatenation of the prescribed key/value pairs exceed this maximum value, it will be right-truncated at the expense of the CustomName value.

- Example of Metadata:

```
(Vendor=ABC; Model= A-1234; FWversion = X.Y.Z; TechSpec =0.9.1; CustomName =
 Toaster 3000 )
```

  Note the uneven use of spaces in the example above is interpreted correctly by trimming leading and trailing spaces as per 11.1.5.2.

## 11.2 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.

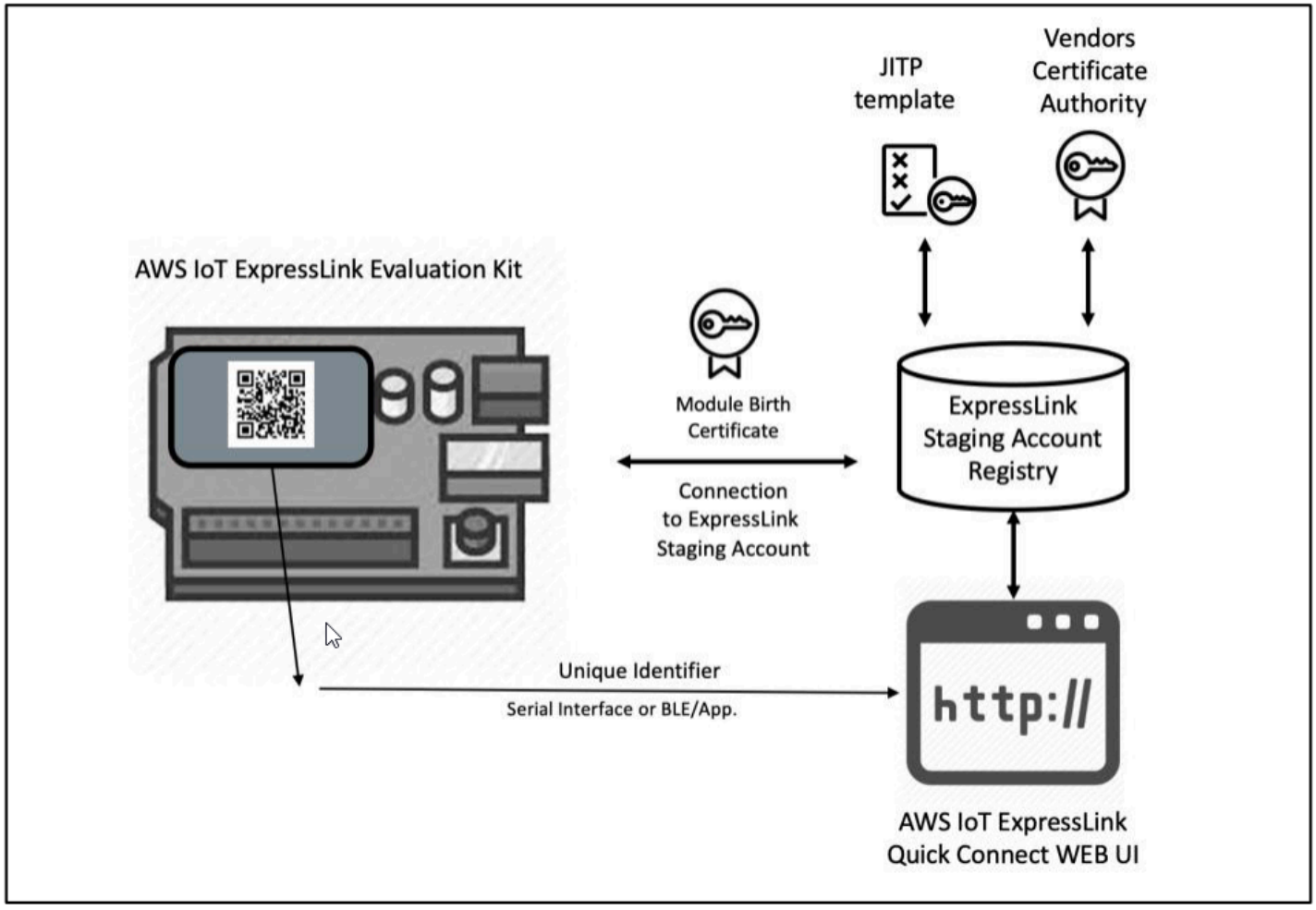


Figure 5 - ExpressLink Evaluation Kit Quick Connect flow

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

### 11.2.1 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 to avoid several (10+) users attempting to simultaneously use a CONFMODE (Bluetooth) connection. This greatly simplifies the room set up.

If implemented, the manufacturer documents such credentials in the module datasheet.

## 11.3 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).

### 11.3.1 ExpressLink late binding 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 11.3.2 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.

### 11.3.2 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:

**11.3.2.1** 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.

**11.3.2.2** 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).

**11.3.2.3** 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.

**11.3.2.4** 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).

**11.3.2.5** 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.

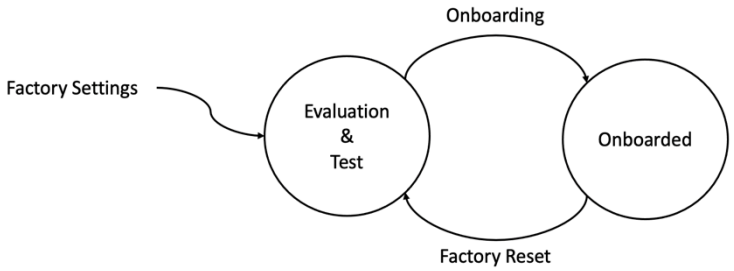


Figure 6 - ExpressLink onboarding states diagram

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.

## 11.3.3 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.

# 12 Testing and qualification

## 12.1 Test hardware

The ExpressLink testing and qualification process requires a Raspberry Pi 4 single board computer and a hardware harness providing the following connections:

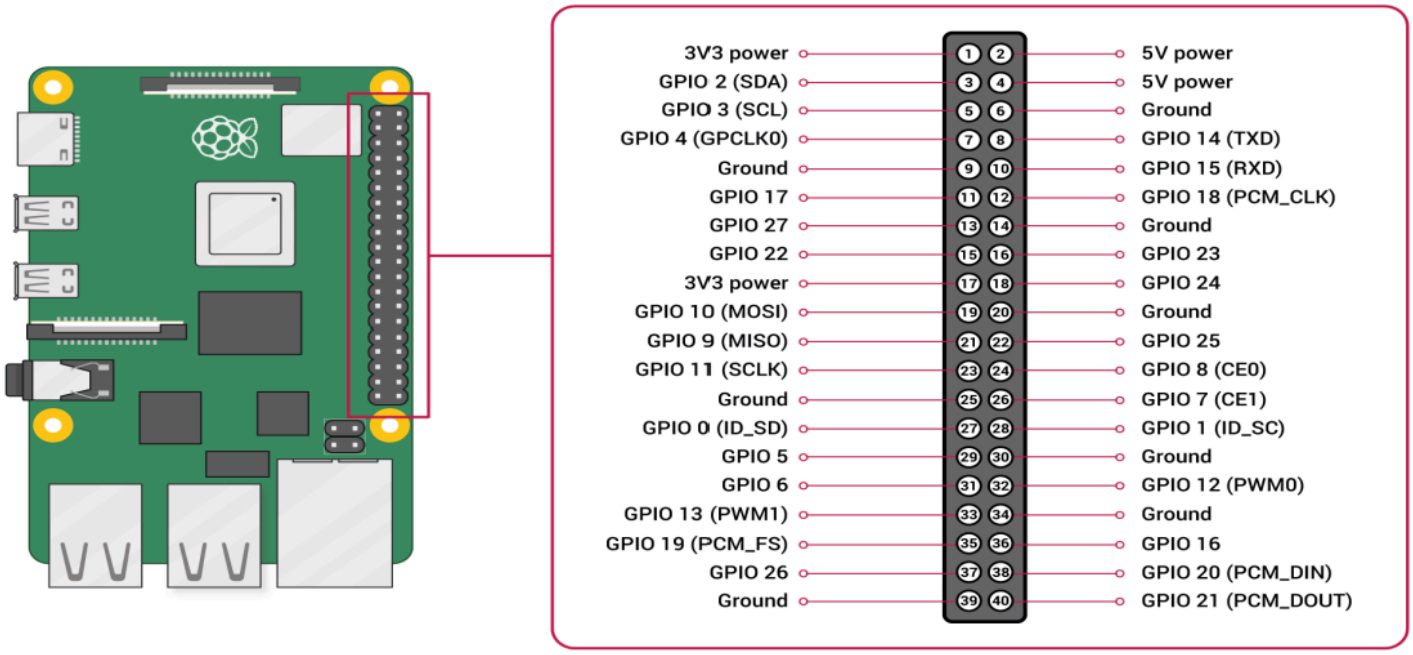


Figure 7 - RaspberryPi 4 I/O connections

The ExpressLink candidate will be attached to the GPIO header of the PI using the following pins.

**Table 7 - Test I/O pin assignments**

| ExpressLink function | GPIO Pin Number |
|---|---|
| Command TX (Transmit from ExpressLink to Host) | GPIO 15 |
| Command RX (Receive from Host to ExpressLink) | GPIO 14 |
| Reset | GPIO 23 |
| Event | GPIO 27 |
| Wake | GPIO 22 |

## 12.2 Test software

A test suite is provided by the AWS FreeRTOS service team to perform the ExpressLink Qualification, a program managed by the AWS Partner Network team.

# 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 1.0 | Initial release. | June 20, 2022 |
| version 1.1.1 | The following sections were added:<br>• 9 AWS IoT Services<br><br>The following sections and tables were updated: | November 17, 2022 |

| Change | Description | Date |
| --- | --- | --- |
| | • 4.3 Delimiters and escaping<br>• 4.4 Maximum values<br>• Table 1 - Error codes<br>• 4.7.2 CONNECT »Establish a connection to an AWS IoT Core Endpoint«<br>• 4.7.7 FACTORY_RESET »Request a factory reset of the ExpressLink module«<br>• 5.1.7 SUBSCRIBE*[#]* »Subscribe to Topic#«<br>• 5.1.8 UNSUBSCRIBE*[#]* »Unsubscribe from Topic#«<br>• Table 2 - Configuration Dictionary Persistent Keys<br>• Table 3 - Configuration dictionary non-persistent keys<br>• Table 4 - ExpressLink event codes<br>• 11 Provisioning<br><br>The following sections were removed:<br>• 4.1.3 SEND *{topic} message* »Publish msg on the specified topic«<br>• 6 Status dictionary<br>note that subsequent sections were renumbered<br>• 9 Additional services | |

| Change | Description | Date |
| --- | --- | --- |
| version 1.1.2 | The following sections and tables were updated:<br>• Table 1 - Error codes<br>  • Introduction of ERROR 26 INVALID CERTIFICATE.<br>• Table 2 - Configuration Dictionary Persistent Keys<br>  • 'Version' now allows for a suffix (for example, 'X.Y.Z beta2').<br>  • OTACertificate is now write-only.<br>• 8 ExpressLink module updates (formerly Chapter 9)<br>  • 8.5 Module OTA signature verification<br>  • Underlined sentence requiring OTA Certificate pre-provisioning.<br>  • Reworded last sentence for clarity.<br>  • 8.6 Module OTA certificate updates<br>  • OTACertificate is now write-only.<br>  • ERR26 is produced when a corrupted or otherwise invalid certificate is presented.<br>  • 8.10 Host OTA Signature Verification<br>  • 8.11 Host OTA certificate update<br>  • 8.11.2 CONF? *{certificate} pem* »Special certificate output formatting option«<br>  • Special 'pem' qualifier is now case insensitive (for example, 'pem' \| 'PEM').<br>  • 8.11.3 CONF {certificate}=pem »Special certificate input formatting option«<br>  • Special 'pem' qualifier is now case insensitive (for example, 'pem' \| 'PEM'). | July 25, 2023 |

# Archive

The following archive versions of this Programmer's Guide are available:

- 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**