

# User manual

## DA14580 Proximity application

### UM-B-010

#### **Abstract**

*This document describes the proximity example applications of the DA14580 Software Development Kit. The software architecture of a host application running on an external processor and the interface with the DA14580 is also presented.*

## Contents

<b>Contents .....</b>	<b>2</b>
<b>Figures.....</b>	<b>2</b>
<b>Tables .....</b>	<b>2</b>
<b>1 Terms and definitions .....</b>	<b>3</b>
<b>2 References .....</b>	<b>3</b>
<b>3 Introduction.....</b>	<b>4</b>
<b>4 Features.....</b>	<b>4</b>
<b>5 Getting started .....</b>	<b>5</b>
<b>6 External processor proximity applications .....</b>	<b>9</b>
6.1 Host application architecture.....	10
6.2 General Transport Layer (GTL) interface.....	11
6.3 BLE software stack .....	12
6.3.1 Generic Access Profile .....	12
6.3.2 Proximity profile .....	14
6.3.3 Device Information Service.....	15
6.3.4 DA14580 external processor mode application.....	16
6.4 Physical interface .....	16
<b>7 Integrated processor proximity application .....</b>	<b>17</b>
<b>8 Revision history .....</b>	<b>18</b>

## Figures

Figure 1: Installing the driver for the BLE USB dongle.....	6
Figure 2: Proximity monitor host program started .....	6
Figure 3: Proximity Monitor enters scanning mode.....	7
Figure 4: Proximity monitor has detected the proximity reporter device .....	7
Figure 5: Proximity monitor connected to proximity reporter.....	8
Figure 6: Proximity reporter's DIS data .....	9
Figure 7: External processor application architecture .....	10
Figure 8: Packet format .....	12
Figure 9: Packet format of the set device configuration command .....	12
Figure 10: GAP interface .....	13
Figure 11: Direct Connection flow chart .....	14

## Tables

Table 1: Proximity applications source code files .....	11
Table 2: Message Fields .....	12
Table 3: DA14580 application files .....	16
Table 4: Integrated processor application source code files .....	17

## 1 Terms and definitions

BLE	Bluetooth low energy
GAP	Generic Access Profile
GATT	Generic Attribute Profile
GTL	Generic Transport Layer
HCI	Host Controller Interface
RSSI	Received Signal Strength Indicator
SDK	Software Development Kit

## 2 References

1. UM-B-015, DA14580 Software Architecture, Dialog Semiconductor.
2. UM-B-014, DA14580 Development Kit, Dialog Semiconductor.
3. RW-BLE-HOST-IS, Host Interface Specification, Riviera Waves.
4. RW-BLE-HOST-SW-FS, Host Software, Riviera Waves.
5. UM-B-003, DA14580 Software development guide, Dialog Semiconductor.
6. RW-BLE-GAP-IS, GAP Interface Specification, Riviera Waves.
7. RW-BLE-PRF-DIS-IS, Device Information Service Interface Specification, Riviera Waves.
8. RW-BLE-PRF-PXP-IS, Proximity Profile Interface Specification, Riviera Waves.

### 3 Introduction

The Proximity profile defines the behaviour of a Bluetooth device when this moves away from a peer device so that the connection is dropped or the path loss increases above a pre-set level, causing an immediate alert. This alert can be used to notify the user that the devices have become separated.

The Proximity profile can also be used to define the behaviour of two devices coming closer together such that a connection is made or the path loss decreases below a pre-set level.

The proximity profile defines two roles:

- Proximity Monitor (PM). The Proximity Monitor shall be a GATT client.
- Proximity Reporter (PR). The Proximity Reporter shall be a GATT server.

The DA14580 Software Development Kit (SDK) is described in detail in [1]. SDK releases include five Keil projects that implement the two roles of the proximity profile:

***Proximity Monitor External Processor configuration for DA14580 DK:***

*dk\_apps\keil\_projects\proximity\monitor\_fe\fe\_proxm\_sdk.uvproj*

***Proximity Monitor - External Processor configuration for DA14580 USB Dongle:***

*dk\_apps\keil\_projects\proximity\monitor\_fe\_usb\fe\_usb\_proxm\_sdk.uvproj*

***Proximity Reporter – Integrated Processor configuration:***

*dk\_apps\keil\_projects\proximity\reporter\_fh\fh\_proxr\_sdk.uvproj*

***Proximity Reporter – Integrated Processor configuration for DA14580 DK:***

*dk\_apps\keil\_projects\proximity\reporter\_fe\fe\_proxr.uvproj*

***Proximity Reporter – Integrated Processor configuration for DA14580 USB Dongle:***

*dk\_apps\keil\_projects\proximity\reporter\_fe\_usb\fe\_usb\_proxr.uvproj*

The host applications that control the Proximity Monitor and Proximity Reporter are implemented in Microsoft Windows environment based on Visual C++ 2010 Express.

These projects reside at

**host\_apps\windows\proximity\monitor\host\_proxm\_sdk\host\_proxm\_sdk.vcxproj**

and **host\_apps\windows\proximity\reporter\host\_proxr\_sdk\host\_proxr\_sdk.vcxproj**, respectively.

### 4 Features

**Proximity Reporter:**

- Immediate Alert service
- Link Loss Service
- Tx Power service
- 2 levels of Alert Indications. Mild/High -> Slow/Fast green LED blinking
- 500 ms Advertise Interval
- 'Deep Sleep' mode after 3 minutes of inactivity
- Push Button
- Stop Alert indications
- Exit "Deep Sleep" mode
- Pairing / Bonding / Encryption
- Support extended sleep mode
- Data Information Service Server

**Proximity Monitor:**

- Immediate Alert service
- Link Loss Service
- Tx Power service
- Device discovery and disconnection
- Pairing / Bonding / Encryption
- RSSI sampling / Path Loss Alert
- Support extended sleep mode
- Data Information Service Client

## 5 Getting started

This section describes how to set up the application software and the DA14580 Development board to run the proximity monitoring application. It is assumed that the user is familiar with using the DA14580 Development Kit (SDK) [3].

**STEP 1: Setting up the Proximity Reporter device for LED alert support**

Before testing any proximity monitoring alert events, the LED and push button interface needs to be set up properly.

On the DA14580 Development board, which will be used as the Proximity Reporter device, PIN3 of J16 connector must be connected to PIN4. PIN5 must be connected to PIN6 in order to enable LED D2 and push button K2 as a user interface.

The Proximity Reporter will notify the user when an alert indication, link loss and immediate alert, is triggered. The Alert Notification will be:

- High level alert indication. A fast (500 ms) green LED blinking.
- Mild level alert indication. A slow (1500 ms) green LED blinking.

The user can stop alert notification by pressing the push button K2.

**STEP 2: Starting the Proximity Reporter device (Integrated Processor configuration)**

Use a Windows machine and start the Keil environment. Now open the Proximity Reporter project:

**dk\_apps\keil\_projects\proximity\reporter\_fh\fh\_proxr\_sdk.uvproj**

Build this project (F7) and follow the steps in the user guide [3] to load the executable to the SDK and run the executable.

Debug session must be stopped (Ctrl + F5) while executable is running, to start application. Debugger will lose connection with DA14580.

**STEP 3: Running Proximity Monitor device (External Processor configuration for USB dongle)**

Start the Keil environment and open the following project:

**dk\_apps\keil\_projects\proximity\monitor\_fe\_usb\fe\_usb\_proxm\_sdk.uvproj**

Build this project (F7) and follow the steps in the user guide [3] to load the executable to the SDK/USB dongle and run it. If applications runs on SDK, debug session must be stopped (Ctrl + F5) while executable is running, to start application. Debugger will lose connection with DA14580.

**STEP 4: Running the Host application**

DA14580 USB Dongle will be used as the Proximity Monitor device. When dongle is inserted in a Windows machine (e.g. laptop) a J-Link device should be discovered in windows devices and printers. In J-Link's properties a JLink CDC UART Port is displayed. User must keep the virtual COM port number to provide it to Proximity monitor console application.

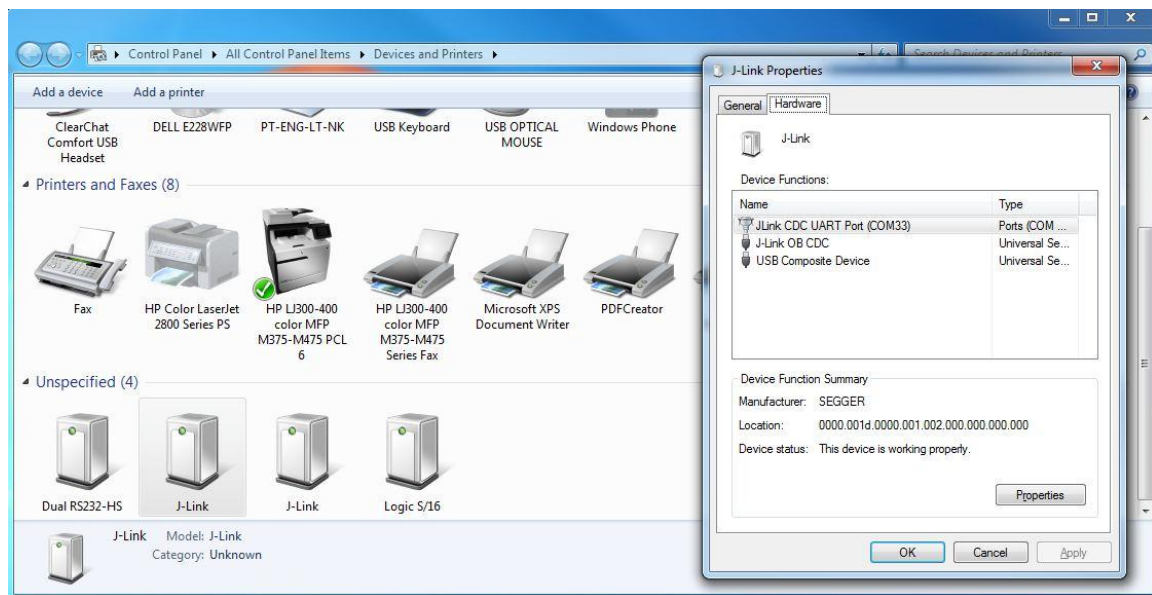


Figure 1: Installing the driver for the BLE USB dongle

The host application, which is part of the Proximity Monitor, needs to run first. User can run directly from the command line precompiled executable file:

**host\_binaries\windows\proximity\monitor\host\_proxm\_sdk.exe**

Alternatively, proximity monitor application's Microsoft Visual C++ 2010 Express project can be used.

- Open the project  
**host\_apps\windows\proximity\monitor\host\_proxm\_sdk\host\_proxm\_sdk.vcxproj**  
with Microsoft Visual C++ 2010 Express.
- Build the project (F7).
- Run the program (F5).

At this point, a console should open as shown in the [Figure 2](#) below:

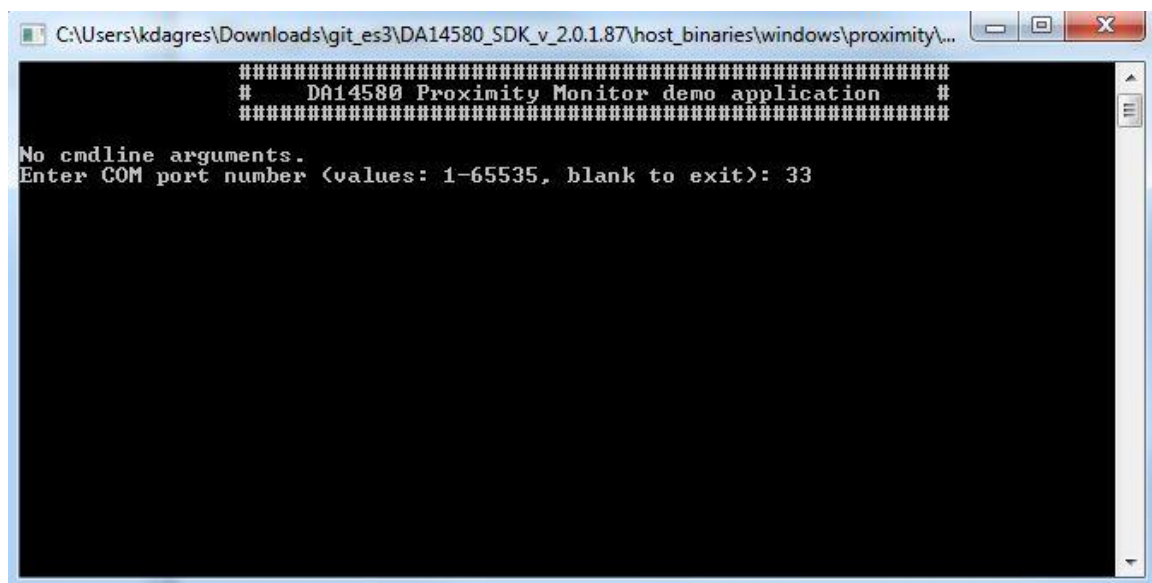


Figure 2: Proximity monitor host program started

User must provide virtual USB COM port number. When COM port is provided, the proximity monitor device will communicate via the proprietary HCI [4] with the Host application (console). The Host

application will receive a message via the UART that the BLE stack is initialized. Then it will send a message to the BLE stack to set it to scanning mode and the console will display the following message, as shown in [Figure 3](#).

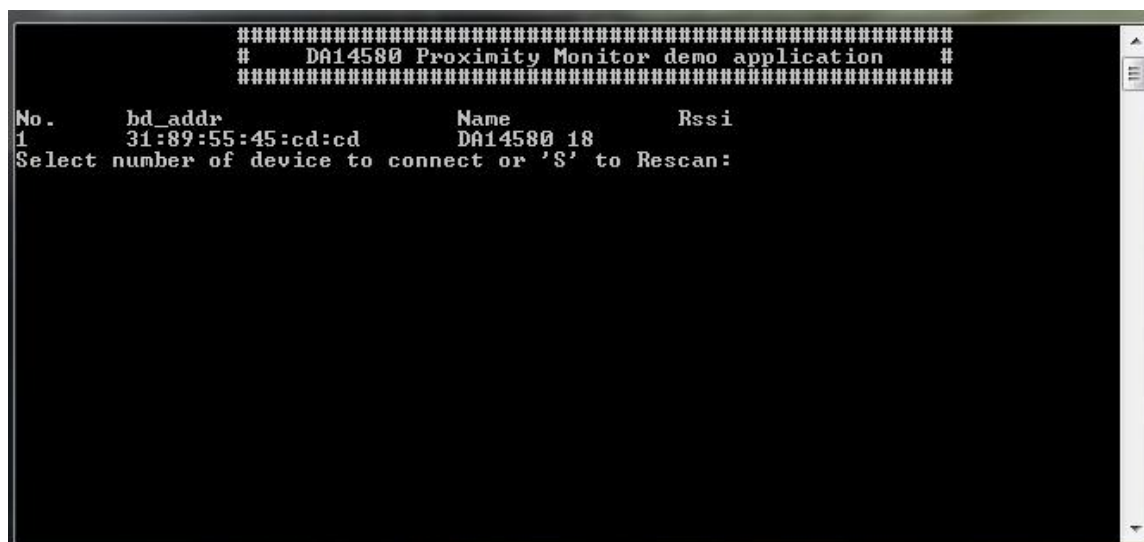


```
#####
#   DA14580 Proximity Monitor demo application   #
#####
Scanning... Please wait
No.      bd_addr      Name      Rssi
```

**Figure 3: Proximity Monitor enters scanning mode**

The Proximity Monitor host application will update the console with a list of the detected devices as shown in the [Figure 4](#) below:

**Note 1** If the Proximity Monitor has exited scanning mode, then 'S' can be pressed to re-enter scanning mode as shown in the console message).



```
#####
#   DA14580 Proximity Monitor demo application   #
#####
No.      bd_addr      Name      Rssi
1        31:89:55:45:cd:cd  DA14580 18
Select number of device to connect or 'S' to Rescan:
```

**Figure 4: Proximity monitor has detected the proximity reporter device**

Proximity Reporter is displayed in discovered devices list. User must select devices number (i.e. '1') and proximity monitor will be connected, paired and bonded with proximity reporter. The console shown in [Figure 5](#).

```

#####
#   DA14580 Proximity Monitor demo application   #
#####

##### Scan List #####
#
# No.  bd_addr          Name          Rssi
# 1    ba:be:fa:ce:f0:0d  Keyfobdemo -52 dB
#
##### Connected Devices #####
#
# No. Model No.      BDA          Bonded  RSSI    LLA  TX_PL
# * 1 DA14580        99:77:15:51:77:99 YES    -53 dB  00
# 2 DA14580        23:10:3a:55:89:12 YES    -54 dB  00
# 3          -- Empty Slot --
# 4          -- Empty Slot --
# 5          -- Empty Slot --
# 6          -- Empty Slot --
#
#####
----- Select Device -----
##### Options #####
#
# 'A' - Read Link Loss Alert Level
# 'B' - Read Tx Power Level
# 'C' - Start High Level Immediate Alert
# 'D' - Start Mild Level Immediate Alert
# 'E' - Stop Immediate Alert
# 'F' - Set Link Loss Alert Level to None
# 'G' - Set Link Loss Alert Level to Mild
# 'H' - Set Link Loss Alert Level to High
# 'P' - Connect to device
# 'I' - Disconnect from device
# 'S' - Rescan Advertising devices
#
# 'Q' - Display/Hide Device Information
# 'Esc' - Exit
#
#####

```

Figure 5: Proximity monitor connected to proximity reporter

**Scan list section**

User can refresh the advertising devices list by pressing “S”. Host application sends a scan message to the embedded device.

To connect to another device from the scan list user has to press “P” and then the number of the device followed by “enter”.

**Connected devices section**

Selected device is distinguished by the star character before the number.

Pressing a number key selects the respective device from the connected devices list, if available. User can also navigate through the list with up-down arrow keys.

Displayed options A through H are referring to the selected device from the connected devices list and perform the corresponding task.

Pressing “I” disconnects the selected device and removes it from the list after a confirmation is received.



```
#####
#   DA14580 Proximity Monitor demo application   #
#####

##### Scan List #####
#
# No.  bd_addr          Name          Rssi          #
#
##### Connected Devices #####
#
# No.  Model No.      BDA          Bonded  RSSI    LLA  TX_PL  #
# * 1  DA14580        99:77:15:51:77:99  YES    -55 dB  00    00    #
# 2  DA14580        23:10:3a:55:89:12  YES    -50 dB  00    00    #
# 3              -- Empty Slot --          #
# 4              -- Empty Slot --          #
# 5              -- Empty Slot --          #
# 6              -- Empty Slot --          #
#
#####
#
# Device Information
# -----
#
# Manufacturer: Dialog Semi
# Model Number: DA14580
# Serial Number:
# Hardware Revision:
# Firmware Revision:
# Software Revision: v_3.0.2.139
# System ID: 12:34:56:ff:fe:9a:bc:de
# IEEE Certification: 00
# PnP ID: 00
#
#####
#
# Options
#
# 'Q' - Display/Hide Device Information
# 'Esc' - Exit
#
#####
```

Figure 6: Proximity reporter's DIS data

When maximum connection number is reached and user tries to connect to another device a message is displayed that informs the user.

## 6 External processor proximity applications

This section describes the external processor configuration mode of the DA14580, the interface between the application running in an external processor and the BLE stack of the DA14580. This description can be used as a reference for any solution an external microcontroller running the application layer is interfacing to the DA14580. It finally presents the basic software architecture of the proximity applications.

In the external processor configuration, the application is implemented in an external processor while the BLE stack (controller, host stack, profiles) are implemented inside the DA14580 chip (Figure 7). The structure of the BLE stack is given in details in [1], [4].

These two components communicate via a proprietary interface, i.e. Generic Transport Layer (GTL) over UART as shown in Figure 7.

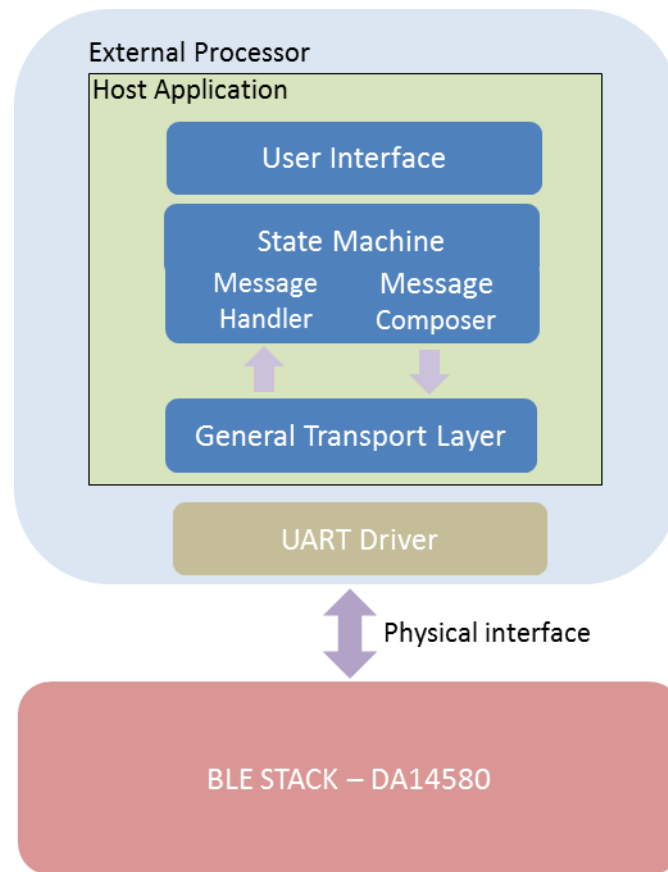


Figure 7: External processor application architecture

## 6.1 Host application architecture

The software architecture of the host architecture is shown in Figure 7. The User Interface, if any, is the module that interacts with the user. The State Machine module implements the finite-state machine of the host application. It stores the application state, takes as input the incoming events or interacts with the user in order to generate a command to the DA14580. The state machine changes state and/or provides information to the end-user. The message handler module is responsible to receive the events from the DA14580, check the integrity, analyse them and call the appropriate handlers of the state machine. The message composer module is responsible to create a packet with the command data provided by the state machine and send it to GTL. GTL module is responsible to receive or transmit the packets from/to UART interface.

### Example

Two example applications following the external processor configuration architecture are provided in the DA14580 SDK, the proximity reporter and proximity monitor. The source code files are described in Table 3. Both projects include header files from the BLE software stack to obtain the message types, message structures, task ID, etc. for supported messages.

Table 1: Proximity applications source code files

Module	Filename	Description
User Interface	console.c	User interface
State machine, Message composer	app.c	Application state machine and message composer.
	app_task.c	BLE message handlers
Message Handler	ble_msg.c	BLE message reception and dispatching to the corresponding handler.
	queue.c	Inter-tasks messaging queue
GTL	uart.c	GTL layer

At start-up, the host application expects to receive a `GAPM_DEVICE_READY_IND` by the DA14580 device and it sends a `GAPM_RESET_CMD` command to the GAPM.

The GAPM will respond with `GAPM_CMP_EVT` and the handler `gapm_cmp_evt_handler()` will be called, resulting in sending the command `GAPM_SET_DEV_CONFIG_CMD` to `TASK_GAPM`. The proximity reporter may send enable request to the DISS profile task to set the device configuration mode.

After GAPM initialization and when the `GAPM_CMP_EVT` has been received in `TASK_APP`, the application initialize the profile database. After each database has been initialized, the profile task will send the `XXX_CREATE_DB_CFM` (where XXX is the name of the profile) to the application. Then the `xxx_create_db_cfm_handler()` will be called and send the `APP_MODULE_INIT_CMP_EVT` from `TASK_APP` to `TASK_APP`. At the end, the `app_adv_start()` will be called to start the advertising procedure in the proximity reporter application. For the proximity monitor application, the `app_inq()` will be called to start the scanning procedure by issuing a `GAPM_START_SCAN_CMD` command. This will be followed by the incoming `GAPM_ADV_REPORT_IND` event, which will contain the details of the discovered device. The proximity monitor application will then send a `GAPM_START_CONNECTION_CMD` message to start a connection establish phase. It will be notified of the completion or failure of the connection with a `GAPC_CONNECTION_REQ_IND` message.

The function `BleMsgAlloc()` allocates memory space for an outgoing packet and fills the message type, source and destination task ID and the size of the message data.

The function `HandleBleMsg()`, in `ble_msg.c` file, is responsible for message reception.. It checks the destination task ID and drops packets that are not destined to the host application. Then, it checks the message type and dispatches the incoming message to the appropriate message handlers of the state machine. The events 'Command Compete' from the Generic Access Profile (GAP) are handled in `HandleGapmCmpEvt()` and `HandleGapcCmpEvt()` functions..

The proximity example applications handle incoming and outgoing messages across GAP, Proximity profile, DIS service and Link Layer Controller (LLC) tasks.

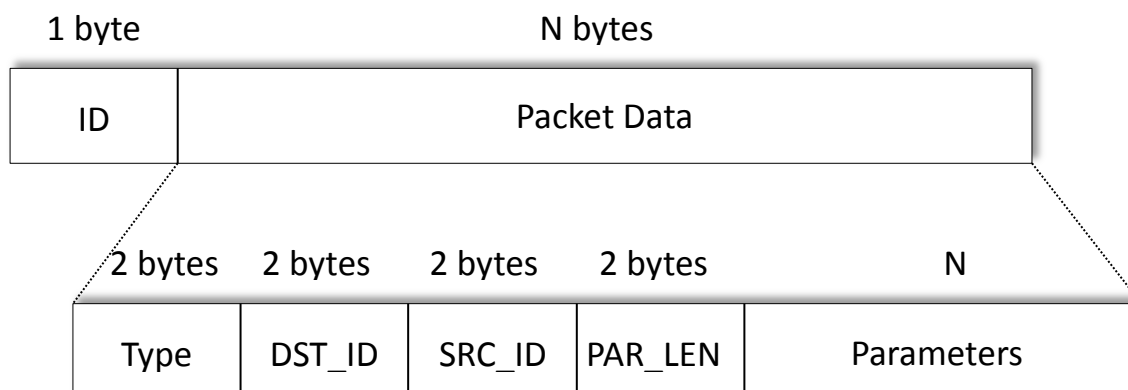
## 6.2 General Transport Layer (GTL) interface

The GTL interface defines the communication protocol between the host application and the BLE stack located inside the DA14580.

In Figure 8, the packet format is illustrated. A single byte at the beginning of the packet is used as a packet identifier (**ID**) in order to indicate the type of the packet. The value of the identifier (**ID**) is 0x5 for both incoming and outgoing packets. The structure of the **packet data** is given in Table 1. The packet format is the same for sending and receiving data and the same packet identifier (0x5) is used for both directions. The protocol uses only commands for sending data from the host application to the BLE stack and events in the opposite direction.

The field **Type**, in the packet format, is 2 bytes and defines a valid message type of the destination task defined in field **DST\_ID**. The number of tasks supported by the DA14580 SDK is defined in `ble_sw\dk_apps\src\ip\ble\hl\src\rwble_hl\rwble_hl_config.h`. More information about the valid message types and the tasks are given in the Interface Specification (IS) documents of the corresponding BLE stack tasks or profiles. The field **SRC\_ID** defines the source task of a valid message. The field **PAR\_LEN** is 2 bytes and defines the size, in bytes, of the message **parameters**.

The content and the size of the message parameters depend of the specific message and they are defined in [6,7,8].



**Figure 8: Packet format**

**Table 2: Message Fields**

Field	Size	Description
Type	2	Message Identification. A valid Message ID of a BLE stack task or profile task API. API message IDs are defined in the Interface Specification documents of the corresponding BLE stack task or profile.
DST_ID	2	Destination Task Identification. A task ID of the BLE stack tasks (i.e. TASK_GAPM, TASK_GAPC, etc.) or the supported by the DA14580 application profile tasks.
SRC_ID	2	Source Task Identification.
PAR_LEN	2	Message Parameter Length. The size, in bytes, of the Parameters field..
Parameters	N	Message Parameters. Format of this field depends on MSG_ID. API message formats are described in Interface Specification documents of the corresponding BLE stack task or profile.

### Example

An example of a valid outgoing packet is given in Figure 9. It corresponds to the Set Device Configuration command sent from the host application to the BLE stack.

The message type is GAPM\_SET\_DEV\_CONFIG\_CMD, the destination task is the TASK\_GAPM, the source task is the host application (TASK\_GTL), the parameter size is 33 (sizeof(gapm\_set\_dev\_config\_cmd )) and the message parameter is the content of the gapm\_set\_dev\_config\_cmd. More information on GAPM commands is given in [6].

0x5	GAPM_SET_DEV_CONFIG_CMD	TASK_GAPM	TASK_GTL	33	gapm_set_dev_config_cmd
-----	-------------------------	-----------	----------	----	-------------------------

**Figure 9: Packet format of the set device configuration command**

## 6.3 BLE software stack

This section provides an overview of the modules and tasks accessed by the proximity applications. The BLE software stack is described in document [1].

### 6.3.1 Generic Access Profile

GAP is responsible for the following services:

- Four roles –central, peripheral, broadcaster and scanner
- Broadcast and Scan
- Modes –Discovery, Connectivity, Bonding
- Security with Authentication, Encryption and Signing
- Link Establishment and Detachment
- Random and Static Addresses
- Privacy Features
- Pairing and Key Generation

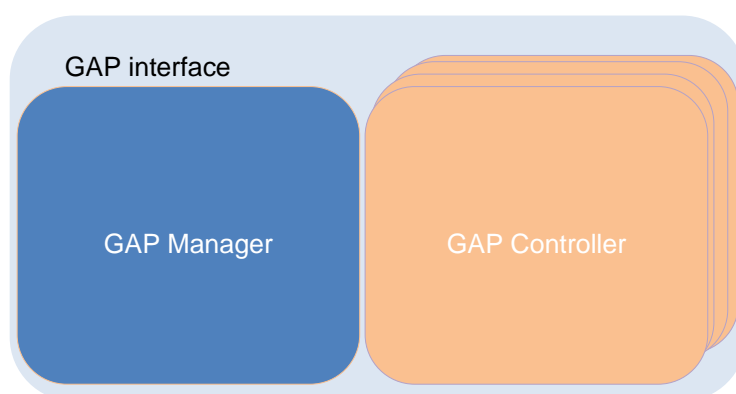


Figure 10: GAP interface

GAP is divided into two parts (Figure 10). First task is mono instantiated and manages all application requests that are not related to an established link (Device configuration). This task is the GAP Manager (GAPM) it also manages creation or suppression of the second type of GAP task: GAP Controller (GAPC). GAPC task is multi instantiated; an instance of GAPC is created when a connection to a peer device is created and deleted when this connection is terminated. Index of the created task is related to a connection index created for the connection.

The **Generic Access Profile Manager** is the GAP task used to manage device configuration:

- Discover/scan for Bluetooth LE devices
- Send advertising data for device that scanning or establishing a connection
- Start connection establishment.

Messages exchanged to and from the RW-BLE GAP can be any of the following:

- **Command:** Always completed with “**complete event**” message.
- **Indication**
- **Indication request:** It requires a **confirmation** message from application.

The GAP Manager block has handlers for the messages defined in `gapm_task` files (.h/.c).

The GAP Manager provides also a security manager message API in order to perform some security operations. Those operations are not related to an active link. It could be used to:

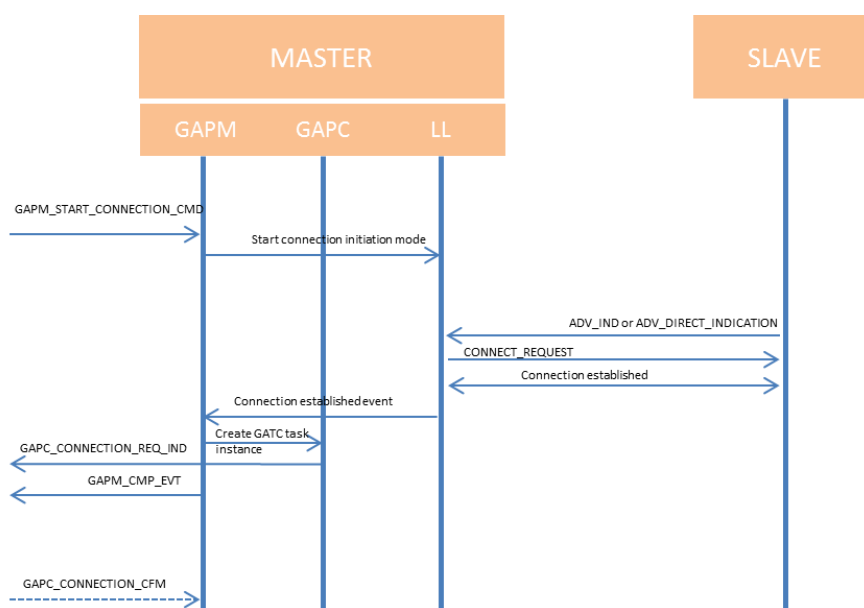
- Resolve resolvable random address.
- Generate keys.
- Generate random BD Addresses (Static or Non Resolvable).

**Note 2** Security manager does not provide an API for application, so security features shall be accessed through GAP API.

The **Generic Access Profile Controller** is a multi-instantiated GAP task used to manage connection to a peer device.

The GAPC API should be used to:

- Retrieve peer device information
- Manage peer device privacy configuration (if Central)
- Start pairing procedure
- Encrypt the link
- Disconnect the link



**Figure 11: Direct Connection flow chart**

One task instance is created for each established link. Each instance of the task is related to a connection index (conidx) with a valid value range: **[0 : BLE\_CONNECTION\_MAX]**

Corresponding GAPC task instance can be retrieved by doing: **(conidx << 8) | TASK\_GAPC**.

Messages exchanged to and from the RW-BLE GAP can be any of the following:

- **Command:** Always completed with “**complete event**” message
- **Indication**
- **Indication request:** It requires a **confirmation** message from application.

The GAP Controller block has handlers for these messages, defined in gapc\_task files (.h/.c).

An example of accessing the GAPM and GAPC tasks for issuing a connection establishment command (direct connection mode) is shown in Figure 11. The command GAPM\_START\_CONNECTION\_CMD is issued to GAPM task and the event GAPM\_CMP\_EVT is received on command completion. More information is given in [6]

### 6.3.2 Proximity profile

This section provides an overview of the Proximity profile (PXP) implementation.

The Bluetooth Low Energy Proximity profile enables the user to monitor proximity/link loss between two devices. Within the profile, two roles can be supported: **Monitor and Reporter**. The profile requires a connection to be established between the two devices for its functionality.

The functionality of a profile requires the presence of certain services and attributes on one of the two devices, which the other device can manipulate. In this case, the Proximity Reporter device must have an instance of the Link Loss Service (LLS), and may also have the Immediate Alert Service (IAS) and Tx Power Service (TPS) in its attribute database. The two last ones must be used together, if one is missing, the other one should be ignored.

The Proximity Monitor will discover these services and their Characteristics: the Alert Level Characteristic in LLS, the Alert Level Characteristic in IAS and the Tx Power Level Characteristic in TPS.

The LLS allows the user to set an alert level in the Reporter, which will be used by the reporter to alert in the corresponding way if the link is lost. The disconnection must not come voluntarily from one of the two devices in order to trigger the alert.

The IAS allows the user to set an immediate alert level based on path loss computation using the read Tx Power Level and RSSI monitored on received packets. According to the alert level set in IAS, the Reporter will start alerting immediately.

The TPS allows the user to read the Tx Power Level for the physical layer. The value is used by the Monitor to continuously evaluate path loss during the connection, and decide to trigger/stop an alert based on path loss going over/under a set threshold in the Monitor application.

The profile is implemented in the software stack as two tasks, one for each role. More information is given in [8].

### Example

A usage example of proximity monitor is described in this paragraph.

Upon connection of Proximity Monitor device (reception of GAPC\_CONNECTION\_REQ\_IND message) host application must enable Proximity Monitor profile by sending PROXM\_ENABLE\_REQ. If this is the first connection to peer device con\_type field must be set to PRF\_CON\_DISCOVERY. Profile will start discovering peer device's service's and characteristics handles. At the completion of discovery procedure profile will send a PROXM\_ENABLE\_CFM message to host application with the details (handles of services and characteristics of the discovered services). Host application must use this handles for read and write access on peer device characteristics. Moreover It can store this information for future reconnections in order to avoid re-discovering services and reduce the transactions with peer device.

Host application can read the characteristic values of Link Loss and Tx Power services by sending PROXM\_RD\_ALERT\_LVL\_REQ and PROXM\_RD\_TXPW\_LVL\_REQ messages respectively. Profile responds to host application with a PROXM\_RD\_CHAR\_RSP for both requests.

Application can write either Link Loss or Immediate Alert values with PROXM\_WR\_ALERT\_LVL\_REQ message. Profile, depending on the requested service, selects the method (GATTC\_WRITE for Link Loss and GATTC\_WRITE\_NO\_RESPONSE for Immediate Alert) to write characteristic value on peer device. When operation completion indication message is received profile sends a PROXM\_WR\_CHAR\_RSP confirmation to application.

### 6.3.3 Device Information Service

This section provides an overview of the Device Information Service (DIS) implementation.

The Bluetooth Low Energy Device Information Service enables the user to expose manufacturer and/or vendor information about a device. Within the profile, two roles can be supported: Client and Server. The profile requires a connection to be established between the two devices for its functionality.

#### Device Information Service Server (DISS)

This role is meant to be activated on the device that share information. It implies it is a GAP Peripheral. More information is given in [7]

#### Device Information Service Client (DISC)

This role is meant to be activated on the device that will locate the Server. It implies it is a GAP Central. The task (TASK\_DISC) for this role will discover the Device Information Service present on the peer Server, after establishing connection, and will allow reading different information about the device.



**Note 3** The TASK\_DISC task is multi-instantiated, one instance is created for each connection for which the profile will be enabled and each of these instances will have a different task ID. Thus, it is very important for the application to keep the source task ID of the DISC\_ENABLE\_CFM message to be able to communicate with the peer device linked to this task ID once it has been enabled.

### 6.3.4 DA14580 external processor mode application

The procedure for creating an application in DA14580 for the external processor application is described in [1].

The main files of the proximity application project in DA14580 are listed in the Table 3.

**Table 3: DA14580 application files**

Group	File	Description
boot	system_ARMCM0.c boot_vectors.s hardfault_handler.c	Application start-up, ISR vectors, hard fault handler
arch	arch_main.c jump_table.c arch_sleep.c nmi_handler.c arch_system.c	Main loop, kernel scheduling, system setup, system clocks, common sysRAM/ROM code structure (jump_table), sleep modes API
	periph_setup.c	Peripheral modules initialization, GPIO pins assignement.
driver	rf_580.c	Radio
	gpio.c	GPIO driver
nvds	nvds.c	Non-volatile data structure
rwble	rwble.c	BLE core interrupt service routines
	rwip.c	BLE sleep function
profiles	proxr.c proxr_task.c	Proximity Reporter profile
	diss.c diss_task.c	Device information service, server role
	findl.c findl_task.c	Find-me Locator profile

## 6.4 Physical interface

The physical interface connects the external processor and the DA14580. UART interface is used by the applications examples provided in SDK distribution, however, other interfaces (SPI, I2C) may be used for customized applications.

Host Application must configure the UART interface according to DA14580 specifications. The default settings for the DA14580 Development Kit [2] are:

- Speed (baudrate): 115200
- Flow control: RTS/CTS
- Data bits: 8
- Stop bits: 1
- Parity: None



## 7 Integrated processor proximity application

A proximity reporter application is provided as example of the integrated processor configuration mode in DA14580 SDK.

In integrated processor configuration, the application runs in DA1450 but the software architecture and the interface to BLE stack are the same with the architecture and interface described in the section for the external configuration with only difference the absence of the transport and physical layer.

The procedure for creating an integrated processor application is given in [1], [2]. In document [2], a step by step methodology for creating a new application project starting from the application template is described.

The integrated processor proximity reporter included in DA14580 SDK supports the Proximity Profile in peripheral role, the battery service, the Device Information Service and the Find-me Locator Profile.

The main files of the proximity reporter application project in DA14580 are listed in the Table 4.

**Table 4: Integrated processor application source code files**

Group	File	Description
boot	system_ARMCM0.c boot_vectors.s hardfault_handler.c	Application start-up, ISR vectors, hard fault handler
arch	arch_main.c jump_table.c arch_sleep.c nmi_handler.c arch_system.c	Main loop, kernel scheduling, system setup, system clocks, common sysRAM/ROM code structure (jump_table), sleep modes API
	periph_setup.c	Peripheral modules initialization, GPIO pins assignment.
driver	wkupct_quadec.c	Wakeup/quadec controller driver for push button functionality.
	battery.c	Battery driver. Used for Battery service server.
	adc.c	Analog to Digital convertor. Required for battery driver
profiles	bass.c bass_task.c	Battery service, server role
app	app.c app_task.c app_sec.c app_sec_task.c	Common Integrated host application code. Main BLE functionality, application task message handlers, security functions etc.
	app_proxr_proj.c	Application specific functionality.
	app_proxr.c app_proxr_task.c	Application layer proximity reporter functionality, message handlers
	app_bass.c app_bass_task.c	Application layer battery service server functionality, message handlers
	app_dis.c app_dis_task.c	Application layer device information service server functionality, message handlers
	app_findme.c app_findme_task.c	Application layer find-me locator functionality, message handlers

## 8 Revision history

Revision	Date	Description
1.0	09-May-2014	Initial version
1.1	08-Jul-2014	Updated after adding support for 6 connections in the proximity monitor

**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**

Information in this document is believed to be accurate and reliable. However, Dialog Semiconductor does not give any representations or warranties, expressed 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.

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.

Customer notes that 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 are subject to Dialog Semiconductor's [Standard Terms and Conditions of Sale](#), unless otherwise stated.

© Dialog Semiconductor GmbH. All rights reserved.

**RoHS Compliance**

Dialog Semiconductor complies to European Directive 2001/95/EC and from 2 January 2013 onwards to European Directive 2011/65/EU concerning Restriction of Hazardous Substances (RoHS/RoHS2).

Dialog Semiconductor's statement on RoHS can be found on the customer portal <https://support.diasemi.com/>. RoHS certificates from our suppliers are available on request.

**Contacting Dialog Semiconductor****Germany Headquarters**

Dialog Semiconductor GmbH

Phone: +49 7021 805-0

**United Kingdom**

Dialog Semiconductor (UK) Ltd

Phone: +44 1793 757700

**The Netherlands**

Dialog Semiconductor B.V.

Phone: +31 73 640 8822

**Email:**

[enquiry@diasemi.com](mailto:enquiry@diasemi.com)

**North America**

Dialog Semiconductor Inc.

Phone: +1 408 845 8500

**Japan**

Dialog Semiconductor K. K.

Phone: +81 3 5425 4567

**Taiwan**

Dialog Semiconductor Taiwan

Phone: +886 281 786 222

**Web site:**

[www.dialog-semiconductor.com](http://www.dialog-semiconductor.com)

**Singapore**

Dialog Semiconductor Singapore

Phone: +65 64 849929

**China**

Dialog Semiconductor China

Phone: +86 21 5178 2561

**Korea**

Dialog Semiconductor Korea

Phone: +82 2 3469 8291