User Manual

DA1469x USB CLI

<UM>-<X>-<nnn>

Abstract

This user manual provides setup and information for the CLI interface for the DA1469x.

Do not remove this section break!

Contents

[Abstract 1](#_Toc64386079)

[1 Terms and Definitions 6](#_Toc64386080)

[2 References 6](#_Toc64386081)

[3 Introduction 7](#_Toc64386082)

[4 Setup 7](#_Toc64386083)

[4.1 Prerequisites 7](#_Toc64386084)

[4.2 Building and Downloading the firmware 7](#_Toc64386085)

[5 Command Line API 7](#_Toc64386086)

[5.1 ECHO 8](#_Toc64386087)

[5.1.1 CLI Command 8](#_Toc64386088)

[5.1.2 CLI Response 8](#_Toc64386089)

[5.1.3 Example Usage 8](#_Toc64386090)

[5.2 SCAN 8](#_Toc64386091)

[5.2.1 CLI Command 8](#_Toc64386092)

[5.2.2 CLI Response 9](#_Toc64386093)

[5.2.3 Expected Event 9](#_Toc64386094)

[5.2.4 Example Usage 9](#_Toc64386095)

[5.3 CONNECT 9](#_Toc64386096)

[5.3.1 CLI Command 9](#_Toc64386097)

[5.3.2 CLI Response 10](#_Toc64386098)

[5.3.3 Expected Event 10](#_Toc64386099)

[5.3.4 Example Usage 10](#_Toc64386100)

[5.4 DISCONNECT 11](#_Toc64386101)

[5.4.1 CLI Command 11](#_Toc64386102)

[5.4.2 CLI Response 11](#_Toc64386103)

[5.4.3 Expected Events 11](#_Toc64386104)

[5.4.4 Example Usage 11](#_Toc64386105)

[5.5 Cancel Connect 12](#_Toc64386106)

[5.5.1 CLI Command 12](#_Toc64386107)

[5.5.2 CLI Response 12](#_Toc64386108)

[5.5.3 Expected Events 12](#_Toc64386109)

[5.5.4 Example Usage 12](#_Toc64386110)

[5.6 Set Connection Parameters 12](#_Toc64386111)

[5.6.1 CLI Command 12](#_Toc64386112)

[5.6.2 CLI Response 13](#_Toc64386113)

[5.6.3 Expected Events 13](#_Toc64386114)

[5.6.4 Example Usage 13](#_Toc64386115)

[5.7 BROWSE DATABASE 13](#_Toc64386116)

[5.7.1 CLI Command 13](#_Toc64386117)

[5.7.2 CLI Response 13](#_Toc64386118)

[5.7.3 Events 14](#_Toc64386119)

[5.7.4 Example Usage 14](#_Toc64386120)

[5.8 GATT WRITE 14](#_Toc64386121)

[5.8.1 CLI Command 14](#_Toc64386122)

[5.8.2 CLI Response 14](#_Toc64386123)

[5.8.3 Events 14](#_Toc64386124)

[5.8.4 Example Usage 14](#_Toc64386125)

[5.9 GATT WRITE NO RESPONSE 15](#_Toc64386126)

[5.10 GATT READ 15](#_Toc64386127)

[5.10.1 CLI Command 15](#_Toc64386128)

[5.10.2 CLI Response 15](#_Toc64386129)

[5.10.3 Expected Events 15](#_Toc64386130)

[5.10.4 Example Usage 15](#_Toc64386131)

[5.11 GAP SET IO CAPABILITIES 15](#_Toc64386132)

[5.11.1 CLI Command 15](#_Toc64386133)

[5.11.2 CLI Response 16](#_Toc64386134)

[5.11.3 Example Usage 16](#_Toc64386135)

[5.12 GAP PAIR 16](#_Toc64386136)

[5.12.1 CLI Command 16](#_Toc64386137)

[5.12.2 CLI Response 16](#_Toc64386138)

[5.12.3 Expected Events 16](#_Toc64386139)

[5.12.4 Example Usage 17](#_Toc64386140)

[5.13 PASSKEY ENTRY 17](#_Toc64386141)

[5.13.1 CLI Command 17](#_Toc64386142)

[5.13.2 CLI Response 17](#_Toc64386143)

[5.13.3 Expected Events 17](#_Toc64386144)

[5.13.4 Example Usage 17](#_Toc64386145)

[5.14 PASSKEY YES OR NO ENTRY 18](#_Toc64386146)

[5.14.1 CLI Command 18](#_Toc64386147)

[5.14.2 CLI Response 18](#_Toc64386148)

[5.14.3 Expected Events 18](#_Toc64386149)

[5.14.4 Example Usage 18](#_Toc64386150)

[5.15 GET BONDED DEVICES 18](#_Toc64386151)

[5.15.1 CLI Command 18](#_Toc64386152)

[5.15.2 CLI Response 19](#_Toc64386153)

[5.15.3 Expected Events 19](#_Toc64386154)

[5.15.4 Example Usage 19](#_Toc64386155)

[5.16 CLEAR BONDED DEVICE 19](#_Toc64386156)

[5.16.1 CLI Command 19](#_Toc64386157)

[5.16.2 CLI Response 19](#_Toc64386158)

[5.16.3 Example Usage 19](#_Toc64386159)

[5.17 SUOTA IMAGE BLOCK UPDATE 19](#_Toc64386160)

[5.17.1 CLI Command 20](#_Toc64386161)

[5.17.2 CLI Response 20](#_Toc64386162)

[5.17.3 Example Usage 20](#_Toc64386163)

[5.18 SUOTA IMAGE ERASE ALL 20](#_Toc64386164)

[5.18.1 CLI Command 21](#_Toc64386165)

[5.18.2 CLI Response 21](#_Toc64386166)

[5.19 SUOTA IMAGE CHECK 21](#_Toc64386167)

[5.19.1 CLI Command 21](#_Toc64386168)

[5.19.2 CLI Response 21](#_Toc64386169)

[5.19.3 Expected Event 21](#_Toc64386170)

[5.20 SUOTA START IMAGE UPDATE TO PERIPHERAL 21](#_Toc64386171)

[5.20.1 CLI Command 22](#_Toc64386172)

[5.20.2 CLI Response 22](#_Toc64386173)

[5.20.3 Expected Events 22](#_Toc64386174)

[5.20.4 Example Usage 22](#_Toc64386175)

[5.21 RESET 22](#_Toc64386176)

[5.21.1 CLI Command 22](#_Toc64386177)

[5.21.2 CLI Response 22](#_Toc64386178)

[5.22 SET BAUD RATE 23](#_Toc64386179)

[5.22.1 CLI Command 23](#_Toc64386180)

[5.22.2 CLI Response 23](#_Toc64386181)

[5.22.3 Example Usage 23](#_Toc64386182)

[6 SUOTA Python Code Example 23](#_Toc64386183)

[6.1 Downloading an Image to the DA1469x for later SUOTA Update 24](#_Toc64386184)

[6.2 SUOTA Connecting Browsing and Starting SUOTA 26](#_Toc64386185)

[7 EVENTS 28](#_Toc64386186)

[7.1 ADVERTISING 28](#_Toc64386187)

[7.1.1 ADVERTISING EVENT 28](#_Toc64386188)

[7.2 CONNECT | DISCONNECT 29](#_Toc64386189)

[7.2.1 CONNECTION EVT 29](#_Toc64386190)

[7.2.2 DISCONNECT EVT 30](#_Toc64386191)

[7.3 CONNECTION PARAMETERS EVENTS 30](#_Toc64386192)

[7.3.1 PARAMS UPDATED 30](#_Toc64386193)

[7.3.2 PARAMS UPDATE REQUST 30](#_Toc64386194)

[7.4 SERVICE DISCOVERY EVENT 31](#_Toc64386195)

[7.4.1 SERVICE FOUND EVT 31](#_Toc64386196)

[7.4.2 BROWSE COMPLETED EVENT 32](#_Toc64386197)

[7.5 ATTRIBUTE EVENTS 32](#_Toc64386198)

[7.5.1 ATTRIBUTE ERROR EVT 33](#_Toc64386199)

[7.5.2 READ EVT 33](#_Toc64386200)

[7.5.3 NOTIFICATION | INDICATION EVENT 33](#_Toc64386201)

[7.6 SECURITY EVENTS 33](#_Toc64386202)

[7.6.1 SECURITY LEVEL CHANGED 33](#_Toc64386203)

[7.6.2 PASSKEY NOTIFY EVENT 34](#_Toc64386204)

[7.6.3 NUMERIC REQUEST EVENT 34](#_Toc64386205)

[7.6.4 PASSKEY REQUEST EVENT 34](#_Toc64386206)

[7.6.5 PAIR REQUEST EVENT 34](#_Toc64386207)

[7.6.6 PAIR COMPLETED 35](#_Toc64386208)

[7.6.7 GET BONDS EVENT 35](#_Toc64386209)

[7.6.8 BLE ADDRESS RESOLVED 36](#_Toc64386210)

[7.7 DATA LENGTH AND MTU EVENTS 36](#_Toc64386211)

[7.7.1 DATA LENGTH CHANGED EVENT 36](#_Toc64386212)

[7.7.2 MTU CHANGED EVENT 36](#_Toc64386213)

[7.8 PEER INFORMATION EVENTS 37](#_Toc64386214)

[7.8.1 PEER VERSION EVENT 37](#_Toc64386215)

[7.8.2 GET PEER FEATURES EVENT 37](#_Toc64386216)

[7.9 SUOTA EVENTS 37](#_Toc64386217)

[7.9.1 SUOTA Ready Event 37](#_Toc64386218)

[7.9.2 SUOTA NOTIF EVT 37](#_Toc64386219)

[7.9.3 SUOTA TRANSFER COMPLETE 38](#_Toc64386220)

[7.9.4 SUOTA SENT PATCH DATA EVENT 38](#_Toc64386221)

[7.9.5 SUOTA VERSION EVENT 38](#_Toc64386222)

[7.9.6 SUOTA L2CAP PSM Event 38](#_Toc64386223)

[8 LIMITATIONS 38](#_Toc64386224)

[Appendix A <Appendix Title> 41](#_Toc64386225)

[A.1 <Appendix Subsection> 41](#_Toc64386226)

[Revision History 42](#_Toc64386227)

Figures

Optional, remove if not required.

[Figure 1: <Figure Title> 4](#_Toc10722600)

Tables

Optional, remove if not required.

[Table 1: <Table Title> 4](#_Toc10722601)

# Terms and Definitions

Optional section, may be replaced by a Glossary at the end of the document. Define acronyms and abbreviations used in this document. Use sentence case for acronym definitions.

<TERM> <Definition>

<TERM> <Definition>

Etc.

# References

List the documents that are relevant for readers of this document. Avoid hyperlinked references.

1. <PART\_NUMBER>, Datasheet, Revision <Rev>, Dialog Semiconductor.
2. <AN-X-XXX>, <Title>, Application Note, Revision <Rev>, Dialog Semiconductor.
3. UM-B-090, Getting Started User Manual,Revision Rev. 2.0, Dialog Semiconductor.
4. <Author>, <Title>, <Publication>, <Publisher>, Revision <Rev>, <YYYY>.
5. Etc.

# Introduction

This section introduces the subject or problem described in this document.

This CLI implementation is intended to provide extended functionality through a UART interface into devices that support SDK10. The commands provide rich and flexible arguments that provide a full functional central implementation. This firmware is intended to provide the user with a means for rapid prototyping, bench-top testing, or continuous integration scheme.

# Setup

This section and any following sections contain the body of the document. Subsections can be added as required. Quick Parts are available for numbered equations, landscape pages, notices and tables.

This firmware can be used with any Dialog development kits but was intended to be used with the USB kits to provide a compact deployment via PC. If another kit is used, be sure to check the UART pin configurations in the ble\_central\_cli project and adjust accordingly.

## Prerequisites

If the user does not have the following tools already installed, please visit Dialog's support website:

* SDK 10.x.x
* Dialog Smarts Snippets Studio.

Once the SDK exists locally on the PC, the project can be copied into the demos folder of the SDK: projecs/dk\_apps/features/.

## Building and Downloading the firmware

If this is the first time the user is setting up an SDK10 development environment, please reference the Getting Started User Manual [3], for setting up environment and the tools.

Once the environment is setup, import ble\_central\_cli into the workspace and be sure that the python\_scripts is imported for downloading. Build the project for DA1469x-00-Release\_QSPI and use the python script within the tools configurator in the IDE to run program\_qspi\_jtag.

The user can open an application such as TeraTerm/RealTerm to send and receive CLI commands. The default baud rate is 921600, but the user can check in src/command\_parser.c at the top of the file for BAUD\_RATE to ensure the proper baud rate is used.

Press the reset button on the USB kit and a startup message should appear from the device. A quick CLI test is to simply send 'DLG' and a 'OK' should be returned.

# Command Line API

The Dialog CLI relies on ASCII base text which is then interpreted by the command\_parser.c file and then communicated to the BLE stack. All commands have the same basic structure.

* Prefix: DLG+
* Command
* Arguments: separated by white space ' '.
* Carriage Return as a terminator: '\r'

All commands have a whitespace after the command and the first argument. Not all commands require any arguments and just require a CR to terminate a command. The maximum number of arguments supported is 6.

A generic example of a command would look like the following:

DLG+COMMAND PARAM1 PARAM2 PARAM3 PARAM4 PARAM5 PARAM6

The following sections describe each command in detail.

## ECHO

This command is strictly used for checking status of the device. Simply just send the prefix and get a response back.

### CLI Command

* COMMAND: NONE
* ARGUMENTS: NONE

### CLI Response

Response: OK

Error Responses:

* ERROR - Unspecified system error

### Example Usage

Host: 'DLG'

Response: 'OK'

## SCAN

This command will start or stop a scanning procedure. This uses a default scan window and scan interval which is >95% Duty cycle.

### CLI Command

* COMMAND: GAPSCAN
* ARGUMENTS
  + Parameter 1: Start/Stop Command (Required)
    1. Parameter Type: Boolean
    2. Supported Values:
       1. 1 - Start Scan
       2. 0 - Stop Scan

### CLI Response

Expected Response: 'OK'

Error Responses:

* ERROR - Unspecified Error

### Expected Event

See ADVERTISE event.

### Example Usage

Host: 'DLG+GAPSCAN 1'

Response: 'OK'

Following an OK, asynchronous advertising events will start.

## CONNECT

This command will issue a connection request to a peripheral. This supports optional arguments and will use default values if not provided.

### CLI Command

* COMMAND: GAPCONNECT
* ARGUMENTS
  + **Parameter 1: BLE Peripheral Address (Minimum Required)**
    1. Parameter Type: Hex String
    2. Supported Values:
       1. Requires a 6-byte Mac Address with 5 colon separators.
  + **Parameter 2: Public/Private Address**
    1. Parameter Type: Numerical String
    2. Supported values:
       1. '0' - PUBLIC ADDRESS
       2. '1' - PRIVATE ADDRESS
    3. Default Value:
       1. PUBLIC ADDRESS
  + **Parameter 3: Minimum connection interval**
    1. Parameter type: Numerical String (milliseconds)
    2. Supported Values:
       1. Minimum connection interval as specified from the Bluetooth SIG
    3. Default Value: 10ms
  + **Parameter 4: Maximum Connection Interval**
    1. Parameter Type: Numerical String (milliseconds)
    2. Supported Values:
       1. Must be >= Parameter 3
       2. Must be within specifications from the Bluetooth SIG
    3. Default Value: 12.5ms
  + **Parameter 5: Slave Latency**
    1. Parameter Type: Numerical String
    2. Supported Values:
       1. Must be within specification from Bluetooth SIG
    3. Default Value: 0
  + **Parameter 6: Supervision Timeout**
    1. Parameter Type: Numerical String (ms)
    2. Supported Values:
       1. Must be within specifications of Bluetooth SIG
    3. Default Value: 2000ms

Note: If one of the parameters from Parameter 3-6 is included, then all other parameters must be included.

### CLI Response

Response: OK

Error Responses:

* ERROR - Unspecified system error

### Expected Event

See CONNECTION event.

### Example Usage

Example 1: Connect to a public peripheral using default values:

Host: 'DLG+GAPCONNECT 12:34:56:78:9A:BC'

Response: 'OK'

Example 2: Connect to a private address using default values:

Host: 'DLG+GAPCONNECT 12:34:56:78:9A:BC 1'

Response: 'OK'

Example 3: Connect to a public address using different connection parameters.

Host: 'DLG+GAPCONNECT 12:34:56:78:9A:BC 0 200 200 0 3000'

Response: 'OK'

## DISCONNECT

Disconnect a peripheral device.

### CLI Command

* COMMAND: GAPDISC
* ARGUMENTS
  + **Parameter 1: Connection Index (required)**
    1. Parameter Type: Numerical String
    2. Supported Values:
       1. Connection index values typically 0-8. If one connection, then connection index will be zero.
  + **Parameter 2: Disconnect Reason (Optional)**
    1. Parameter Type: Numerical String
    2. Supported Values:
       1. 0 - BLE\_HCI\_ERROR\_REMOTE\_USER\_TERM\_CON
       2. 1 - BLE\_HCI\_ERROR\_AUTH\_FAILURE
       3. 2 - BLE\_HCI\_ERROR\_REMOTE\_DEV\_TERM\_LOW\_RESOURCES
       4. 3 - BLE\_HCI\_ERROR\_REMOTE\_DEV\_POWER\_OFF
       5. 4 - BLE\_HCI\_ERROR\_UNSUPPORTED\_REMOTE\_FEATURE
       6. 5 - BLE\_HCI\_ERROR\_PAIRING\_WITH\_UNIT\_KEY\_NOT\_SUP
       7. 6 - BLE\_HCI\_ERROR\_UNACCEPTABLE\_CONN\_INT
    3. Default Value: 0 - BLE\_HCI\_ERROR\_REMOTE\_USER\_TERM\_CON

### CLI Response

Response: OK

Error Responses:

* ERROR - Unspecified system error

### Expected Events

See DISCONNECT event.

### Example Usage

Example 1: Disconnect from index 0

Host: 'DLG+GAPDISC 0'

Response: 'OK'

Example 2: Disconnect from index 0, with reason 3

Host: 'DLG+GAPDISC 0 3'

Response: 'OK'

## Cancel Connect

Cancel an ongoing connection request. Typically used in the rare case where a link layer CONNECT\_IND fails to be processed by the peripheral. The host side can start a connection timer after sending the connect request and send a cancel connect in the case the timer expires.

### CLI Command

* COMMAND: GAPCANCELCONNECT
* ARGUMENTS: NONE

### CLI Response

Expected Response: OK

Error Responses:

* ERROR - Unspecified system error

### Expected Events

See CONNECT events.

### Example Usage

Host: 'DLG+GAPCANCELCONNECT'

Response: 'OK'

## Set Connection Parameters

This command will update connection parameters on an existing connection.

### CLI Command

* COMMAND: GAPSETCP
* ARGUMENTS:
  + **Parameter 1: Connection Index (required)**
    1. Parameter Type: Numerical String
    2. Parameter Values:
       1. Connection Index 0 - 8
  + **Parameter 2: Connection Interval Min (required)**
    1. Parameter Type: Numerical String (milliseconds)
    2. Parameter Values:
       1. Within the Bluetooth SIG specification
  + **Parameter 3: Connection Interval Max (required)**
    1. Parameter Type: Numerical String (milliseconds)
    2. Parameter Values:
       1. Within the Bluetooth SIG Specification
  + **Parameter 4: Slave Latency**
    1. Parameter Type: Numerical String
    2. Parameter Values:
       1. Within the Bluetooth SIG Specification
  + **Parameter 5: Supervision Timeout**
    1. Parameter Type: Numerical String (milliseconds)
    2. Parameter Value:
       1. Within Bluetooth SIG specifications

### CLI Response

Expected Response: OK

Error Responses:

* ERROR - Unspecified system error

### Expected Events

See PARAMS UPDATED event.

### Example Usage

Host: 'DLG+GAPSETCP 0 0 200 200 0 3000'

Response 'OK'

## BROWSE DATABASE

This command will start a service discovery through the entire attribute database. When a client configuration characteristic is discovered, the central will write to this value to enable indications/notifications.

### CLI Command

* COMMAND: GATTBROWSE
* ARGUMENTS
  + **Parameter 1: Connection Index (required)**
    1. Parameter Type: Numerical String
    2. Parameter Value: 0-8

### CLI Response

Expected Response: OK

Error Responses:

* ERROR - Unspecified system error

### Events

Events that CAN be triggered:

* SERVICE DISCOVERY EVENTS
* GATT Writes
* GATT Read (if SUOTA supported on the peripheral)

### Example Usage

Host: 'DLG+GATTBROWSE 0'

Response: 'OK'

## GATT WRITE

This command will do a GATT write to a specific attribute.

### CLI Command

* COMMAND: GATTWRITE
* ARGUMENTS
  + **Parameter 1: Data to be Written (required)**
    1. Parameter Type: ASCII Hex String
    2. Parameter Values:
       1. hex values with no separation between values.
  + **Parameter 2: Attribute Handle (required)**
    1. Parameter Type: ASCII Hex String
    2. Parameter Values: 16-bit value
  + **Parameter 3: Connection Index (required)**
    1. Parameter Type: Numerical String
    2. Parameter Values: 0-8

### CLI Response

Expected Response: OK

Error Responses:

* ERROR - Unspecified system error

### Events

See ATT Events.

### Example Usage

Write the hex string 0xFFFF to handle 0x25

Host: 'DLG+GATTWRITE ffff 25 0'

Response: 'OK'

## GATT WRITE NO RESPONSE

COMMAND: GATTWRITENORESP

For everything else, refer to GATT WRITE.

## GATT READ

This command will read the value from the server's database.

### CLI Command

* COMMAND: GATTREAD
* ARGUMENTS
  + **Parameter 1: Connection Index (Required)**
    1. Parameter Type: Numerical String
    2. Parameter Values: 0-8
  + **Parameter 2: Handle (Required)**
    1. Parameter Type: Numerical String
    2. Parameter Values: 1-0xFFFF

### CLI Response

Expected Response: OK

Error Responses:

* ERROR - Unspecified system error

### Expected Events

See GATT READ Events

### Example Usage

Host: 'DLG+GATTREAD 0 2'

Response: 'OK'

## GAP SET IO CAPABILITIES

The CLI firmware defaults with a GAP\_IO\_KEYBOARD\_DISP IO Capability. This command will change the local configuration of the central.

### CLI Command

* COMMAND: GATTREAD
* ARGUMENTS
  + **Parameter 1: IO Capabilities (Required)**
    1. Parameter Type: Numerical String
    2. Parameter Values:
       1. 0 - GAP\_IO\_CAP\_DISP\_ONLY
       2. 1 - GAP\_IO\_CAP\_DISP\_YES\_NO
       3. 2 - GAP\_IO\_CAP\_KEYBOARD\_ONLY
       4. 3 - GAP\_IO\_CAP\_NO\_INPUT\_OUTPUT
       5. 4 - GAP\_IO\_CAP\_KEYBOARD\_DISP

### CLI Response

Expected Response: OK

Error Responses:

* ERROR - Unspecified system error

### Example Usage

Host: 'DLG+GAPSETIOCAP 2'

Response: 'OK'

## GAP PAIR

This command will start the pairing / bonding procedure with a peripheral.

### CLI Command

* COMMAND: GAPPAIR
* ARGUMENTS
  + **Parameter 1: Connection Index (Required)**
    1. Parameter Type: Numerical String
    2. Parameter Values: 0-8
  + **Parameter 2: Bond**
    1. Parameter Type: Boolean String
    2. Parameter Values:
       1. 0 - Do not Bond
       2. 1 - Bond device

### CLI Response

Expected Response: OK

Error Responses:

* ERROR - Unspecified system error

### Expected Events

See PAIR Events.

### Example Usage

Host: 'DLG+GAPPAIR 0 1'

Response: 'OK'

## PASSKEY ENTRY

This command is to be used in conjunction with an asynchronous key prompt which is displayed by the peripheral. Please see PAIR events for passkey event.

### CLI Command

* COMMAND: PASSKEYENTRY
* ARGUMENT
  + **Parameter 1: Connection Index (Required)**
    1. Parameter Type: Numerical String
    2. Parameter Values: 0-8
  + **Parameter 2: Accept Pairing (Required)**
    1. Parameter Value: Boolean String
    2. Parameter Values: 0,1
  + **Parameter 3: Passkey (Required even if rejecting)**
    1. Parameter Type: Numerical String value
    2. Parameter Values: 32-bit value max

### CLI Response

Expected Response: OK

Error Responses:

* ERROR - Unspecified system error

### Expected Events

This will trigger the final stages of the pairing bonding procedure. See PAIR Events for final events.

### Example Usage

Connection Index prompts with: ' BLE\_EVT\_GAP\_PASSKEY\_REQUEST: conn\_idx: 0000'. The peripheral displays a key of 123456.

Host: 'DLG+PASSKEYENTRY 0 1 123456'

Response: 'OK'

## PASSKEY YES OR NO ENTRY

This command is to be entered in conjunction with a secure connection scheme. The peripheral will display a passkey and the CLI will prompt with an asynchronous event of a NUMERIC Request also displaying a matching passkey and the user is to accept or reject the pairing.

### CLI Command

* COMMAND: YESNOENTRY
* ARGUMENTS
  + **Parameter 1: Connection Index (Required)**
    1. Parameter Type: Numerical String
    2. Parameter Values: 0-8
  + **Parameter 2: Accept Pairing**
    1. Parameter Type: Boolean String
    2. Parameter Values: 0, 1

### CLI Response

Expected Response: OK

Error Responses:

* ERROR - Unspecified system error

### Expected Events

See PAIR Events.

### Example Usage

Peripheral Displays passkey of 123456, and CLI prompts user with 'BLE\_EVT\_GAP\_NUMERIC\_REQUEST: conn\_idx: 0000 passkey: 123456'

Host: 'DLG+YESNOENTRY 0 1'

Response: 'OK'

## GET BONDED DEVICES

This command will retrieve all bonded devices that are stored in non-volatile memory and display them back to the user. The central currently supports 8 bonded devices and will replace the oldest bond in the database in the case that a 9th device is bonded.

### CLI Command

* COMMAND: GETBONDS
* ARGUMENTS: NONE

### CLI Response

Expected Response: OK

Error Responses:

* ERROR - Unspecified system error

### Expected Events

See BONDED DEVICES EVENT.

### Example Usage

Host: 'DLG+GETBONDS'

Response: 'OK'

## CLEAR BONDED DEVICE

This command will take an address as a parameter and delete the bond in the database. To clear all bonds, call GETBONDS and then make subsequent calls to this command.

### CLI Command

* COMMAND: CLEARBOND
* ARGUMENTS:
  + **Parameter 1: Bluetooth Address**
    1. Format: 6-byte hex string with a colon as a delimiter (xx:xx:xx:xx:xx:xx)
  + **Parameter 2: Bluetooth Address Type**
    1. Format: Boolean String
    2. Values:
       1. 0 - Public Address
       2. 1 - Private Address

### CLI Response

Expected Response: OK

Error Responses:

■ ERROR - Unspecified system error

### Example Usage

Host: 'DLG+CLEARBOND: 80:ea:ca:a0:06:53 0'

Response: 'OK'

## SUOTA IMAGE BLOCK UPDATE

This command is intended to be used by the host for sending a local SUOTA image to the DA1469x device. The image is stored in the binary partition of NVM and retrieved later when SUOTA is started. The DA1469x has a local buffer of 4096 bytes and the user should be sure not to overrun this buffer by setting the write\_to\_flash bit in the command. Additionally, the CLI has a local buffer where the block data parameter cannot exceed 1024 bytes (before encoding).

### CLI Command

* COMMAND: IMGBLOCKUPDATE
* ARGUMENTS
  + **Parameter 1: Block Data (Required)**
    1. Parameter Type: Base64 encoded Data
    2. Parameter Length Max: 1369 bytes
  + **Parameter 2: Data Length (Required)**
    1. Parameter Type: Numerical String Base10
    2. Parameter Value: 0-1024 bytes (Length before encoding)
  + **Parameter 3: Address to write (Required)**
    1. Parameter Type: Integer String
    2. Parameter Value: 0-0xFFFFFFFFFFFF
  + **Parameter 4: CRC16 (Required)**
    1. Parameter Type: Integer String
    2. Parameter Values: 16-bit CRC (See example for calculation)
  + **Parameter 5: Write to Flash (Required)**
    1. Parameter Type: Boolean String
    2. Parameter Values: 0,1

### CLI Response

Expected Response: OK

Error Responses:

* ERROR - Unspecified system error
* ERROR: FLASH\_WRITE - Error Writing to flash
* ERROR: BUFFER\_FULL - Local Buffer overrun of flash sector buffer (4096 bytes)
* ERROR: CRC - Error in CRC calculation of block data
* ERROR: Invalid Elements - Parameter 2 does not match the data length set in Parameter 1
* ERROR: Sector Pointer - Local sanity check failed for Parameter 3 against current pointer

### Example Usage

Please see SUOTA Example Code.

## SUOTA IMAGE ERASE ALL

This command will erase a local image being used for SUOTA update. Should be used prior to any IMAGE BLOCK UPDATE command.

### CLI Command

* COMMAND: IMGERASEALL
* ARGUMENTS: NONE

### CLI Response

Expected Response: OK

Error Responses:

* ERROR - Unspecified system error

## SUOTA IMAGE CHECK

This command should be sent after the image block update is issued. It should also be sent on any device resets. This command will have the DA1469x do an image integrity check on the local SUOTA image. If not performed, SUOTASTART will not work.

### CLI Command

* COMMAND: IMGCHECK
* ARGUMENTS: NONE

### CLI Response

Expected Response: OK

Error Responses:

* ERROR - Unspecified system error
* ERROR: No Valid Image

### Expected Event

If a valid image exists then the following image information is written:

FW Image information:

\tCode Size: X bytes

\tVersion: x.x

\tTimestamp: (UTC format)

\tCRC: XXXXXXXX

END IMAGE

## SUOTA START IMAGE UPDATE TO PERIPHERAL

This command will start a SUOTA image update. The user should be connected, and a GATTBROWSE should have been issued to identify the SUOTA service on the peripheral.

### CLI Command

* COMMAND: SUOTASTART
* ARGUMENTS
  + **Parameter 1: Force GATT Update (Required)**
    1. Parameter Type: Boolean String
    2. Parameter Values:
       1. 0 - Try L2CAP Update, GATT Otherwise
       2. 1 - Force Image update over GATT

### CLI Response

Expected Response: OK

Error Responses:

* ERROR - Unspecified system error
* ERROR: No valid image detected (IMGCHECK was not run or no image was downloaded)
* ERROR: peer does not support SUOTA or GATTBROWSE was never completed
* ERROR: not connected

### Expected Events

See SUOTA Events.

### Example Usage

Host: 'DLG+ SUOTASTART 0'

Response: 'OK'

Response: 'Updating…'

## RESET

This will do a software reset of the device. This command should be used if a host connects to the device in an unknown state.

### CLI Command

* COMMAND: RESET
* ARGUMENTS: NONE

### CLI Response

If the command is accepted, then a reboot should occur and an initialization message will appear.

Expected Response: APP\_INITIALIZED

Error Responses:

* ERROR - Unspecified system error

## SET BAUD RATE

This will change the baud rate used to communicate from the host to the CLI firmware.

### CLI Command

* COMMAND: SETBAUD
* ARGUMENT
  + **Parameter 1: BAUD RATE (Required)**
    1. Parameter Type: Enum String
    2. Default Value: 10000000
    3. Parameter Values:
       1. 0 - 4800
       2. 1 - 9600
       3. 2 - 14400
       4. 3 - 19200
       5. 4 - 28800
       6. 5 - 38400
       7. 6 - 57600
       8. 7 - 115200
       9. 8 - 230400
       10. 9 - 500000
       11. 10-1000000

### CLI Response

Expected Response: OK

Error Responses:

* ERROR - Unspecified system error

### Example Usage

Host: 'DLG+SETBAUD 9'

Response: 'OK'

# SUOTA Python Code Example

The full SUOTA implementation should fit the following flow: IMGERASEALL->DOWNLOAD\_IMAGE->IMGCHECK->SCAN->CONNECT->BROWSE SERVICES->SUOTASTART

## Downloading an Image to the DA1469x for later SUOTA Update

Method enters at download\_image.

@classmethod  
def verify\_img(cls):  
 log = []  
 image\_size\_str = 'Test'  
 cls.echo = True  
 result = cls.cmdack('AT+IMGCHECK', 'END IMAGE', 1000, mylog=log)  
  
 for item in log:  
 if 'FW Image size:' not in item:  
 continue  
  
 image\_size\_str = item.split('FW Image size:')[1].split('bytes')[0].strip()  
 cls.image\_size = int(image\_size\_str, 10)  
  
 cls.print\_term(f'Image size {cls.image\_size}..\n',  
 color='Blue', weight='Bold', fixed=True)

@classmethod  
def erase\_image(cls):  
 log = []  
  
 tstart = datetime.now()  
 cls.echo = True  
 cls.cmdack('AT+IMGERASEALL', 'OK', 30000, mylog=log)  
 elapsed = datetime.utcfromtimestamp((datetime.now() - tstart).total\_seconds())  
 cls.print\_term(f'Erase time is (mm:ss) {elapsed.strftime("%M:%S.%f")}\n',  
 color='Blue', weight='Bold', fixed=True)

@classmethod  
def download\_image(cls):  
 """Save contents of output window."""  
 filename = None  
  
 chunk\_size = 1024  
 sector\_size = 4096  
 blocks = sector\_size/chunk\_size  
  
 # noinspection PyBroadException  
 try:  
 with wx.FileDialog(cls.\_AppSubscriber\_\_frame, message="Choose SUOATA Image", defaultDir=".",  
 defaultFile="",  
 wildcard="SUOTA IMG File|\*.img",  
 style=wx.FD\_OPEN) as dlg:  
 if dlg.ShowModal() == wx.ID\_OK:  
 filename = dlg.GetPath()  
 print(filename)  
 else:  
 return  
  
 except Exception:  
 exc\_type, exc\_value, exc\_traceback = sys.exc\_info()  
 print(repr(traceback.format\_exception(exc\_type, exc\_value, exc\_traceback)))  
  
 with open(filename, 'rb') as fo:  
 bytes\_read = fo.read()  
  
 file\_len = len(bytes\_read)  
 chunks = math.ceil(file\_len / chunk\_size)  
 chunk\_items = 0  
  
  
 log = []  
 sp = " "  
  
 tstart = datetime.now()  
 total\_time\_sent = 0.0  
  
 cls.print\_term(f'Erasing Image...\n',  
 color='Blue', weight='Bold', fixed=True)  
 cls.erase\_image()  
 cls.echo = False  
  
 for i in range(chunks):  
  
 crc = 0xFFFF  
 write\_to\_flash = 0  
 if i == (chunks - 1):  
 chunk\_items = file\_len % chunk\_size  
 write\_to\_flash = 1  
 else:  
 chunk\_items = chunk\_size  
 if (i % blocks) == blocks-1:  
 write\_to\_flash = 1  
  
 addr = i\*chunk\_size  
  
 mydata = base64.b64encode(bytes\_read[addr:addr+chunk\_items])  
 mydata = mydata.rstrip(b'\n').decode()  
 crc = cls.crc16\_sdk(bytes\_read[addr:addr+chunk\_items], crc)  
  
 command\_string = f'AT+IMGBLOCKUPDATE {mydata} {chunk\_items} {addr} {crc} {write\_to\_flash}'  
  
 cls.clear\_resp(mylog=log)  
 cls.send(command\_string, echo=False)  
 start = datetime.now()  
 result = cls.wait\_for\_resp('OK', timeout=5000, mylog=log)  
 total\_time\_sent += (datetime.now() - start).total\_seconds()  
  
 cls.update\_prog((i/chunks)\*100)  
 if result == -1:  
 cls.print\_term(f'Image Download Fail...\n',  
 color='Red', weight='Bold', fixed=True)  
 return  
  
 elapsed = datetime.utcfromtimestamp((datetime.now() - tstart).total\_seconds())  
  
 cls.print\_term(f'Image Downloaded Successfully...\n',  
 color='Blue', weight='Bold', fixed=True)  
  
 cls.print\_term(f'Elapsed time is (mm:ss) {elapsed.strftime("%M:%S.%f")}\n',  
 color='Blue', weight='Bold', fixed=True)  
  
 cls.print\_term(f'Total Sent Time = {total\_time\_sent}s\n')  
  
 time.sleep(2)  
  
 cls.update\_prog(0)  
  
 cls.verify\_img()

Sd

## SUOTA Connecting Browsing and Starting SUOTA

This code snippet scans, parses the scan and opens a dialogue box to choose a MAC address and start a connection.

@classmethod  
def scan\_and\_connect(cls, tscan=10):  
 log = []  
  
 tstart = datetime.now()  
 cls.echo = False  
 rsp = cls.cmdack('AT+GAPSCAN 1', 'OK', mylog=log)  
 if rsp != -1:  
 cls.print\_term(f'Scanning for {tscan} seconds...\n',  
 color='Blue', weight='Bold', fixed=True)  
 else:  
 cls.print\_term(f'Error...\n',  
 color='Blue', weight='Bold', fixed=True)  
 cls.echo = True  
 return -1  
  
 time.sleep(tscan)  
 cls.send('AT+GAPSCAN 0', mylog=log)  
 rsp = cls.wait\_for\_resp('SCAN COMPLETED', mylog=log)  
 if rsp == -1:  
 cls.print\_term(f'Error...\n',  
 color='Blue', weight='Bold', fixed=True)  
 return -1  
  
 cls.echo = True  
 elapsed = datetime.utcfromtimestamp((datetime.now() - tstart).total\_seconds())  
 cls.print\_term(f'Scan Finished, total time is (mm:ss) {elapsed.strftime("%M:%S.%f")}\n',  
 color='Blue', weight='Bold', fixed=True)  
 # Find unique MAC Addresses  
 mac\_addr = {}  
 count = 0  
 for item in log:  
 if 'ADDR:' not in item:  
 continue  
 count += 1  
 addr = item.split('ADDR:')[1].split(',')[0].strip()  
  
 try:  
 data = item.split('DATA:')[1].split(',')[0].strip()  
 data\_hex = [int(x, 16) for x in data.split()]  
 except IndexError:  
 data = []  
 data\_hex = 0  
 print(f'Exception Captured: {item}\n')  
  
 try:  
 rssi = item.split('RSSI:')[1].split(',')[0].strip()  
 rssi\_val = int(rssi, 10)  
 except IndexError:  
 rssi\_val = 256  
 print(f'Exception Captured: {item}\n')  
  
 if addr in mac\_addr.keys():  
 mac\_addr[addr][0] += 1  
 if rssi\_val != 256:  
 mac\_addr[addr][3] = mac\_addr[addr][3] + rssi\_val  
 if data\_hex:  
 if data\_hex[0] != 2:  
 mac\_addr[addr][2] = data\_hex  
 else:  
 mac\_addr[addr][1] = data\_hex  
 else:  
 if data\_hex:  
 if data\_hex[0] != 2:  
 mac\_addr[addr] = [1, [], data\_hex, 0]  
 else:  
 mac\_addr[addr] = [1, data\_hex, [], 0]  
  
 if rssi\_val !=256:  
 mac\_addr[addr][3] = rssi\_val  
  
 cls.print\_term(f'We scanned {len(mac\_addr)} devices and received {count} events:\n')  
  
 choices = []  
 for item in mac\_addr.keys():  
 name = "Unknown Device"  
 data = mac\_addr[item][1]  
 data += mac\_addr[item][2]  
  
 if data:  
 name = cls.parse\_adv\_data\_name(data)  
  
 rssi\_avg = round(mac\_addr[item][3]/mac\_addr[item][0])  
 choices.append(f'{item} - {name} [{rssi\_avg} dBm]')  
  
 try:  
 choices.sort(key=lambda x: int(x.split('[')[1].split()[0], 10), reverse=True)  
 except ValueError:  
 print(f'Sort Exception Caught\n')  
  
 cls.select\_mac\_addr(choices)  
 while not cls.get\_selection():  
 time.sleep(.1)  
  
 selection = cls.get\_selection().pop().split('-')[0].strip()  
  
 if "cancel" in selection:  
 cls.print\_term(f'Cancel Connect\n')  
 return  
  
 cls.print\_term(f'We Selected {selection} with Data 'f'{mac\_addr[selection][1]} and {mac\_addr[selection][2]}\n')  
  
 command\_string = f'AT+GAPCONNECT {selection} {0}'  
 rsp = cls.cmdack(command\_string, 'CONNECTED', 15000, mylog=log)  
 cls.echo = True  
  
 if rsp == -1:  
 cls.print\_term(f'Connection Unsuccessful\n')  
 cls.cmdack(f'AT+GAPCANCELCONNECT', 'OK', 1000, mylog=log)  
 return  
  
 # need to flow this with connection being completely setup, timeout for now  
 time.sleep(1)  
 log = []  
 rsp = cls.cmdack(f'AT+GATTBROWSE {0}', 'BROWSE COMPLETED', 10000, mylog=log)  
 database = cls.parse\_gatt\_db(log)  
  
 cls.show\_ble\_database(database)

# EVENTS

This section describes asynchronous events generated by the central.

## ADVERTISING

Advertisements will be reported with the advertisement data and the scan response in separate advertisement report. If it is critical to the application to distinguish between both, the GAP Flags should be checked on the first three bytes of data. Please reference {TODO: GAP GLAGS REFERENCE}.

### ADVERTISING EVENT

* FORMAT: 'EVT\_ADV - ADDR: {addr}, ADDR\_TYPE {addr\_type}, TYPE: {adv\_type}, RSSI: {rssi}, DATA: {data}\r\n"
* DATA:
  + addr - Bluetooth Address
    1. Format: xx:xx:xx:xx:xx:xx
  + addr\_type - Public/Private Address
    1. Data Format: Enum String
    2. Data Values:
       1. 0 - Public Address
       2. 1 - Private Addres
  + adv\_type
    1. Data Format: Enum String
    2. Data Values:
       1. 00 - General Advertisement
       2. 01 - Direct Advertisement
       3. 02 - Scannable Advertising Indication
       4. 03 - Non Connectable Advertisement
       5. 04 - Scan Response
  + Rssi - Signal Strength
    1. Data Format: Integer string
  + Data
    1. Data Format: Hex String (no spaces)

## CONNECT | DISCONNECT

Events that are associated with the connection/disconnection process.

### CONNECTION EVT

After a successful connection, the central will provide the appropriate event. Additionally the connection process will check to see if the device is currently bonded, and if it is, will start he encryption process.

#### CONNECTION INDICATION

* FORMAT: 'EVT\_CONNECTED \r\n\t ADDR: {addr}, CONN\_ID: {conn\_id}\r\n'
* DATA
  + **Addr - Peer Bluetooth Address**
    1. Format: xx:xx:xx:xx:xx:xx
  + **Conn\_id - Connection ID**
    1. Format: Numerical String
    2. Values: 0-8

#### BONDED INDICATION

The central will see if the device bonded and also provide additional data from the peer.

* FORMAT: ' EVT\_BONDED: {bonded} MITM: {mitm} SEC: {sec}\r\n'
* DATA:
  + **Bonded - has peer been previously bonded**
    1. **Format - Boolean String**
  + **mitm - Does the peer support MITM security**
    1. Format - Boolean String
  + **Sec - Does the peer support secure connections**
    1. Format - Boolean String

#### SET SECURITY LEVEL INDICATION

If the bonded flag is 1 in the BONDED INDICATION, then the central will set the highest security level possible with the peer device and provide this associated event.

* FORMAT: ' EVT\_BLE\_GAP\_SET\_SEC\_LEVEL: level: {level} err: {err}\r\n'
* DATA:
  + **Level - security level**
    1. Format - Enum String
    2. Values:
       1. 0 - GAP\_SEC\_LEVEL\_1 (No Security)
       2. 1 - GAP\_SEC\_LEVEL\_2 (Unauthenticated with encryption)
       3. 2 - GAP\_SEC\_LEVEL\_3 (Authenticated with encryption)
       4. 3 - GAP\_SEC\_LEVEL\_4 (Authenticated with LE Secure Connections and encryption)
  + **Err - Error Return**
    1. Format - Enum String
    2. Values:
       1. 0 - No error
       2. > 1 - SEE ble\_error\_t in SDK.

#### CONNECTION FAILED

If the connection attempt failed, then this event occurs.

* FORMAT: 'EVT\_CONNECT\_FAILED: {status}\r\n'
* DATA:
  + **Status - error return**
    1. format - Enum String
    2. Values:
       1. See ble\_error\_t in SDK.

### DISCONNECT EVT

On disconnection from peer, this event is triggered.

* FOMRAT: 'EVT\_DISCONNECTED - ADDR: {addr}, CONN\_ID: {conn\_id} REASON: {reason}\r\n'
* DATA
  + **Addr - Peer Bluetooth Address**
    1. Format: xx:xx:xx:xx:xx:xx
  + **Conn\_id - Connection ID**
    1. Format: Numerical String
    2. Values: 0-8
  + **Reason - Reason for disconnection**
    1. See LL Error Codes

## CONNECTION PARAMETERS EVENTS

Events Generated from either updating the connection parameters or when a request is made.

### PARAMS UPDATED

This event occurs whenever the central updates the connection parameters successful.

* FORMAT: 'EVT\_PARAMS\_UPDATE\_COMPLETE\r\n'
* DATA: NONE

### PARAMS UPDATE REQUST

This event occurs whenever a peer requests new connection parameters. The code will automatically accept any request made by the peripheral and display the resulting request.

* FORMAT: 'EVT\_PARAM\_UPDATE\_REQUEST: conn\_idx={conn\_id}\r\n'
  + Additional lines:
    1. '\tInterval Min: {min}\r\n'
    2. '\tInterval Max: {max}\r\n'
    3. '\tSlave Latency: {latency}\r\n'
    4. '\tSupervision TO: {to}\r\n'
* DATA:
  + **Conn\_id - connection ID**
    1. Format - Numerical String
    2. Values - 0-8
  + **Min - Minimum connection interval**
    1. Format: Numerical String
    2. Value Format: Double slots (e.g. 1.25ms\*value)
    3. Value Range: See Bluetooth SIG standards
  + **Max - Maximum connection interval**
    1. Format: Numerical String
    2. Value Format: Double slots (e.g. 1.25ms\*value)
    3. Value Range: See Bluetooth SIG standards
  + **Latency - slave latency** 
    1. Format: Numerical string
    2. Value Range: See Bluetooth SIG Standards
  + **To - Supervision Timeout**
    1. Format: Numerical String
    2. Value Range: See Bluetooth SIG Standards.

## SERVICE DISCOVERY EVENT

This event is triggered from a GATTBROWSE command. A separate event is triggered for each service that is discovered in the database. Another event is triggered when the entire database has been discovered.

### SERVICE FOUND EVT

For each service discovered the entire service is displayed to the user. If Device Information or the SUOTA service is discovered then the DA1469x client will save this information off in case a SUOTA update is ever initiated. SEE SUOTA EVTs for more detail.

If an attribute is found with a UUID of 0x2902 (Client Characteristic Configuration), the central will write to this characteristic to enable either Notifications or Indications.

* FORMAT: 'EVT\_SERVICE\_FOUND: conn\_idx={id} start\_h={s\_handle} end\_h={e\_handle}\r\n'
  + Possible Additional Lines:
    1. Service Declaration
       1. '\t{handle} serv {uuid}\r\n'
    2. Include Service Declaration
       1. '\t{handle} incl {uuid}\r\n

Characteristic Information

* + - 1. '\t{handle} char %s prop={props} ({props\_s})\r\n'
      2. '\t{handle} ---- {char\_uuid}\r\n'
    1. Descriptor Information
       1. '\t%{handle} desc {uuid}\r\n
* DATA
  + **Id - Connection ID**
  + **S\_handle - start handle of the service**
    1. Format: Hex String
    2. Value: 1-0xFFFF
  + **E\_handle - last handle of the service**
    1. Format: Hex String
    2. Value: 1-0xFFFF
  + **handle - handle of the individual attribute**
    1. Format: Hex String
    2. Value: 1-0xFFFF
  + **Uuid - UUID of characteristic or service**
    1. Format: String
    2. Value Format: Follows XML hex string standard from Bluetooth SIG
  + **Props - Properties of characteristic**
    1. Format: Hex String
    2. Value Format: Bitwise property value
    3. Value Range: See Bluetooth SIG Standard
  + **Prop\_s - Properties string**
    1. Format: String
    2. Value Format: Each string location will be toggled to associated property or with '-' if property is not available.
    3. Values:
       1. 'B' - Broadcast
       2. 'R' - Read
       3. 'X' - Write without Response
       4. 'W' - Write
       5. 'N' - Notify
       6. 'I' - Indicate
       7. 'S' - Signed Write
       8. 'E' - Extended Properties

### BROWSE COMPLETED EVENT

Whenever all services has been discovered this event occurs. If SUOTA or Device Information Service was found then the central will start to read necessary characteristics to prepare for a SUOTA update.

* FORMAT: EVT\_BROWSE\_COMPLETED
* DATA: NONE

If SUOTA was found an additional event will occur:

* FORMAT: EVT\_READING\_SUOTA\_DATABASE\_CHARS
* DATA: NONE

## ATTRIBUTE EVENTS

These events occur with relation to reading, writing the database or receiving indications or notifications.

### ATTRIBUTE ERROR EVT

In the case of reading or writing, the associated response will be reported in an event.

* FORMAT: 'ATT\_ERROR\_XXXXX'
* DATA:
  + XXXXX - represents the associated error code
    1. Values: See att\_error\_t within SDK

### READ EVT

Following an ATT\_ERROR\_OK reply from the ATTRIBUTE ERROR EVT, the characteristic value will be reported.

* FORMAT: 'EVT\_GATT\_READ: conn\_idx:{id} handle:{handle} offset:{offset}\r\n\tValue: {value}\r\n'
* DATA:
  + **ID - Connection ID**
    1. Format: Hex String
    2. Value - 0x0000 - 0x0008
  + **Handle - Attribute handle**
    1. Format: 2 byte hex string
  + **Offset - value offset**
    1. Format 2 byte hex string
  + **Value - characteristic value read**
    1. Format: Hex String
    2. String Format: Each byte is separated by a whitespace

### NOTIFICATION | INDICATION EVENT

* FORMAT: 'EVT\_XXX\_RECEIVED: conn\_idx={id} handle={handle} Value: {value}\r\n'
* DATA:
  + **ID - Connection ID**
    1. Format: Hex String
    2. Value - 0x0000 - 0x0008
  + **Handle - Attribute handle**
    1. Format: 2 byte hex string
  + **Value - characteristic value read**
    1. Format: Hex String
    2. String Format: Each byte is separated by a whitespace

## SECURITY EVENTS

Any event associated with the pairing or bonding process is contained in this section.

### SECURITY LEVEL CHANGED

This event is triggered after connection and the central has completed the encryption process.

* FORMAT: 'BLE\_EVT\_GAP\_SEC\_LEVEL\_CHANGED: conn\_idx: {id} level:{level}\r\n'
* DATA:
  + **ID - Connection ID**
    1. Format: Hex String
    2. Value - 0x0000 - 0x0008
  + **Level - Security Level**
    1. SEE SET SECURITY EVT.

### PASSKEY NOTIFY EVENT

This event is triggered in the case that the central needs to display the passkey in legacy pairing mode.

* FORMAT: 'BLE\_EVT\_GAP\_PASSKEY\_DISPLAY: conn\_idx: {id} passkey: {passkey}\r\n'
* DATA:
  + **ID - Connection ID**
    1. Format: Hex String
    2. Value - 0x0000 - 0x0008
  + **Passkey - passkey to display**
    1. Format: uint32\_t String

### NUMERIC REQUEST EVENT

In the case pairing using secure connections, this event occurs.

* FORMAT: 'BLE\_EVT\_GAP\_PASSKEY\_DISPLAY: conn\_idx: {id} passkey: {passkey}\r\n'
* DATA:
  + **ID - Connection ID**
    1. Format: Hex String
    2. Value - 0x0000 - 0x0008
  + **Passkey - passkey to display**
    1. Format: uint32\_t String

### PASSKEY REQUEST EVENT

This event is triggered when the user on the central side needs to input a password displayed by the peripheral.

* FORMAT: BLE\_EVT\_GAP\_PASSKEY\_REQUEST: conn\_idx: {id}\r\n\
* DATA:
  + **ID - Connection ID**
    1. Format: Hex String
    2. Value - 0x0000 - 0x0008

### PAIR REQUEST EVENT

This event is triggered whenever a peripheral triggers a security request. The event will be shown and the central will start the pairing process.

* FORMAT: ' EVT\_PAIR\_REQUEST\r\n\tconn\_idx: {id}, bond: {bond}, mitm: {mitm}r\n'
* DATA:
  + **ID - Connection ID**
    1. Format: Hex String
    2. Value - 0x0000 - 0x0008
  + **Bond - if bonding should occur**
    1. Format: Boolean String
    2. Values: 0,1
  + **Mitm - if the pairing should include man-in-the-middle protection**
    1. Format: Boolean String
    2. Values: 0,1

### PAIR COMPLETED

Whenever the central has completed the pairing process.

* FORMAT: ' EVT\_GAP\_PAIR\_COMPLETE\r\n\tconn\_idx: {id}, bond: {bond}, mitm: {bond}, status: {status}\r\n'
* DATA:
  + **ID - Connection ID**
    1. Format: Hex String
    2. Value - 0x0000 - 0x0008
  + **Bond - if bonding was used**
    1. Format: Boolean String
    2. Values: 0,1
  + **Mitm - if man-in-the-middle protection is used**
    1. Format: Boolean String
    2. Values: 0,1
  + **Status - status of the completed pairing process**
    1. Format: Enum String
    2. Values: 0 = Success
    3. >=1 See GAP ERROR CODES

### GET BONDS EVENT

Whenever a GETBONDS command is issued, this event is triggered.

* FORMAT: ' EVT\_GET\_BONDS: {num} devices\r\n',

A separate line will be displayed for each bonded device:

EVT\_BONDED\_DEVICE: slot: {slot}: addr: {addr} addr\_type: {type} bonded: {bonded} connected: {connected}\r\n'

The event will finish with the following line:

' EVT\_GET\_BONDS\_COMPLETE\r\n'

* DATA:
  + **Num - number of bonded devices**
    1. Format: Integer String
    2. Value - 0-8
  + **Slot - slot in database**
    1. Format: Integer String
    2. Value - 0-8
  + **Addr - Bluetooth Address**
    1. Format: 6-byte hex string with colon delimiter
  + **Type - Address Type**
    1. Format: Boolean String
    2. Values:
       1. 0- Public Address
       2. 1- Private Address
  + **Bonded - Is the device bonded**
    1. Format: Boolean String
    2. Value: 0,1
  + **Connected - Is the device currently connected**
    1. Format: Boolean String
    2. Value: 0,1

### BLE ADDRESS RESOLVED

This event occurs after a resolvable address has been solved.

* Format: 'BLE\_EVT\_GAP\_ADDR\_RESOLVED: conn\_idx: {id} addr: {addr} resolved\_addr: {r\_addr}r\n'
* DATA:
  + **Id - Connection ID**
    1. Format: 4-byte Hex String, no delimiters
    2. Values: 0000-0008
  + **Addr - Unresolved Address**
    1. Format: 6-byte Hex string with colon delimiter
  + **R\_addr - Resolved Address**
    1. Format: 6-byte Hex string with colon delimiter

## DATA LENGTH AND MTU EVENTS

The central supports a maximum MTU of 512 bytes, additionally it supports Data Length Extension by default. If the peer supports larger MTUs or it supports Data Length Extensions, the central will negotiate larger values if the default are not adequate.

### DATA LENGTH CHANGED EVENT

In the case that DLE has been negotiate, this event occurs.

* FORMAT: BLE\_EVT\_DATA\_LEN\_CHANGED: \tconn\_idx:{id} rx\_len:{rx\_len} tx\_len:{tx\_len}\rn"
* DATA
  + **ID - Connection ID**
    1. Format: Hex String
    2. Value - 0000 - 0008
  + **Rx\_len - maximum RX packet size**
    1. Format: Hex String
  + **Tx\_len: Maximum TX packet size**
    1. Format: Hex String

### MTU CHANGED EVENT

* FORMAT: 'BLE\_EVT\_MTU\_CHANGED: \tconn\_idx:{id} mtu\_size:{mtu}\r\n'
* DATA:
  + **ID - Connection ID**
    1. Format: Hex String
    2. Value - 0000 - 0008
  + **MTU - New MTU Size**
    1. Format: Hex String

## PEER INFORMATION EVENTS

The central will display the peer version information and also peer features are requested.

### PEER VERSION EVENT

This event occurs after a peer version request.

* FORMAT: ' BLE\_EVT\_GAP\_PEER\_VERSION: \tconn\_idx:{id} lmp\_version:{lmp} comp\_id:{comp\_id} lmp\_subversion:{lmp}\r\n'
* DATA
  + **ID - Connection ID**
    1. Format: Hex String
    2. Value - 0000 - 0008
  + **LMP - Link Manager Protocol**
    1. Format: Hex String
  + **SMP - Security Manager Protocol**
    1. Format: Hex String
  + **Comp\_id - Company ID**

### GET PEER FEATURES EVENT

The central will issue a LL get peer version and the response is displayed in this event.

* FORMAT: ' BLE\_EVT\_GAP\_PEER\_FEATURES: \t{id} features:{features}\r\n'
* DATA:
  + **ID - Connection ID**
    1. Format: Hex String
    2. Value - 0000 - 0008
  + **Features - peer features** 
    1. Format: 1 byte hex string
    2. Values:
       1. See Bluetooth SIG Standards

## SUOTA EVENTS

This describes the associated events related to SUOTA updates.

### SUOTA Ready Event

This event occurs whenever the appropriate perquisite steps have been mode following a connection.

* FORMAT: ' EVT\_SUOTA\_READY\r\n'
* DATA: NONE

### SUOTA NOTIF EVT

A change has been notified from the peripheral and the status is displayed in this event.

* FORMAT: ' EVT\_SUOTA\_NOTIF: status: {status}\r\n'
* DATA:
  + **Status - status of SUOTA process**
    1. Format: 1 byte Hex String
    2. Values:
       1. See suota\_status\_t for more information.

Subsequent lines could also be printed in case of error or completion of SUOTA.

* FORMAT: ' ERROR: SUOTA remote did not ACK data block'
* FORMAT: ' EVT\_SUOTA\_REBOOT\_REMOTE\r\n'

### SUOTA TRANSFER COMPLETE

* FORMAT: ' EVT\_SUOTA\_TRANSFER\_COMPLETE\r\n'
* DATA: NONE

### SUOTA SENT PATCH DATA EVENT

Whenever patch data is sent to the peer this event will display the current number of bytes sent. The purpose of this callback is to monitor progress of the current transfer.

* FORMAT: ' EVT\_SUOTA\_SENT: {b} bytes\r\n'
* DATA
  + **B - number of bytes sent**
    1. Format: Decimal String

### SUOTA VERSION EVENT

After the version characteristic is read this event will display the value.

* FORMAT: 'EVT\_SUOTA\_VERSION: v{version}\r\n'
* DATA:
  + **Version-SUOTA Version of Peer**
    1. Format: 1 byte Hex String

### SUOTA L2CAP PSM Event

If L2CAP is supported by the peer and the force\_gatt\_update was not used in SUOTASTART, then this event will occur.

* FORMAT: ' EVT\_SUOTA\_L2CAP PSM: 0x{psm}\r\n'
* DATA:
  + **PSM - Protocol Service Multiplexer version**
    1. Format: 2 byte Hex String

# LIMITATIONS

* Baud Rate on USB Kit
  + The DA1469x USB KIT will not work with 921600 and will only work with 1000000 Baud. 1000000 is not common for terminal tools (will work with PC) so if desired to test via terminal window, use 460800.

1. <Appendix Title>

Optional section, remove if not required. Add subsections as required (Appendix Heading 2 to 5).

<Insert text here>

* 1. <Appendix Subsection>

<Insert text here>

Revision History

Summarize the changes made to the product and/or the document, which are relevant to the customer. The latest revision is shown at the top of the Revision History table. Optionally show change details for the latest revision in a separate table row, with hyperlinks to the related sections.

|  |  |  |
| --- | --- | --- |
| Revision | Date | Description |
| 0.1 | 29-July-2020 | Initial Draft version. |

**Approvals**

This section should be completed by the author. As a style of hidden text, it does not appear in the final document. Please refer to section 6.1 of [CQD1008](http://dms/sites/Forms/QualityTemplates/CQD1008.docx) for further guidance.

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Section(s) Reviewed** | **Name** | **Date of Approval** |
| <Function> | <Section number(s)> | <Name> | <DD-Mmm-YYYY> |
| <Function> | <Section number(s)> | <Name> | <DD-Mmm-YYYY> |
| <Function> | <Section number(s)> | <Name> | <DD-Mmm-YYYY> |
| <Function> | <Section number(s)> | <Name> | <DD-Mmm-YYYY> |

Status Definitions

|  |  |
| --- | --- |
| Status | Definition |
| DRAFT | The content of this document is under review and subject to formal approval, which may result in modifications or additions. |
| APPROVED or unmarked | The content of this document has been approved for publication. |

Disclaimer

Unless otherwise agreed in writing, the Dialog Semiconductor products (and any associated software) referred to in this document are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of a Dialog Semiconductor product (or associated software) can reasonably be expected to result in personal injury, death or severe property or environmental damage. Dialog Semiconductor and its suppliers accept no liability for inclusion and/or use of Dialog Semiconductor products (and any associated software) in such equipment or applications and therefore such inclusion and/or use is at the customer’s own risk.

Information in this document is believed to be accurate and reliable. However, Dialog Semiconductor does not give any representations or warranties, express or implied, as to the accuracy or completeness of such information. Dialog Semiconductor furthermore takes no responsibility whatsoever for the content in this document if provided by any information source outside of Dialog Semiconductor.

Dialog Semiconductor reserves the right to change without notice the information published in this document, including, without limitation, the specification and the design of the related semiconductor products, software and applications. Notwithstanding the foregoing, for any automotive grade version of the device, Dialog Semiconductor reserves the right to change the information published in this document, including, without limitation, the specification and the design of the related semiconductor products, software and applications, in accordance with its standard automotive change notification process.

Applications, software, and semiconductor products described in this document are for illustrative purposes only. Dialog Semiconductor makes no representation or warranty that such applications, software and semiconductor products will be suitable for the specified use without further testing or modification. Unless otherwise agreed in writing, such testing or modification is the sole responsibility of the customer and Dialog Semiconductor excludes all liability in this respect.

Nothing in this document may be construed as a license for customer to use the Dialog Semiconductor products, software and applications referred to in this document. Such license must be separately sought by customer with Dialog Semiconductor.

All use of Dialog Semiconductor products, software and applications referred to in this document is subject to Dialog Semiconductor’s [Standard Terms and Conditions of Sale](http://www.dialog-semiconductor.com/standard-terms-and-conditions), available on the company website ([www.dialog-semiconductor.com](http://www.dialog-semiconductor.com/)) unless otherwise stated.

Dialog, Dialog Semiconductor and the Dialog logo are trademarks of Dialog Semiconductor Plc or its subsidiaries. All other product or service names and marks are the property of their respective owners.

© 2021 Dialog Semiconductor. All rights reserved.

RoHS Compliance

Dialog Semiconductor’s suppliers certify that its products are in compliance with the requirements of Directive 2011/65/EU of the European Parliament on the restriction of the use of certain hazardous substances in electrical and electronic equipment. RoHS certificates from our suppliers are available on request.

|  |  |  |  |
| --- | --- | --- | --- |
| Contacting Dialog Semiconductor | | | |
| United Kingdom (Headquarters)  Dialog Semiconductor (UK) LTD  Phone: +44 1793 757700  Germany  Dialog Semiconductor GmbH  Phone: +49 7021 805-0  The Netherlands  Dialog Semiconductor B.V.  Phone: +31 73 640 8822 | North America  Dialog Semiconductor Inc.  Phone: +1 408 845 8500  Japan  Dialog Semiconductor K. K.  Phone: +81 3 5769 5100  Taiwan  Dialog Semiconductor Taiwan  Phone: +886 281 786 222 | Hong Kong  Dialog Semiconductor Hong Kong  Phone: +852 2607 4271  Korea  Dialog Semiconductor Korea  Phone: +82 2 3469 8200 | China (Shenzhen)  Dialog Semiconductor China  Phone: +86 755 2981 3669  China (Shanghai)  Dialog Semiconductor China  Phone: +86 21 5424 9058 |
| Email:  enquiry@diasemi.com | Web site:  www.dialog-semiconductor.com |  |  |