Texas Instruments CC2540/41
Bluetooth® Low Energy
Sample Applications Guide
v1.2

Document Number: SWRU297

Table Of Contents

References

Included with Texas Instruments *Bluetooth* Low Energy v1.2 Stack Release (All path and file references in this document assume that the BLE development kit software has been installed to the default path C:\Texas Instruments\BLE-CC254-1.2\):

[1] Texas Instruments *Bluetooth*® Low Energy Software Developer's Guide (SWRU271B)

C:\Texas Instruments\BLE-CC254x-1.2\Documents\TI_BLE_Software_Developer's_Guide.pdf

Adopted *Bluetooth* specifications (which can be downloaded from https://www.bluetooth.org/Technical/Specifications/adopted.htm):

[2] Blood Pressure Profile (BLP) Specification v1.0

[3] Blood Pressure Service (BLS) Specification v1.0

[4] Health Thermometer Profile (HTP) Specification v1.0

[5] Health Thermometer Service (HTS) Specification v1.0

[6] Heart Rate Profile (HRP) Specification v1.0

[7] Heart Rate Service (HRS) Specification v1.0

[8] HID over GATT Profile (HOGP) Specification v1.0

[9] HID Service (HIDS) Specification v1.0

[10] Scan Parameters Profile (ScPP) v1.0

[11] Scan Parameters Service (ScPS) v1.0

[12] Device Information Service (DIS) Specification v1.1

[13] Battery Service (BAS) specification v1.0

[14] Proximity Profile (PXP) Specification v1.0

[15] Find Me Profile (FMP) Specification v1.0

[16] Link Loss Service (LLS) Specification v1.0

[17] Immediate Alert Service (IAS) Specification v1.0

[18] Tx Power Service (TPS) Specification v1.0

[19] Time Profile (TIP) Specification v1.0

[20] Alert Notification Profile (ANP) Specification v1.0

[21] Phone Alert Status (PASP) Specification v1.0

1 Overview

The purpose of this document is to give an overview of the sample applications that are included in the Texas Instruments CC2540/41 *Bluetooth*® low energy (BLE) software development kit. It is recommended that you read [1] before attempting to use these sample applications, as some knowledge of the CC2540/41 BLE protocol stack and software is required.

1.1 Introduction

Version 1.2 of the Texas Instruments CC2540/41 BLE software development kit includes several sample applications implementing a variety of GATT-based profiles. Some of these implementations are based on specifications that have been adopted by the *Bluetooth* Special Interest Group (BT SIG), while others are based on specifications that are a work-in-progress and have not been finalized. In addition, some applications are not based on any standardized profile being developed by the BT SIG, but rather are custom implementations developed by Texas Instruments. In order to interoperate with other *Bluetooth* low energy devices (such as a mobile phone), an application would need to be written on the other device which implements the proper GATT client and/or server functionality that matches the CC2540/41 sample application. The status of the implementation of each profile/application is included in this document.

The information in this guide specifically mentions only CC2540 projects; however all of the applications and configurations (with the exception of those that use the USB interface) also can run on the CC2541. Be sure to open the correct project file depending on the chipset that is being used.

2 Blood Pressure Sensor

This sample project implements the Blood Pressure profiles in a BLE peripheral device to provide an example blood pressure monitor using simulated measurement data. The application implements the "Sensor" role of the blood pressure profile. The project is based on the adopted profile and service specifications for Blood Pressure ([2] and [3]). The project also includes the Device Information Service ([12]).

The project can be opened with the following IAR workspace file:

C:\Texas Instruments\BLE-CC254x-1.2

Projects\ble\BLoodPressure\CC2540DB\bloodpressure.eww

2.1 Project Overview

The project structure is very similar to that of the SimpleBLEPeripheral project. The APP directory contains the application source code and header files. The project contains two configurations.

- **CC2540DK-MINI Keyfob Slave**: using the keyfob hardware platform.
- **CC2540 Slave:** using the SmartRF platform**.**

2.1.1 User Interface

There are two button inputs for this application.

KeyFob Right or SmartRF Joystick Right

When not connected, this button is used to toggle advertising on and off. When in a connection, this increases the value of various measurements.

KeyFob Left or SmartRF Joystick Up

This button cycle through different optional measurement formats.

2.1.2 Basic Operation

Power up the device and press the right button to enable advertising. From a blood pressure collector peer device, initiate a device discovery and connection procedure to discover and connect to the blood pressure sensor. The peer device should discover the blood pressure service and configure it to enable indication or notifications of the blood pressure measurement.

The peer device may also discover the device information service for more information such as mfg and serial number.

Once blood pressure measurements have been enabled the application will begin sending data to the peer containing simulated measurement values. Pressing the left button cycles through different data formats as follows:

- **MMHG | TIMESTAMP | PULSE | USER | STATUS**
- **MMHG | TIMESTAMP**
- **MMHG**
- **KPA**
- **KPA | TIMESTAMP**
- **KPA |TIMESTAMP | PULSE**

If the peer device initiates pairing, the blood pressure sensor will request a passcode. The passcode is “000000”.

Upon termination, the BPM will not begin to advertising again until the button is pressed.

The peer device may also query the blood pressure for read only device information. Further details on the supported items are listed in the GATT_DB excel sheet for this project. Examples are model number, serial number, etc.

2.2 Software Description

The application is implemented in the file **bloodpressure.c**.

2.2.1 Initialization

The initialization of the application occurs in two phases: first, the **Bloodpressure_Init** function is called by the OSAL. This function configures parameters in the peripheral profile, GAP, and GAP bond manager. It also sets up the blood pressure service along with standard GATT and GAP services in the attribute server. Then it sets an OSAL **START_DEVICE_EVT** event. This triggers the second phase of the initialization, which can be found within the **Bloodpressure_ProcessEvent** function. During this phase, the **GAPRole_StartDevice** function is called to set up the GAP functions of the application. Then **GAPBondMgr_Register** is called to register with the bond manager.

2.2.2 Event Processing

The application has two main event processing functions, **Bloodpressure_ProcessEvent** and **Bloodpressure_ProcessOSALMsg**.

Function **Bloodpressure_ProcessEvent** handles events as follows:

- **SYS_EVENT_MSG**: Service the OSAL queue and process OSAL messages.
- **BP_START_DEVICE_EVT**: Start the device, as described in the previous section.
- **BP_START_DISCOVERY_EVT**: Start discovery, search for time service on collector.
- **TIMER_BPMEAS_EVT**: Perform final measurement
- **BP_TIMER_CUFF_EVT**: Perform a cutoff measurement
- **BP_DISCONNECT_EVT**: Disconnect after sending measurement

Function **Bloodpressure_ProcessOSALMsg** handles OSAL messages as follows:

- **KEY_CHANGE**: Handle key presses.
- **GATT_MSG_EVENT**: This will handle reception of time information from collector.

2.2.3 Callbacks

The application callback functions are as follows:

- **PeripheralStateNotificationCB**: This is the GAP event callback. It processes GAP events for startup and link connect/disconnect.

- **bpServiceCB**: This is the blood pressure service callback. It handles enabling or disabling measurements.
- **TimeAppPairStateCB**: This is a GAPBOND callback to handle pairing states.
- **TimeAppPasscodeCB**: Returns the passcode of 0.

2.2.4 Sending Blood Pressure Measurement Indications

The application sends indication of the blood pressure measurement when configured to do so by the peer device.

When the peer device configures the blood pressure measurement for indication the application will receive a blood pressure service callback. The application starts a timer to begin periodic simulated blood pressure measurements. When the timer expires the application calls **bpSendStoredMeas** to build and send a measurement using the blood pressure service API. The application expects the peer device to send back an indication confirmation.

2.2.5 Sending Intermediate Measurement Notifications

The application sends notification of the blood pressure measurement when configured to do so by the peer device.

When the peer device configures the blood pressure measurement for notification the application will receive a blood pressure service callback. The application starts a timer to begin periodic simulated blood pressure measurements. When the timer expires the application calls **bloodPressureIMeasNotify** to build and send a measurement using the blood pressure service API.

2.2.6 Blood Pressure Measurement

	Flags	**Blood Pressure Measurement Value**			**Time Stamp**	**Pulse Rate**	**User ID**
		Systolic	**Diastolic**	**MAP**			
Size	1 octet	2 octets	2 octets	2 octets	7 octets	2 octets	1 octet

3 Health Thermometer

This sample project implements a Health Thermometer and Device Information profile in a BLE peripheral device to provide an example health thermometer application using simulated measurement data. The application implements the "Sensor" role of the Health Thermometer profile. The project is based on the adopted profile and service specifications for Health Thermometer (see [4] and [5]). The project also includes the Device Information Service ([12]).

The project can be opened with the following IAR workspace file:

C:\TexasInstruments\BLE-CC254x-1.2\Projects\ble\Thermometer\CC2540DB\thermometer.eww

3.1 Project Overview

The project structure is very similar to that of the SimpleBLEPeripheral project. The APP directory contains the application source code and header files. The project contains two configurations.

- **CC25.0DK-MINI Keyfob Slave**: using the keyfob hardware platform.
- **CC2540 Slave:** using the SmartRF platform**.**

3.1.1 User Interface

There are two button inputs for this application.

KeyFob Right | SmartRF Joystick Right

When not connected and not configured to take measurements, this button is used to toggle advertising on and off.

When in a connection or configured to take measurements, this increases the temperature by 1 degree Celsius. After 3 degrees in temperature rise, the interval will be set to 30 seconds and if configured, this will indicate to the peer an interval change initiated at the thermometer.

KeyFob Left | SmartRF Joystick Up

This button cycle through different measurement formats.

3.1.2 Basic Operation

Power up the device and press the right button to enable advertising. From a thermometer collector peer device, initiate a device discovery and connection procedure to discover and connect to the thermometer sensor. The peer device should discover the thermometer service and configure it to enable indication or notifications of the thermometer measurement. The peer device may also discover the device information service for more information such as mfg and serial number.

Once thermometer measurements have been enabled the application will begin sending data to the peer containing simulated measurement values. Pressing the left button cycles through different data formats as follows:

- **CELCIUS | TIMESTAMP | TYPE**
- **CELCIUS | TIMESTAMP**
- **CELCIUS**
- **FARENHEIT**
- **FARENHEIT | TIMESTAMP**
- **FARENHEIT | TIMESTAMP | TYPE**

If the peer device initiates pairing, the the HT will request a passcode. The passcode is “000000”.

The HT operates in the following states:

- **Idle –** In this state, the thermometer will wait for the right button to be pressed to start advertising.
- **Idle Configured** – The thermometer waits the interval before taking a measurement and proceeding to Idle Measurement Ready state**.**
- **Idle Measurement Ready** – The thermometer has a measurement ready and will advertise to allow connection. The thermometer will periodically advertise in this state.
- **Connected Not Configured** - The thermometer may be configured to enable measurement reports. The thermometer will not send stored measurements until the CCC is enabled. Once connection is established, the thermometer sets a timer to disconnect in 20 seconds.
- **Connected Configured** - The thermometer will send any stored measurements if CCC is set to send measurement indications.
- **Connected Bonded** - The thermometer will send any stored measurements if CCC was previously set to send measurement indications.

The peer device may also query the thermometers read only device information. Examples are model number, serial number, etc.

3.2 Software Description

The application is implemented in the file **thermometer.c**.

3.2.1 Initialization

The initialization of the application occurs in two phases: first, the **Thermometer_Init** function is called by the OSAL. This function configures parameters in the peripheral profile, GAP, and GAP bond manager. It also sets up the thermometer service along with standard GATT and GAP services in the attribute server. Then it sets an OSAL **START_DEVICE_EVT** event. This triggers the second phase of the initialization, which can be found within the **Thermometer_ProcessEvent** function. During this phase, the **GAPRole_StartDevice** function is

called to set up the GAP functions of the application. Then **GAPBondMgr_Register** is called to register with the bond manager.

3.2.2 Event Processing

The application has two main event processing functions, **Thermometer_ProcessEvent** and **Thermometer_ProcessOSALMsg**.

Function **Thermometer_ProcessEvent** handles events as follows:

- **SYS_EVENT_MSG**: Service the OSAL queue and process OSAL messages.
- **START_DEVICE_EVT**: Start the device, as described in the previous section.
- **TH_START_DISCOVERY_EVT**: Start discovery, search for time service on collector.
- **TH_PERIODIC_MEAS_EVT**: Start a measurement.
- **TH_PERIODIC_IMEAS_EVT**: Send immediate measurement.
- **TH_DISCONNECT_EVT**: Terminate connection.

Function **Thermometer_ProcessOSALMsg** handles OSAL messages as follows:

- **KEY_CHANGE**: Handle key presses.
- **GATT_MSG_EVENT**: This will handle reception of time information from collector.

3.2.3 Callbacks

The application callback functions are as follows:

- **PeripheralStateNotificationCB**: This is the GAP event callback. It processes GAP events for startup and link connect/disconnect.
- **ThermometerCB**: This is the thermometer service callback. It handles enabling or disabling measurements.
- **TimeAppPairStateCB**: This is a GAPBOND callback to handle pairing states.
- **TimeAppPasscodeCB**: Returns the passcode of 0.

3.2.4 Sending Temperature Indications

The application enables indication of the thermometer measurement when configured to do so by the peer device.

When the peer device configures the thermometer measurement for indication the application will receive a thermometer service callback. The application starts a timer to begin periodic simulated thermometer measurements. When the timer expires the application calls **thermometerMeasIndicate** to build and store a measurement. Once a measurement is ready, the thermometer will enter connectable state and send advertisements. If the peer device connects and the CCC is enabled, the thermometer will send the stored measurements. The thermometer expects the peer device to send back an indication confirmation for each indication sent.

3.2.5 Sending Intermediate Measurement Notifications

The application sends notification of the thermometer measurement when configured to do so by the peer device.

When the peer device configures the thermometer measurement for notification the application will receive a thermometer service callback. The application starts a timer to begin periodic simulated thermometer measurements. When the timer expires the application calls **thermometerIMeasIndicate** to build and send a measurement using the thermometer service API.

3.2.6 Sending Interval Change Indications

If the CCC for interval change is enabled, the thermometer will send an indication to the peer if the interval is changed by the thermometer. This can be triggered by pressing the right button three times which will increase the simulated temperature by 3 degrees and also reset the interval to 30 seconds.

3.2.7 Thermometer Measurement Format

	Flags	Temperature Measurement Value	Time Stamp (if present)	Temperature Type (if present)
Size	1 octet	4 octets	0 or 7 octets	0 or 1 octet
Units	None	Based on bit 0 of Flags field	Smallest unit in seconds	None

4 Heart Rate Sensor

This sample project implements the Heart Rate and Battery profiles in a BLE peripheral device to provide an example heart rate sensor using simulated measurement data. The application implements the "Sensor" role of the Heart Rate profile and the "Battery Reporter" role of the Battery profile. The project is based on adopted profile and service specifications for Health Rate ([6] and [7]). The project also includes the Device Information Service ([12]).

The project can be opened with the following IAR workspace file:

C:\Texas Instruments\BLE-CC254x-1.2\Projects\ble\HeartRate\CC2540DB\heartrate.eww

4.1 Project Overview

The project structure is very similar to that of the SimpleBLEPeripheral project. The APP directory contains the application source code and header files. The project contains one configuration, **CC2540DK-MINI Keyfob Slave**, using the keyfob hardware platform.

4.1.1 User Interface

When not connected, the keyfob's right button is used to toggle advertising on and off. When in a connection, the keyfob's left button cycles through different heart rate sensor data formats and the right button sends a battery level-state notification.

4.1.2 Basic Operation

Power up the device and press the right button to enable advertising. From a heart rate collector peer device, initiate a device discovery and connection procedure to discover and connect to the heart rate sensor. The peer device should discover the heart rate service and configure it to enable notifications of the heart rate measurement. The peer device may also discover and configure the battery service for battery level-state notifications.

Once heart rate measurement notifications have been enabled the application will begin sending data to the peer containing simulated measurement values. Pressing the left button cycles through different data formats as follows:

- Sensor contact not supported.
- Sensor contact not detected.
- Sensor contact and energy expended set.
- Sensor contact and R-R Interval set.
- Sensor contact, energy expended, and R-R Interval set.
- Sensor contact, energy expended, R-R Interval, and UINT16 heart rate set.
- Nothing set.

If the peer device initiates pairing then the devices will pair. Only "just works" pairing is supported by the application (pairing without a passcode).

The application advertises using either a fast interval or a slow interval. When advertising is initiated by a button press or when a connection is terminated due to link loss, the application will start advertising at the fast interval for 30 seconds followed by the slow interval. When a connection is terminated for any other reason the application will start advertising at the slow interval. The advertising intervals and durations are configurable in file **heartrate.c**.

4.2 Software Description

The application is implemented in the file **heartrate.c**.

4.2.1 Initialization

The initialization of the application occurs in two phases: first, the **HeartRate_Init** function is called by the OSAL. This function configures parameters in the peripheral profile, GAP, and GAP bond manager. It also sets up the heart rate service and the battery service along with standard GATT and GAP services in the attribute server. Then it sets an OSAL **START_DEVICE_EVT** event. This triggers the second phase of the initialization, which can be found within the **HeartRate_ProcessEvent** function. During this phase, the **GAPRole_StartDevice** function is called to set up the GAP functions of the application. Then **GAPBondMgr_Register** is called to register with the bond manager.

4.2.2 Event Processing

The application has two main event processing functions, **HeartRate_ProcessEvent** and **HeartRate_ProcessOSALMsg**.

Function **HeartRate_ProcessEvent** handles events as follows:

- **SYS_EVENT_MSG**: Service the OSAL queue and process OSAL messages.
- **START_DEVICE_EVT**: Start the device, as described in the previous section.
- **HEART_PERIODIC_EVT**: Send periodic heart rate measurements.
- **BATT_PERIODIC_EVT**: Check the battery level and send notification if it changed.

Function **HeartRate_ProcessOSALMsg** handles OSAL messages as follows:

- **KEY_CHANGE** messages: Call function **HeartRate_HandleKeys** to handle key presses.

4.2.3 Callbacks

The application callback functions are as follows:

- **HeartRateGapStateCB**: This is the GAP event callback. It processes GAP events for startup and link connect/disconnect.
- **HeartRateCB**: This is the heart rate service callback. It handles enabling or disabling periodic heart rate measurements when notifications of the heart rate measurement are enabled or disabled.
- **HeartRateBattCB**: This is the battery service callback. It handles enabling or disabling periodic battery measurements when notifications of the battery level-state are enabled or disabled.

4.2.4 Sending Notifications

The application sends notifications of the heart rate measurement and the battery level-state when configured to do so by the peer device.

When the peer device configures the heart rate measurement for notification the application will receive a heart rate service callback. The application starts a timer to begin periodic simulated heart rate measurements. When the timer expires the application calls **heartRateMeasNotify** to build and send a measurement using the heat rate service API.

When the peer device configures the battery level-state for notification the application will receive a battery service callback. The application starts a timer to periodically measure the battery level. When the timer expires the application calls battery service API function **Batt_MeasLevel** to measure the battery level using the CC2450 ADC. Notification of the battery level-state is handled inside the battery service; if the battery level has dropped since the previous measurement a notification is sent.

5 HID Emulated Keyboard

This sample project implements the HID Over GATT profile in a BLE peripheral device to provide an example of how a HID keyboard can be emulated with a simple two button remote control device. The project is based on adopted profile and service specifications for HID over GATT ([8]

and [9]) and Scan Parameters ([10] and [11]). The project also includes the Device Information Service ([12]) and Battery Service ([13]).

The project can be opened with the following IAR workspace file:

C:\TexasInstruments\BLE-CC254x-1.2\Projects\ble\HIDEmuKbd\CC2540DB\HidEmuKbd.eww

5.1 Project Overview

The project structure is very similar to that of the SimpleBLEPeripheral project. The APP directory contains the application source code and header files.

The project contains one configuration, **CC2540DK-MINI Keyfob Slave**, using the keyfob hardware platform.

5.1.1 User Interface

When not connected and not already advertising, pressing either button will initiate advertising. When in a connection, the keyfob's left button sends a “left arrow” key and the right button sends a “right arrow” key.

Note that a secure connection must be established before key presses will be sent to the peer device.

5.1.2 Basic Operation

Power up the device and press either button to enable advertising. From a HID Host peer device, initiate a device discovery and connection procedure to discover and connect to the HID device. The peer device should discover the HID service and recognize the device as a keyboard. The peer device may also discover and configure the battery service for battery level-state notifications.

By default the HID device requires security and uses “just works” pairing. After a secure connection is established and the HID host configures the HID service to enable notifications, the HID device can send HID key presses to the HID host. A notification is sent when a button is pressed and when a button is released.

The HID host can send keyboard LED information to the device to illuminate the keyfob LEDs. The “caps lock” setting controls the green LED and the “num lock” setting controls the red LED.

If there is no HID activity for a period of time (20 seconds by default) the HID device will disconnect. When the connection is closed the HID device will not advertise. Press either button to enable advertising and connect again.

5.2 Software Description

The project uses the following services and profiles:

- Battery service (battservice.c and battservice.h).
- Device Information service (devinfoservice.c and devinfoservice.h).
- Scan Parameters service (scanparamservice.c and scanparamservice.h).
- HID service for keyboard (hidkbdservice.c hidkbdservice.h).
- HID device profile (hiddev.c and hiddev.h). This is a common profile for HID devices that performs the following procedures:
    - Advertising, connection procedures, and security procedures.
    - Sending HID notifications.
    - Handling read and write of HID service characteristics.

5.3 HidEmuKbd Application

The application is implemented in the file **hidemukbd.c**.

5.3.1 Initialization

The **HidEmuKbd_Init** function is called by the OSAL to perform task initialization procedures. This function sets parameters for the peripheral profile, GAP bond manager, and Battery service. The function also registers the HID keyboard service and HID device profile.

5.3.2 Event Processing

The application has two main event processing functions, **HidEmuKbd_ProcessEvent** and **HidEmuKbd_ProcessOSALMsg**.

Function **HidEmuKbd_ProcessEvent** handles the **SYS_EVENT_MSG** event, which services the OSAL queue and processes OSAL messages.

Function **HidEmuKbd_ProcessOSALMsg** handles OSAL messages as follows:

- **KEY_CHANGE** messages: Call function **HidEmuKbd_HandleKeys** to handle keypresses.

5.3.3 Callbacks

The application callback functions are as follows:

- **hidEmuKbdRptCB**: This is the HID device report callback. It processes HID reports received from the HID host.
- **hidEmuKbdEvtCB**: This is the HID device event callback. It handles HID events, such as enter/exit suspend or enter/exit boot mode.

5.3.4 Sending Notifications

The application sends notifications containing HID keypress data when a button is pressed. This is done in function **HidEmuKbd_HandleKeys** by calling HID device profile function **HidDev_Report**. The details of sending notifications are handled in the HID device profile.

5.4 HID Device Profile

The HID device profile is implemented in the file **hiddev.c**.

5.4.1 Initialization

The initialization of occurs in two phases: first, the **HidDev_Init** function is called by OSAL. This function sets up the battery, device information, and scan parameters services along with standard GATT and GAP services in the attribute server. Then it sets an OSAL **START_DEVICE_EVT** event. This triggers the second phase of the initialization, which can be found within the **HidDev_ProcessEvent** function. During this phase, the **GAPRole_StartDevice** function is called to set up the GAP functions of the application. Then **GAPBondMgr_Register** is called to register with the bond manager.

5.4.2 Event Processing

The application has two main event processing functions, **HidDev_ProcessEvent** and **HidDev_ProcessOSALMsg**.

Function **HidDev_ProcessEvent** handles events as follows:

- **SYS_EVENT_MSG**: Service the OSAL queue and process OSAL messages.
- **START_DEVICE_EVT**: Start the device, as described in the previous section.
- **HID_IDLE_EVT**: Terminate the connection if idle.
- **BATT_PERIODIC_EVT**: Check the battery level and send notification if it changed.

Function **HidDev_ProcessOSALMsg** handles OSAL messages as follows:

- **GATT_MSG_EVENT** messages: Call function **hidDevProcessGattMsg** to process GATT messages.

5.4.3 Callbacks

The HID device profile callback functions are as follows:

- **HidDevGapStateCB**: This is the GAP event callback. It processes GAP events for startup and link connect/disconnect.
- **hidDevPairStateCB**: This is the pairing state callback. Handle pairing events.
- **hidDevPasscodeCB**: This is the passcode callback. Send a passcode response.
- **HidDevBattCB**: This is the battery service callback. It handles enabling or disabling periodic battery measurements when notifications of the battery level-state are enabled or disabled.
- **hidDevScanParamCB**: This is the scan parameters service callback. Handle a scan parameters service event.

5.4.4 GATT Read and Write Callbacks

The HID device profile GATT read and write callbacks, **HidDev_WriteAttrCB** and **HidDev_ReadAttrCB** handle reading and writing of all HID characteristics. These functions are used by a HID service when registering the service with GATT.

5.4.5 Mapping HID Reports to HID Characteristics

A HID service defines one or more HID reports in its service that are used to send and receive HID data. The HID device profile has a table that maps HID reports to HID characteristics. The table is constructed by the HID service using the HID device profile and must be registered with the HID device profile by calling function **HidDev_RegisterReports**.

5.4.6 Sending and Receiving HID Reports

The application calls function **HidDev_Report** to send a HID report. The HID device sends HID report notifications to the HID host when configured to do so. When notifications are enabled or disabled for a HID report characteristic the HID report callback is executed with event HID_DEV_OPER_ENABLE or HID_DEV_OPER_DISABLE.

A HID report is received from the HID host is either a read event or write event. The HID report callback is executed with event HID_DEV_OPER_READ or HID_DEV_OPER_WRITE.

5.4.7 Advertising and Connection Procedures

The HID device profile manages advertising and connection procedures. The device will start advertising if the **HidDev_Report** function is called when not connected and not already advertising.

Advertising is performed at an “initial” rate when not bonded, and at a “low” or “high” rate if bonded. The advertising intervals and durations for the different rates are configurable.

If the device is bonded the device will advertise at the high rate when reconnecting to send a HID report. If the connection is terminated and the device is bonded and the flags are set to HID_FLAGS_NORMALLY_CONNECTABLE the device will advertise at the low rate. Otherwise the device will not advertise when disconnected until it has data to send.

If no HID data is sent or received within an idle timeout period the HID device profile will terminate the connection, unless pairing is in progress. The idle timeout is configured by the application and can be disabled by setting it to zero.

6 HostTestRelease- BLE Network Processor

The HostTestRelease project implements a BLE network processor, for use with an external microcontroller or a PC software application such as BTool. More information on the HostTestRelease project can be found in [1].

7 KeyFobDemo

The KeyFobDemo application will demonstrate the following.

- Report battery level
- Report 3 axis accelerometer readings.
- Report proximity changes
- Report key press changes

The following GATT services are used:

- Device Information (see [12])
- Link Loss (for Proximity Profile, Reporter role; see [14] and [16])
- Immediate Alert (for Proximity Profile, Reporter role and Find Me Profile, Target role; see [14], [15], and [17])
- Tx Power (for Proximity Profile, Reporter role; see [18])
- Battery (see [13])
- Accelerometer
- SimpleKeys

The accelerometer and simple keys profiles are not aligned to official SIG profiles, but rather serve as an example of profile service implementation. The device information service and proximity-related services are based on adopted specifications.

7.1 Project Overview

The project structure is very similar to that of the SimpleBLEPeripheral project. The APP directory contains the application source code and header files. The project supports the following configurations.

- **CC2540DK-MINI Keyfob Slave**: using the keyfob hardware platform.

7.1.1 User Interface

There are two button inputs for this application, an LED, and buzzer.

Right Button

When not connected, this button is used to toggle advertising on and off. When in a connection, this will register a key press which may be enabled to notify a peer device or may be read by a peer device.

Left Button

When in a connection, this will register a key press which may be enabled to notify a peer device or may be read by a peer device.

LED

Flash when Link Loss Alert is triggered.

Buzzer

The buzzer turns on if a Link Loss Alert is triggered.

7.1.2 Battery Operation

They KeyFob used an ADC to read remaining battery level. The battery profile allows for the USB Dongle to read the percentage of battery remaining on the keyfob by reading the value of < BATTERY_LEVEL_UUID>

7.1.3 Accelerometer Operation

The keyfob uses SPI to interface to a 3 axis accelerometer on the KeyFobDemo. The accelerometer must be enabled < ACCEL_ENABLER_UUID> by writing a value of “01”. Once the accelerometer is enabled, each axis can be configured to send notifications by writing “01 00” to the characteristic configuration for each axis < GATT_CLIENT_CHAR_CFG_UUID>. In addition, the values can be read by reading <ACCEL_X_UUID>, <ACCEL_Y_UUID>,<ACCEL_Z_UUID>.

7.1.4 Keys

The simple keys service on the keyfob allows the device to send notifications of key presses and releases to a central device. The application registers with HAL to receive a callback in case HAL detects a key change.

The peer device may read the value of the keys by reading <SK_KEYPRESSED_UUID>.

The peer device may enable key press notifications by writing a “01” to <GATT_CLIENT_CHAR_CFG_UUID>.

A value of “00” indicates that neither key is pressed. A value of “01” indicates that the left key is pressed. A value of “02” indicates that the right key is pressed. A value of “03” indicates that both keys are pressed.

7.1.5 Proximity

One of the services of the proximity profile is the link loss service, which allows the proximity reporter to begin an alert in the event the connection drops.

The link loss alert can be set by writing a value to <PROXIMITY_ALERT_LEVEL_UUID>.

The default alert value setting is “00”, which indicates “no alert.” To turn on the alert, write a 1-byte value of “01” (low alert) or “02” (high alert). By default, the link does not timeout until 20 seconds have gone by without receiving a packet. This “Supervision Timeout” value can be changed in the “Connection Services” tab; however the timeout value must be set before the connection is established. After completing the write, move the keyfob device far enough away from the USB Dongle until the link drops. Alternatively, you can disconnect the USB Dongle from the PC, effectively dropping the connection. Once the timeout on the keyfob expires, the alarm will be triggered. If a low alet was set, the keyfob will make a low pitched beep. If a high alert was set, the keyfob will make a high pitched beep and the LED will blink. In either case, the keyfob will beep ten times and then stop. Alternatively to stop the beeping, either a new connection can be formed with the keyfob, or the button can be pressed.

7.2 Software Description

The application is implemented in the file **keyFobDemo.c**.

7.2.1 Initialization

The initialization of the application occurs in two phases: first, the **KeyFobApp_Init** function is called by the OSAL. This function configures parameters in the peripheral profile, GAP, and GAP bond manager. It also sets up the KeyFobDemo example services along with standard GATT and GAP services in the attribute server. Then it sets an OSAL **START_DEVICE_EVT** event. This triggers the second phase of the initialization, which can be found within the **KeyFobApp_ProcessEvent** function. During this phase, the **GAPRole_StartDevice** function is called to set up the GAP functions of the application. Then **GAPBondMgr_Register** is called to register with the bond manager.

7.2.2 Event Processing

The application has two main event processing functions, **KeyFobApp_ProcessEvent** and **KeyFobApp_ProcessOSALMsg**.

Function **KeyFobApp_ProcessEvent** handles events as follows:

- **SYS_EVENT_MSG**: Service the OSAL queue and process OSAL messages.
- **KFD_START_DEVICE_EVT**: Start the device, as described in the previous section.
- **KFD_ACCEL_READ_EVT**: Read accelerometer and set timer for periodic reads.
- **KFD_BATTERY_CHECK_EVT**: Read battery level and set timer for periodic reads.

- **KFD_TOGGLE_BUZZER_EVT**: Toggle buzzer on proximity state.

Function **KeyFobApp_ProcessOSALMsg** handles OSAL messages as follows:

- **KEY_CHANGE**: Handle key presses.

7.2.3 Callbacks

The application callback functions are as follows:

- **PeripheralStateNotificationCB**: This is the GAP event callback. It processes GAP events for startup and link connect/disconnect.
- **ProximityAttrCB**: Receive info on link loss and setup from proximity service.
- **AccelEnablerChangeCB**: Handle enabling of accelerometer.

8 SimpleBLECentral

The SimpleBLECentral project implements a very simple BLE central device with GATT client functionality. It makes use of the SmartRF05 + CC2540EM hardware platform. This project can be used as a framework for developing many different central-role applications. More information on the SimpleBLECentral project can be found in [1].

9 SimpleBLEPeripheral

The SimpleBLEPeripheral project implements a very simple BLE peripheral device with GATT services, including configurations for the CC2540DK-MINI keyfob as well as the SmartRF05 + CC2540EM hardware platforms. This project can be used as a framework for developing many different peripheral-role applications. More information on the SimpleBLEPeripheral project can be found in [1].

10 TimeApp- BLE Watch

This sample project implements time and alert-related profiles in a BLE peripheral device to provide an example of how Bluetooth LE profiles are used in a product like a watch. The project is based on adopted profile specifications for Time ([19]), Alert Notification ([20]), and Phone Alert Status ([21]). All profiles are implemented in the Client role. In addition, the following Network Availability Profile, Network Monitor role has been implemented, based on Network Availability Draft Specification d05r04 (UCRDD).

The project can be opened with the following IAR workspace file:

C:\Texas Instruments\BLE-CC254x-1.2\Projects\ble\TimeApp\CC2540\TimeApp.eww

10.1 Project Overview

The TimeApp project structure is very similar to that of the SimpleBLEPeripheral project. The APP directory contains the application source code and header files. The project contains one configuration, **CC2540EM Slave**, using the SmartRF05EB + CC2540EM hardware platform.

10.1.1 User Interface

The SmartRF05EB joystick and display provide a user interface for the application. The joystick and buttons are used as follows:

- Joystick Up: Start or stop advertising.
- Joystick Left: If connected, send a command to the Alert Notification control point.
- Joystick Center: If connected, disconnect. If held down on power-up, erase all bonds.
- Joystick Right: If connected, initiate a Reference Time update.
- Joystick Down: If connected, initiates a Ringer Control Point update

The LCD display is used to display the following information:

- Device BD address.
- Connection state.

- Pairing and bonding status.
- Passcode display.
- Time and date.
- Network availability.
- Battery state of peer device.
- Alert notification messages.
- Unread message alerts.
- Ringer status.

10.1.2 Basic Operation

When the application powers up it displays "Time App", the BD address of the device, and a default time and date of "00:00 Jan01 2000". To connect, press Joystick Up to start advertising then initiate a connection from a peer device. The connection status will be displayed. Once connected, the application will attempt to discover the following services on the peer device:

- Current Time Service
- DST Change Service
- Reference Time Service
- Alert Notification Service
- Phone Alert Status Service
- Network Availability Service
- Battery Service

The discovery procedure will cache handles of interest. When bonded to a peer device, the handles are saved so that the discovery procedure is not performed on subsequent connections.

If a service is discovered certain service characteristics are read and displayed. The network availability status and battery level will be displayed and the current time will be updated.

The application also enables notification or indication for characteristics that support these operations. This allows the peer device to send notifications or indications updating the time, network availability, or battery status. The peer device can also send alert notification messages and unread message alerts. These updates and messages will be displayed on the LCD.

The peer device may initiate pairing. If a passcode is required the application will generate and display a random passcode. Enter this passcode on the peer device to proceed with pairing.

The application advertises using either a fast interval or a slow interval. When advertising is initiated by a button press or when a connection is terminated due to link loss, the application will start advertising at the fast interval for 30 seconds followed by the slow interval. When a connection is terminated for any other reason the application will start advertising at the slow interval. The advertising intervals and durations are configurable in file **timeapp.c**.

10.2 Software Description

The TimeApp application is implemented in the following files:

- **timeapp.c**: Main initialization, event handling and callback functions.
- **timeapp_clock.c**: Clock timekeeping and display functions.
- **timeapp_config.c**: Characteristic configuration functions.
- **timeapp_discovery.c**: Service discovery functions.
- **timeapp_ind.c**: Indication and notification handling functions.

10.2.1 Initialization

The initialization of the application occurs in two phases: first, the **TimeApp_Init** function is called by the OSAL. This function configures parameters in the peripheral profile, GAP, and GAP bond manager and also initializes GATT for client operation. It also sets up standard GATT and GAP services in the attribute server. Then it sets an OSAL **START_DEVICE_EVT** event. This triggers the second phase of the initialization, which can be found within the **TimeApp_ProcessEvent**

function. During this phase, the **GAPRole_StartDevice** function is called to set up the GAP functions of the application. Then **GAPBondMgr_Register** is called to register with the bond manager.

10.2.2 Event Processing

The application has two main event processing functions, **TimeApp_ProcessEvent** and **timeApp_ProcessOSALMsg**.

Function **TimeApp_ProcessEvent** handles events as follows:

- **SYS_EVENT_MSG**: Service the OSAL queue and process OSAL messages.
- **START_DEVICE_EVT**: Start the device, as described in the previous section.
- **START_DISCOVERY_EVT**: Start service discovery.
- **CLOCK_UPDATE_EVT**: Update the clock display.

Function **timeApp_ProcessOSALMsg** handles OSAL messages as follows:

- **KEY_CHANGE** messages: Call function timeApp_HandleKeys to handle keypresses.
- **GATT_MSG_EVENT** messages: Call function timeAppProcessGATTMsg to handle messages from GATT.

10.2.3 Callbacks

The application callback functions are as follows:

- **timeAppGapStateCB**: This is the GAP event callback. It processes GAP events for startup and link connect/disconnect.
- **timeAppPairStateCB**: This is the GAP bond manager state callback. It displays the status of pairing and bonding operations.
- **timeAppPasscodeCB**: This is the GAP bond manager passcode callback. It generates and displays a passcode.

10.2.4 Service Discovery

The application performs service discovery for several Bluetooth LE services. Discovery is initiated when a connection is established to a peer device with which there is no existing bond. Discovery starts when the discovery delay timer expires, which sets event **START_DISCOVERY_EVT**. This will result in execution of function **timeAppDiscStart**. However if a pairing procedure is in progress when the timer expires discovery will be postponed until pairing completes. This is done in case the peer device requires security before its characteristics are read or written.

A service discovery procedure is performed for each service until discovery has been attempted on all services of interest. The service discovery procedure is generalized as follows:

1. Discovery the service by UUID.
2. If found, discover all characteristics of the service. Cache the handles of characteristics of interest.
3. If a discovered characteristic uses a client characteristic configuration descriptor (abbreviated as CCCD in the code), discover all descriptors of the characteristic.

If the mandatory characteristics and descriptors of the service are discovered then discovery of the service is deemed successful.

The handles of discovered characteristics of interest are stored in array **timeAppHdlCache**.

The main service discovery function is **timeAppDiscGattMsg**. This function is executed when a discovery-related GATT message response is received. This function then executes a separate discovery function for each service. The service discovery state is maintained in variable **timeAppDiscState**.

10.2.5 Service Configuration

When service discovery completes the service configuration procedure is initiated. This procedure reads and writes characteristics of interest in the discovered services.

The main service configuration function is **timeAppConfigNext**. This function searches the cached handle array for the next characteristic of interest, and once found it performs a read or write on that characteristic.

When a GATT read response or write response is received, function **timeAppConfigGattMsg** is called. This function processes the received response and performs an action, such as updating the clock display, and then calls **timeAppConfigNext** to initiate the next read or write.

The application writes all discovered client characteristic configuration descriptors to enable notification or indication. The application also reads some characteristics and then performs no action with the received response. This is done simply for testing and demonstration.

10.2.6 Handling Indications and Notifications

Handling of received indications and notifications is performed by function **timeAppIndGattMsg**. This function is called when a GATT indication or notification message is received. The function will process the data in the received message and display it on the LCD.

10.2.7 Clock Time

The application uses the OSAL Clock service to update and maintain the clock time. When new date and time data is received from the peer device, function **timeAppClockSet** is called to update the time in OSAL and display the updated time on the LCD. The LCD is periodically updated by an OSAL timer that sets event **CLOCK_UPDATE_EVT**.

11 Serial Bootloader

This sample allows the user load an image over the UART0-Alt1 port.

C:\Texas Instruments\BLE-CC254x-1.2\Projects\ble\SBL\iar\cc254x\sbl.eww

11.1 Basic Operation

The SBL is a utility application allowing the user download an image over the serial port. This might be useful for field updates or allowing an external MCU to change firmware without using the CC Debugger. The example provided in this release is specific to UART0-Alt1, although source is provided if modification to settings is required.

11.1.1 Build and Flash the SBL

Build the sbl project and flash it to CC254X.

11.1.2 Build the Project to be Bootloaded

This release contains an example project which creates a bootload compatible image. The example project is located at

C:\TexasInstruments\BLE-CC254x-1.2\Projects\ble\HostTestApp\CC2540\ HostTestRelease.ewp

Select the CC2540-EM-SBL project configuration, and build the project. The .bin is located at

C:\TexasInstruments\BLE-CC254x-1.2\Projects\ble\HostTestApp\CC2540\CC2540EM-SBL\Exe\HostTestReleaseCC2540-SBL.bin

11.1.3 Download the User Project Image (.bin)

TI provides a PC tool to download the image. The latest version can be found on our wiki at *http://processors.wiki.ti.com/index.php/Category:BluetoothLE .*

To re-bootload send the HCI_EXT_UTIL_FORCE_BOOT command, to put a device back in bootloader mode.

12 USB Bootloader

This sample application allows the user load an image over the USB port.

12.1 Basic Operation

The UBL is a utility application allowing the user to download an image to the USB dongle by drag and drop in Microsoft Windows. This would be useful for updating the firmware on a USB dongle when no programming pins are available. The example provided in this release is specific to USB dongle and Nano dongle.

12.1.1 Flash UBL

The UBL is provided as a hex file. Some source is provided, but this is only for reference. Flash the image using the TI Flash Programmer.

For USB Dongle provided with CC2540DK-mini kit use

C:\Texas Instruments\BLE-CC254x-1.2\Projects\ble\UBL\soc_8051\usb_msd\bin\ ubl_cc2540-dk.hex

For the Nano dongle use

C:\Texas Instruments\BLE-CC254x-1.2\Projects\ble\UBL\soc_8051\usb_msd\bin\ ubl_cc2540-nano.hex

12.1.2 Build the Project to be Bootloaded

This release contains an example project which creates a bootload compatible image. The example project is located at

C:\Texas Instruments\BLE-CC254x-1.2\Projects\ble\HostTestApp\CC2540\HostTestRelease.ewp

Select the CC2540USB-UBL project configuration, and build the project. The .bin is located at

C:\TexasInstruments\BLE-CC254x-1.2\Projects\ble\HostTestApp\CC2540\CC2540USB-UBL\Exe\ HostTestReleaseCC2540USB-UBL.bin

12.1.3 Download the User Project Image (.bin)

Once the bootloader has been flashed, the USB dongle will show up as a mass storage device. Use Microsoft Windows explorer to drag and drop the HostTestReleaseCC2540USB-UBL.bin to the mass storage. Once it is copies over, the dongle will change and register with Windows as a virtual com port. Press and hold the USB dongle button furthest from port while inserting the dongle to run the bootloader again. The nano dongle will enter mass storage on insertion, but will stay in that mode for a short amount of time only.

13 General Information

13.1 Document History

Table 1: Document History

Revision	Date	Description/Changes
1.0	2011-07-13	Initial release, documenting sample applications included in BLEv1.1 release
1.2	2012-02-07	Updated to align with BLEv1.2 release sample applications.

14 Address Information

Texas Instruments Norway AS
Gaustadalléen 21
N-0349 Oslo
NORWAY
Tel: +47 22 95 85 44
Fax: +47 22 95 85 46
Web site: http://www.ti.com/lpw

15 TI Worldwide Technical Support

Internet

TI Semiconductor Product Information Center Home Page: support.ti.com
TI Semiconductor KnowledgeBase Home Page: support.ti.com/sc/knowledgebase

TI LPRF forum E2E community http://www.ti.com/lprf-forum

Product Information Centers

Americas

Phone: +1(972) 644-5580
Fax: +1(972) 927-6377
Internet/Email: support.ti.com/sc/pic/americas.htm

Europe, Middle East and Africa

Phone:
Belgium (English) +32 (0) 27 45 54 32
Finland (English) +358 (0) 9 25173948
France +33 (0) 1 30 70 11 64
Germany +49 (0) 8161 80 33 11
Israel (English) 180 949 0107
Italy 800 79 11 37
Netherlands (English) +31 (0) 546 87 95 45
Russia +7 (0) 95 363 4824
Spain +34 902 35 40 28
Sweden (English) +46 (0) 8587 555 22
United Kingdom +44 (0) 1604 66 33 99
Fax: +49 (0) 8161 80 2045
Internet: support.ti.com/sc/pic/euro.htm

Japan

Fax	International	+81-3-3344-5317
	Domestic	0120-81-0036
Internet/Email	International	support.ti.com/sc/pic/japan.htm
	Domestic	www.tij.co.jp/pic

Asia

Phone	International	+886-2-23786800
	Domestic	Toll-Free Number
	Australia	1-800-999-084
	China	800-820-8682
	Hong Kon	800-96-5941
	India	+91-80-51381665 (Toll)
	Indonesia	001-803-8861-1006
	Korea	080-551-2804
	Malaysia	1-800-80-3973
	New Zealand	0800-446-934
	Philippines	1-800-765-7404
	Singapore	800-886-1028
	Taiwan	0800-006800
	Thailand	001-800-886-0010
Fax		+886-2-2378-6808
Email		tiasia@ti.com or ti-china@ti.com
Internet		support.ti.com/sc/pic/asia.htm

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		Applications	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265