Programmer's Guide

ExpressLink

ExpressLink: Programmer's Guide

Table of Contents

AWS IoT ExpressLink programmer's guide v0.5

Topics

AWS IoT ExpressLink commands

Tables

1 Overview

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.

ExpressLink modules were conceived after discussions with microcontroller vendors, OEMs, and module makers. These modules reduce the complexity and repetitiveness of connecting existing and new hardware and software designs to new or different MCUs and RTOSs. This allows a scalable migration for millions of embedded applications to cloud-connected applications.

2 Run states

Although an ExpressLink module operates as a state machine that moves through a number of internal states (see figure 1 for a partial representation), its user interface is designed to abstract these details entirely. This removes all the complexity from the concern of an application or host processor.

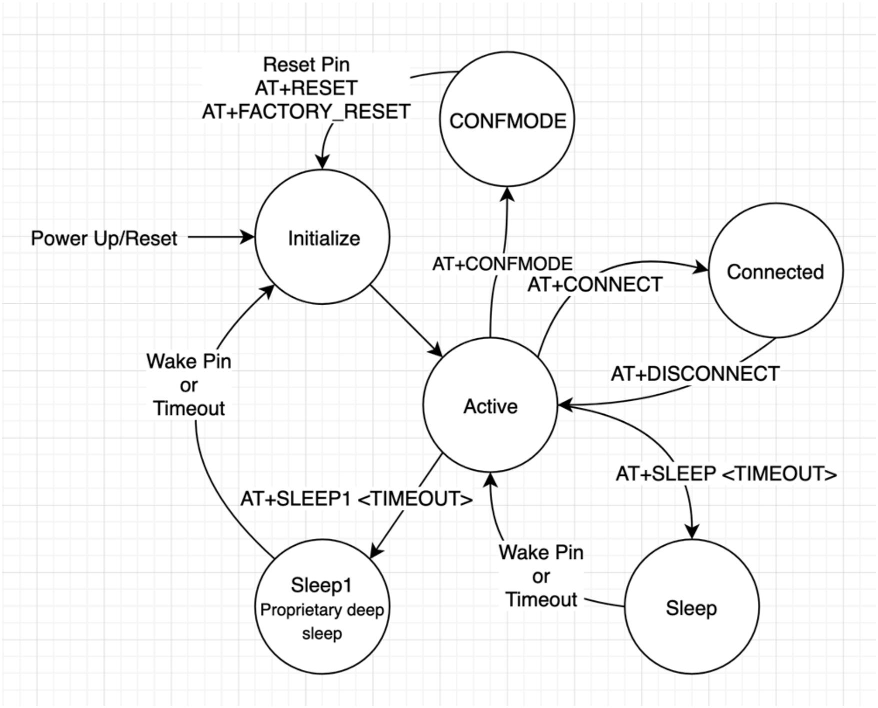

Figure 1 - ExpressLink internal states diagram (partial)

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

The command 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. A host can choose to ignore most event notifications and only periodically poll the receive queue if desired. (See 6.1 Event handling commands.)

3 ExpressLink commands

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. The baud rate is NOT configurable.

3.1 ExpressLink commands format

All ExpressLink commands assume the following general format:

◊ **AT+***{command}[#][?]{separator}[parameter]{eol}*

Where:

3.1.1 *{command}*

A short, alphabetical character string matching one of the commands listed in the following sections (CONNECT, SEND).

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

Returns:

3.1.1.1 `ERR3 COMMAND NOT FOUND`

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

3.1.2 *[#]*

An optional decimal (0..N) suffix qualifier (multiple digits allowed, only used by a few commands).

Returns:

3.1.2.1 `ERR4 PARAMETER ERROR`

If the command does not use a numeric suffix, the module returns 'PARAMETER ERROR'.

3.1.2.2 `ERR4 PARAMETER ERROR`

If the numeric suffix is out of the valid range for the command, the module returns 'PARAMETER ERROR'.

3.1.3 *[?]*

A query marker used with some commands to indicate that information is being requested.

Returns:

3.1.3.1 `ERR22 INVALID QUERY`

If the command does not respond to a query request, then the module returns 'INVALID QUERY'.

3.1.4 *{separator}*

A single ASCII space character (0x20).

Returns:

3.1.4.1 `ERR2 PARSE ERROR`

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

3.1.5 *[parameter]*

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

Returns:

3.1.5.1 `ERR4 PARAMETER ERROR`

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

3.1.6 *{eol}*

A single ASCII newline character (0x0a).

3.1.7 Parameter string note

The parameter string includes all bytes from the separator to the {eol}, not including either the separator or the {eol}. ALL ASCII values from 0 - 0x1F are valid in the parameter string.

3.1.8 Numerical suffix range

The maximum value allowed for the numerical suffix is implementation-specific (typically 0-15).

3.2 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) and backslash ('\').

3.2.1 Escaping {eol} in the parameter string

{eol} (0x0a) in the parameter string is represented by the following sequence: 0x5C 0x41

3.2.2 Escaping backslash ('\') in the parameter string

Backslash (0x5C) in the parameter string is represented by the following sequence: 0x5C 0x5C

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

3.3 Maximum values

3.3.1 Maximum bytes in the formatted command string

The formatted command string as received by ExpressLink can be up to 5K bytes in length.

◊ **AT+**`[ 5K bytes ]`*{eol}*

3.3.2 Maximum command word size

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

3.4 Data processing

3.4.1 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 receiving the {eol}.

3.4.2 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.

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

3.4.3 Data arriving after {eol}

Any data that arrives after {eol} and before 'AT+' will be ignored and discarded.

3.5 Command responses and error codes

All commands respond when the command has been completed. In some cases, this can take a significant amount of time.

3.5.1 General response formats:

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

Where:

- **OK**: Indicates that the command was valid and ran.
- **ERR{#}**: Error codes are listed in Table 1 - Error codes .
- *{separator}*: Is a single ASCII space character (0x20).
- *[detail]*: Is an optional ASCII string.
- *{eol}*: Is a single ASCII newline character (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.

Code	ExpressLink text	Description
6	NO CONNECTION	Command requires an active connection to AWS IoT.
7	TOPIC OUT OF RANGE	The topic index is larger than the maximum valid topic index.
8	TOPIC UNDEFINED	The topic index provided references an empty topic position.
9	INVALID KEY LENGTH	Key is longer than 16 characters.
10	INVALID KEY NAME	A non-alphanumeric character was used in the key name.
11	UNKNOWN KEY	The supplied key cannot be found in the system.
12	KEY READONLY	The key cannot be written.
13	KEY WRITEONLY	The key cannot be read.
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.

Code	ExpressLink text	Description
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	INVALID QUERY	The command does provide a query option.
23	INVALID SIGNATURE	A signature verification failed.

3.5.2 Response timeout

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

3.5.3 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'.

3.6 Power and connection control

3.6.1 CONNECT?: Request the connection status

Requests the current status of the module's connection to the AWS cloud.

Returns:

`OK` *`{status}`*

3.6.1.1 `OK 1`

If the connection is active, then the module returns 'OK 1'.

3.6.1.2 `OK 0`

If the connection is inactive, then the module returns 'OK 0'.

3.6.2 CONNECT: Explicitly request a module to connect to AWS 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.

Returns:

3.6.2.1 `OK 1 CONNECTED`

The module has successfully connected to AWS IoT Core.

3.6.2.2 `ERR14 UNABLE TO CONNECT` *`[detail]`*

The module is unable to connect. Additional clues can be provided by the optional *[detail]* provided (for example, 'Invalid Endpoint').

If the ExpressLink module is already connected, issuing a CONNECT command does not return an error, but simply returns immediately with a success value ('OK CONNECTED').

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 retry request from the host arrives too close to the previous attempt (the interval between requests is shorter than the prescribed minimum backoff time), the ExpressLink module automatically delays running the command. Delays will increase according to the backoff algorithm until a successful connection is established.

Example:

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

Or

```
ERR14 UNABLE TO CONNECT Invalid Endpoint
```

3.6.3 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:

3.6.3.1 `OK 0 DISCONNECTED`

If the module is already disconnected, the command does not return an error, and instead returns immediately with a success value ('OK').

3.6.4 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 duration parameter, if present, indicates the number of seconds before the module awakes automatically. If absent, the module remains in low power mode until a Reset event is generated externally, or a new AT CONNECT command is received.

A SLEEP command without a numerical suffix defaults to mode 0. This is a basic low power mode where the ExpressLink module disconnects and reduces its power consumption as low as possible. At the same time, it maintains the serial interface active, and preserves the contents of all configuration parameters and the status dictionary.

Be aware that *advanced low power modes* can deactivate 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 signal.

Similarly, a deep sleep state might imply loss of all volatile (RAM) information, including all module state information and 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:

3.6.4.1 `OK [{detail}]`

The device is ready and will immediately proceed to the lower power mode selected.

3.6.4.2 `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.

Example 1:

```
AT+SLEEP 100  # Disconnect and suspend all activities for 100 seconds
OK            # Drop connections and goes to sleep
AT+CONNECT    # Resume connection and all pending activities
```

Example 2:

```
AT+SLEEP9     # Goes to deep sleep (proprietary mode) indefinitely
OK
```

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

3.6.5 CONFMODE *[parameter]*: Activate modal credential entry

ExpressLink modules that require additional user credentials can be set by the host to enter CONFMODE (see Figure 1) to enable or repurpose an interface to receive additional connection credentials from user input.

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:

3.6.5.1 `OK CONFMODE ENABLED`

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

3.6.5.2 `ERR17 MODE NOT AVAILABLE`

This ExpressLink model/version does not support CONFMODE.

3.6.5.3 `ERR18 CURRENT CONNECTION`

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

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

The host may issue a RESET command at any time to shut down CONFMODE (see Figure 1).

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.

3.6.6 RESET: Request a 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:

3.6.6.1 `OK`

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

3.6.7 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), but also selected persistent parameters as indicated in Table 2 - Configuration Dictionary Persistent Keys (see the Factory Reset column).

Returns:

3.6.7.1 OK

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

4 Messaging

4.1 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.

4.1.1.1

Topic Index 0 is reserved as a catch-all for messages that do not match other existing topics (the list of topics must not contain an entry for Topic0).

4.1.1.2

Topic *Index{MaxTopic}* is an implementation-dependent value ≥ 16.

4.1.2 Topic usage rules

ExpressLink uses the following rules for creating the topics used to publish and subscribe.

4.1.2.1 Default *TopicRoot*

The default topic root is used to prefix topics that are used in SEND/GET and SUBSCRIBE commands. The *TopicRoot* is meant to simplify the work the host must do to assemble a path that contains its UID/ThingName (whose default value is as shown in Table 2 - Configuration Dictionary Persistent Keys).

4.1.2.2 Topic Strings prefixed with '/' are complete

Topic strings that are prefixed with '/' are considered complete. The topic root will not be prepended to the topic name. The leading '/' will be stripped before using the topic to publish or subscribe.

4.1.2.3 Publish Data Topics are *<TopicRoot>/<Topic@Index>*

Topic names used for publishing are created by combining the TopicRoot (set in the CONF dictionary) with the values at the indexed position in the topic table.

4.1.2.4 Receive Data Topics are *<TopicRoot>/<Topic@Index>*

Topic names used for subscriptions are created by combining the *TopicRoot* (set in the CONF dictionary) with the value at the indexed position in the topic.

4.1.3 SEND *{topic} message*: Publish msg on the specified topic

Where:

{topic}

A string formatted according to topic rules.

message

The message to publish (string).

Example 1:

```
AT+SEND data Hello World    # Publish the classic 'Hello World' message on topic 'data'
OK                          # Message sent
```

4.1.4 SEND*{#} message*: Publish msg on a topic selected from topic list

Where:

{#}

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

message

The message to publish (string).

Example 2:

```
AT+SEND2 Hello World    # Publish 'Hello World' on Topic2
OK                      # Message Sent
```

Returns:

4.1.4.1 OK

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

4.1.4.2 ERR6 NO CONNECTION

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

Example 3:

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

4.1.4.3 ERR7 TOPIC OUT OF RANGE

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

Example 4:

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

4.1.4.4 ERR8 TOPIC UNDEFINED

If the supplied topic index points to a topic entry that has not been defined, the module returns 'TOPIC NOT DEFINED'.

Example 5:

```
AT+SEND3 Hello World    # Publish Hello World on Topic3
ERR8 TOPIC UNDEFINED    # The key Topic3 was not found in the config dictionary
```

4.1.5 GET: Request next message pending on any topic

Retrieve the next message received in the order of arrival.

Returns:

4.1.5.1 OK*{separator}<Topic>{separator}<MESSAGE>{eol}*

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

Example 1:

```
AT+GET                 # poll for messages received on any topic
OK data Hello World    # a message was received from topic 'data'
```

4.1.5.2 OK*{eol}*

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

4.1.6 GET0: Request next message pending on an unassigned topic

Retrieve the next message received on a topic that was not in the topic list.

Returns:

4.1.6.1 OK*{separator}<Topic>{separator}<MESSAGE>{eol}*

Example 2:

```
AT+GET0                # poll for messages received on any unassigned topic
OK data Hello World    # a message was received from topic 'data'
```

4.1.6.2 OK*{eol}*

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

4.1.7 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:

4.1.7.1 `OK`*`{separator}{MESSAGE}{eol}`*

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

Example 3:

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

4.1.7.2 `OK`*`{eol}`*

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

4.1.7.3

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

4.1.7.4 `ERR7 TOPIC OUT OF RANGE`

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

4.1.7.5 `ERR8 TOPIC UNDEFINED`

If the requested topic is not defined, then the module returns 'TOPIC UNDEFINED'.

4.1.7.6 Message queue overflow conditions

If the host fails to retrieve a message and does not free up space and the buffer capacity is exceeded, an overrun occurs. The oldest message in the buffer will be lost. The condition will be reported as a new (OVERFLOW) event and added to the event queue. It is then accessible to the host processor by means of the EVENT? command. Also, the overflow condition is reported in the status dictionary and can be verified by using the STAT? Overflow command. If there is overflow, the number of messages-received events in the queue will exceed the actual number of messages that are present.

4.1.8 SUBSCRIBE*{#}*: Subscribe to the indicated topic

The module subscribes to the topic and start 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.

4.1.8.1 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 1:

```
AT+CONF TopicRoot=building1/floor1
AT+CONF Topic1=sensor1/state
AT+SUBSCRIBE1    # The module will subscribe to the topic building1/floor1/sensor1/
state
```

Example 2:

```
AT+CONF Topic2=/sensor1/state
AT+SUBSCRIBE2    # The module will subscribe to the topic sensor1/state
```

Returns:

4.1.8.2 `ERR6 NO CONNECTION`

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

4.1.8.3 `ERR8 TOPIC UNDEFINED`

If the requested topic is not defined, then the module returns 'TOPIC NOT DEFINED'.

4.1.8.4 `ERR7 TOPIC OUT OF RANGE`

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

4.1.9 UNSUBSCRIBE*{#}*: Unsubscribe from Topic#

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

4.1.9.1 Use the topic at the specified index.

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

Example 1:

```
AT+CONF TopicRoot=building1/floor1
AT+CONF Topic1=sensor1/state
AT+SUBSCRIBE1      # The module will subscribe to topic building1/floor1/sensor1/state
...
AT+UNSUBSCRIBE1    # The module will unsubscribe topic building1/floor1/sensor1/state
```

Returns:

4.1.9.2 `ERR6 NO CONNECTION`

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

4.1.9.3 `ERR8 TOPIC UNDEFINED`

If the requested topic is not defined, then the module returns 'TOPIC NOT DEFINED'.

4.1.9.4 `ERR7 TOPIC INDEX OUT OF RANGE`

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

5 Configuration Dictionary

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

5.1 Data values referenced

5.1.1.1 Maximum key length is 16 characters

A key can be from 1 to 16 characters.

5.1.1.2 `ERR9 INVALID KEY LENGTH`

If the command sends a key with more than 16 characters, the ExpressLink module returns 'ERR9 INVALID KEY LENGTH'.

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

A key may only contain alphanumeric characters in any order.

5.1.1.4 `ERR10 INVALID KEY NAME`

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

5.1.1.5 `ERR11 UNKNOWN KEY`

All keys are predefined in the ExpressLink module. If an invalid key is used, then the module returns 'ERR11 UNKNOWN KEY'.

5.2 Data accessed through the CONF command

5.2.1 Assignment using 'CONF *key*=*value*'

Returns:

5.2.1.1 `OK`

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

5.2.1.2 `ERR9 INVALID KEY LENGTH`

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

5.2.1.3 `ERR10 INVALID KEY NAME`

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

5.2.1.4 `ERR11 UNKNOWN KEY`

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

5.2.1.5 `ERR12 KEY READONLY`

Some keys are read-only and cannot be written. If the key is not present in the system, then the module returns 'KEY READONLY'.

Example 1:

```
AT+CONF VERSION=HELLO
ERR12 KEY READONLY
```

5.2.2 Retrieval using 'CONF? *key*'

Returns:

5.2.2.1 `OK`

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

5.2.2.2 `ERR9 INVALID KEY LENGTH`

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

5.2.2.3 `ERR10 INVALID KEY NAME`

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

5.2.2.4 `ERR11 UNKNOWN KEY`

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

5.2.2.5 `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 2:

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

Key-value pairs can be used to set default values for command parameters, to set credentials, and to select connectivity options and timing preferences. These configuration key-value pairs are meant to be long-lived (persist) throughout the life of the application and are therefore stored in non-volatile memory. A basic set of key-value pairs is defined for all ExpressLink devices (EndPoint, Certificate). Some of these key-value pairs have factory presets and/or are read only.

Table 2 - Configuration Dictionary Persistent Keys

Configuration Parameter	Type	Persist	Initial Value	Factory Reset	Description
About	R	Y	Vendor - Model	N	A descriptive string that uniquely identifies the device make and model - communication capabilities.
Version	R	Y	Current module firmware version	N	The specific ExpressLink firmware version.
TechSpec	R	Y	Technical Specification Compatibility	N	The Technical Specification version this model is based upon (for example 'v0.6').
ThingName	R	Y	UID	N	The UID as natively provided by the module HW root of trust.

Configuration Parameter	Type	Persist	Initial Value	Factory Reset	Description
Certificate	R	Y	Device Birth Certificate	N	Device certificate used to authenticate with AWS cloud, signed by the manufacturer CA.
EndPoint	R/W	Y	Staging account endpoint	Y	The endpoint of the AWS account to which the ExpressLink module connects.
TopicRoot	R/W	Y	UID	Y	The default prefix that is used for all user defined topics.
Defender	R/W	Y	0	Y	The Device Defender upload interval in seconds. (0 indicates the service is disabled.)

Configuration Parameter	Type	Persist	Initial Value	Factory Reset	Description
HOTAcertificate	R/W	Y	{empty}	Y	Host OTA certificate (see chapter 9.10).
OTAcertificate	R/W	Y	Vendor OTA Certificate	N	Module OTA certificate. Vendor and Model specific (see chapter 9.5). (Wi-Fi modules only.)
SSID	R/W	Y	{Empty}	Y	SSID of local router (Wi-Fi modules only).
Passphrase	W	Y	{Empty}	Y	Passphrase of local router (Wi-Fi modules only).
APN	R/W	Y	{default}	Y	Cellular modules only.

Additional configuration parameters are *non-persistent*, and they are re-initialized at power up, or following any reset event. Among these are the topics list items. The host processor has to re-initialize them following any reset, and possibly a deep-sleep awakening (depending on the implementation).

Table 3 - Configuration Dictionary Non-persistent Keys

Configuration Parameter	Type	Persist	Initial Value	Description
IPv4Address	R	N	0.0.0.0	Current device IPv4 address
IPv6Address	R	N	::	Current device IPv6 address
DNSAddress	R	N	0.0.0.0	Current DNS address (IPv4 or IPv6)
GatewayAddress	R	N	0.0.0.0	Current router IP address (IPv4 or IPv6)
Topic1	R/W	N	{Empty}	Custom defined topic 1
Topic2	R/W	N	{Empty}	Custom defined topic 2
…				
Topic<Max Topic>	R/W	N	{Empty}	Custom defined topic MaxTopic

5.3 Configuration commands

5.3.1 CONF *key=value*: Assign a value to selected configuration parameter

Assigns a specific value to the specified configuration parameter.

Returns:

OK

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

ERR# *{message}*

If the command was not successful, the module returns an error.

Example:

```
AT+CONF SSID=MY_SSID    # Assign the preferred (local) Wi-Fi router SSID
```

5.3.2 CONF? *key*: Read value of selected configuration parameter

Query the value of a configuration parameter.

Returns:

OK *{value}*

If the command was successful, the module returns 'OK' followed by the value.

ERR# *{message}*

If the command was not successful, the module returns an error.

Example:

```
AT+CONF? MyParameter
```

6 Event handling

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 can be polled periodically by the host processor using the EVENT? command, or if connected, following an interrupt that is the result of the module activating the EVENT pin. The

EVENT pin is automatically deactivated as soon as the host processor has emptied the queue of pending events.

If the event queue contains one or more events, the value returned by an EVENT? command is the last event that occurred. The event queue depth is implementation dependent; a queue length of 1 means that only the last event (or the highest priority error) is reported in case of overrun.

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

6.1 Event handling commands

6.1.1 EVENT?: Request the next event in the queue

Returns:

6.1.1.1 OK *[{event_identifier} {parameter} {mnemonic [detail] }]{{eol}*

6.1.1.2 **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 topic #.
2	0	STARTUP	The module has entered the active state.
3	0	CONLOST	Connection lost.

Event Identifier	Parameter	Mnemonic	Description
4	0	OVERRUN	Receive buffer Overrun (topic in detail).
5	0	OTA	OTA event (see OTA? for detail).
6	0	SHADOW	SHADOW Event.
7	0	CONFMODE	CONFMODE exit with success.
≤ 999	-		RESERVED.
≥1000	-		Available for custom implementation.

6.2 Diagnostic commands

6.2.1 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.

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
 of
...
OK {SCANRES} -32db SSID1\n-48db SSID2\n ... -90dB SSIDx
```

In the results, the list of SSIDs is provided in the response detail as a multiline string with the newline characters escaped ('\n').

7 ExpressLink module OTA updates

ExpressLink modules natively support over the air (OTA) firmware updates utilizing the AWS IoT OTA service (as currently implemented in the AWS Embedded C-SDK v.202103.00). 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. Consult the manufacturer's datasheet to verify the amount of memory available on a specific model.

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 7.2 OTA commands.)

7.1 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:

7.1.1.1 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.

ExpressLink module	Host Processor
Returns an OTA pending code (1) and metadata about the proposed update (for example, "v2.5.7").	
	OTA ACCEPT or OTA FLUSH Accepts or rejects the proposed update based on metadata provided.
Starts Receiving an OTA payload.	The host processor can shut down the process at any time by issuing the OTA FLUSH command.
When download completed, generates an event (and raises the EVENT Pin).	
	EVENT? polls the event queue.
Returns: OK - OTA Indicates 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 shut down).
If HOTA ready…	Retrieve the payload in chunks of appropriate size.
	READ 1024 Requests first chunk of payload data.

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

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

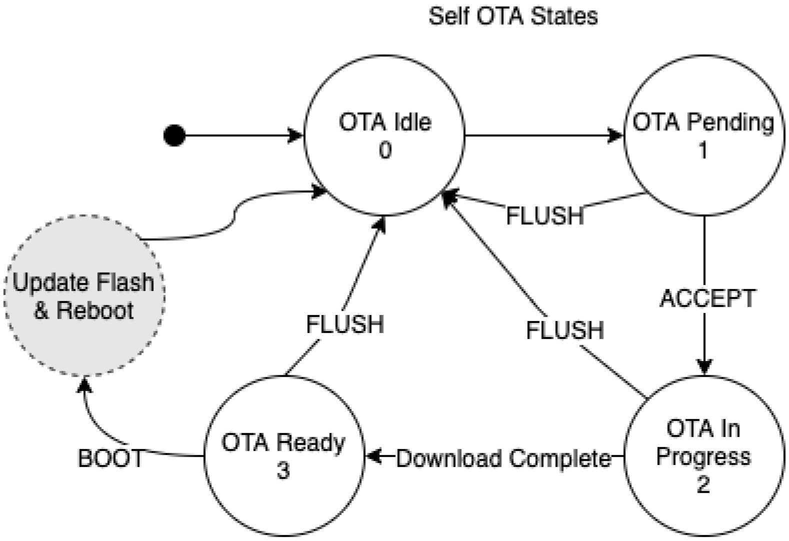

Figure 2 - ExpressLink module OTA state diagram

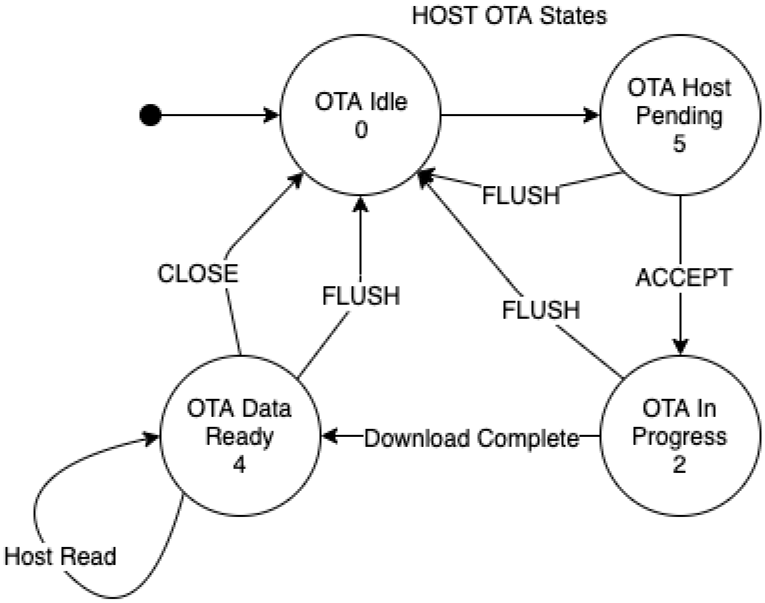

Figure 3 - ExpressLink Host OTA state diagram

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

7.2 OTA commands

7.2.1 OTA?: Fetches the current state of the OTA process

Returns:

```
OK {code} {detail}
```

7.2.2 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.

7.2.3 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:

OK

The OTA operation is allowed to commence.

Example:

```
AT+OTA ACCEPT    # accept the OTA download
OK 1
```

7.2.4 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:

`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.

7.2.4.1 `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.

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
```

7.2.5 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).

7.2.5.1 `ERR20 INVALID ADDRESS`

If the address provided was out of bounds (> OTA buffer content size), then the module returns 'ERR20'.

7.2.5.2 `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.

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
```

7.2.6 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:

7.2.6.1 OK

The module has initiated a boot sequence.

7.2.6.2 `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.

7.2.6.3 `ERR21 INVALID OTA UPDATE`

The module is unable to apply the new module images.

Upon successful completion of the boot sequence, the ExpressLink module communicates the new status and firmware revision number to the AWS IoT OTA service. An event is generated to inform the host processor of the process completion result. The host processor must assume that all state and configuration parameters of the module are reset in a way similar to a factory reset command.

7.2.7 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:

7.2.7.1 OK

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 later. In that case, an event is generated to inform the host and help diagnose the problem.

7.2.8 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:

7.2.8.1 OK

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.

7.3 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 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. Meta-data 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 - Reserved OTA file type codes (0-255) .

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

fileType	Reserved for	Signed	Certificate	Req. Host Permission
101	Module firmware update	Required	Module OTA	Y
103	Module OTA certificate update	Self[1]	Module OTA	N
107	Server Root certificate update	Self[1]	Server Root	N
202	Host update	Optional	Host OTA	N
204	Host OTA certificate update	Self[1]	Host OTA	N

[1] Certificates are already hashed and signed, no additional signing is required.

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.

7.4 Module OTA signature verification

ExpressLink modules are pre-provisioned with a manufacturer's module certificate that is used to validate the signature of firmware update image.

Module OTA jobs include additional meta-data to clearly identify the module's manufacturer, model, and major and minor versions. This allows the ExpressLink executive to discard incorrect OTA images ahead of time to prevent unnecessary downloads.

7.5 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 accessed (read) by means of the serial API (see the CONF? command). Module OTA certificates may also 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:

7.5.1.1 OK

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

7.5.1.2 ERR23 INVALID SIGNATURE

The module returns 'ERR23' if the new certificate could not be verified.

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

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

Module OTA certificates are NOT deleted upon a factory reset.

7.6 Module OTA override

As described in 7.1.1.1 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 meta-data 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 erase 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.

7.7 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.

7.8 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.

7.9 Host OTA Signature Verification

Meta-data 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, must 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).

7.10 Host OTA certificate update

Host OTA certificates can also be updated using the same OTA mechanism or using the AT command interface. In both cases, the new certificate must be signed with the private key corresponding to the previous valid host OTA certificate. The host OTA certificate can be updated by the module manufacturer (OEM) at the end of the product assembly line using the AT+CONF command, or later using the OTA mechanism by making use of the code indicated in Table 5 - Reserved OTA file type codes (0-255) (Host OTA certificate update).

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.

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

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

7.10.1 CONF? *{certificate} pem*: Special certificate output formatting option

The special qualifier *pem* can be appended to read a certificate configuration dictionary key (Certificate, OTAcertificate, HOTAcertificate) 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.

Example:

```
AT+CONF? HOTAcertificate pem
```

Returns:

7.10.1.1 OK *pem*

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

```
-----END CERTIFICATE-----
```

7.10.2 CONF {certificate key}=pem: Special certificate input formatting option

The special value pem can be used to input a certificate (Certificate, OTAcertificate, HOTAcertificate) as a multi-line string to allow the developer to cut and paste directly 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:

7.10.2.1 OK

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

7.10.2.2 `ERR23 INVALID SIGNATURE`

The module returns 'ERR23' 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.

7.11 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).

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

Server Root certificates are NOT deleted upon a factory reset

8 AWS IoT Services

8.1 Device Defender

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

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.

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 specific events).

The device defender feature is activated by setting the *Defender* configuration parameter to a value greater than 0 in the configuration dictionary (using the AT+CONF command).

The Defender parameter value is used to indicate the number of seconds between successive updates of the Device Defender metrics.

The Defender parameter is non-volatile so that an ExpressLink device resumes sending Device Defender metrics at the rate to which it was configured after any reset (soft command or hardware).

Examples:

```
AT+CONF Defender=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 Defender=60      # Device Defender metrics will be sent every minute
AT+CONF Defender=3600    # Device Defender metrics will be sent every hour
```

8.1.1 DEFENDER: {not implemented in this version}

The DEFENDER command is reserved for use with the AWS IoT Device defender service.

8.2 AWS IoT Device Shadow

8.2.1 SHADOW: {not implemented in this version}

The SHADOW command is reserved for the support of the Device Shadow service.

8.3 AWS IoT JOBS

8.3.1 JOB: {not implemented in this version}

The JOB command is reserved for the support of the AWS IoT Jobs service.

9 Additional services

9.1.1 TIME?: Request current time information

ExpressLink modules must provide time information as available from SNTP, GPS, or cellular network sources. Note that 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:

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

If time information is available and recently obtained.

9.1.1.2 `ERR15 TIME NOT AVAILABLE`

If a recent time fix could not be obtained.

9.1.2 WHERE?: Request location information

ExpressLink modules can *optionally* provide last location information as available from GPS, GNSS, cellular networks, or other triangulation methods. A timestamp is provided to let the host determine the currency of the information. The implementation of this command is optional.

Returns:

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

If location coordinates could be obtained at date/time.

9.1.2.2 `ERR16 LOCATION NOT AVAILABLE`

If a location fix could not be obtained.