Texas Instruments CC2540 *Bluetooth*® Low Energy Sample Applications Guide v1.1a

Document Number: SWRU297A

Table Of Contents

References

Included with Texas Instruments *Bluetooth* Low Energy v1.1 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-CC2540-1.1\):

[1] Texas Instruments *Bluetooth*® Low Energy Software Developer's Guide (SWRU271A)

C:\Texas Instruments\BLE-CC2540-1.1\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] Health Thermometer Profile (HTP) Specification v1.0

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

[4] Device Information Service (DIS) Specification v1.0

[5] Device Information Service (DIS) Specification v1.0

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

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

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

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

[10] Tx Power Service (TPS) 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 *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 BLE protocol stack and software is required.

1.1 Introduction

Version 1.1 of the Texas Instruments CC2540 BLE software development kit includes several new 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 and would not have any interoperability with other *Bluetooth* devices. The status of the implementation of each profile/application is included in this document.

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 following specifications:

- Blood Pressure Profile

  Based on Blood Pressure Profile Draft Specification d09r04

- Blood Pressure Service

  Based on Blood Pressure Service Draft Specification d09r05

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

C:\Texas Instruments\BLE-CC2540-1.1

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**
- **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 begin advertising again.

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.
- **START_DEVICE_EVT**: Start the device, as described in the previous section.
- **TH_START_DISCOVERY_EVT**: Start discovery, search for time service on collector.
- **PERIODIC_EVT**: Send periodic BPM measurements.
- **PERIODIC_IMEAS_EVT**: Send intermediate BPM measurements.

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.
- **BloodpressureCB**: 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 **bloodPressureMeasIndicate** 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 Measurent

	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 Emulated Keyboard

This project implements a two-button BLE “keyboard” with the CC2540DK-MINI keyfob acting as a BLE peripheral device. When used in conjunction with the HIDDongle project as a BLE central device (see section 6) this application allows you to actually type text or enter commands on any PC with a USB port, as long as the operating system supports the USB HID (Human Interface Device) profile.

It is important to note that this project and the HIDDongle project are not compliant with any *Bluetooth* specification. They are simply intended to be used a demonstration of the HID capabilities of BLE.

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

C:\Texas Instruments\BLE-CC2540-1.1\Projects\ble\EmulatedKeyboard\CC2540\EmulatedKeyboard.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 one configuration, **CC2540DK-MINI Keyfob Slave**, using the keyfob hardware platform.

3.1.1 Basic Operation

When not connected, the keyfob will always be advertising, using the fixed public address of 22:22:22:22:22:22. The HIDDongle application will always try to initiate a connection with this device address. This fixed address can be changed by modifying the value of **DEVICE_ADDRESS** in **EmulatedKeyboard.c**; however the HIDDongle application will then no longer connect to the emulated keyboard unless it is modified as well. This concept of using a fixed address is not standard by the *Bluetooth* specification, and having multiple devices advertising with the same address may result in some unexpected behaviour.

When in a connection, the keyfob's left button well send an “up” command, and the right button will send a “down” command. If connected to a CC2540 USB dongle running the HIDDongle

project, the connected PC should treat these key presses the same as pressing the up and down keys on a keyboard.

To change the key codes for the two keys, simply modify the **emulatedKeyboard_HandleKeys** and change the value of **keyData[1]** when a button is pressed. Instead of setting the values to **KEY_UP_ARROW** or **KEY_DOWN_ARROW**, you can use a different keyboard scan code. Scan codes for each of a standard keyboard can be found in the file **hal_scancodes.h**.

3.2 Software Description

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

3.2.1 Initialization

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

3.2.2 Event Processing

The application has two main event processing functions, **EmulatedKeyboard_ProcessEvent** and **EmulatedKeyboard_ProcessOSALMsg**.

Function **EmulatedKeyboard_ProcessEvent** handles events as follows:

- **SYS_EVENT_MSG**: Service the OSAL queue and process OSAL messages.
- **EK_START_DEVICE_EVT**: Start the device, as described in the previous section.
- **EK_PERIODIC_EVT**: Calls the performPeriodicTask function, which current does nothing (it is left there for use by the developer).

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

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

3.2.3 HID Keyboard Profile

The custom HID Keyboard profile implements one GATT service containing one characteristic. The GATT service can be found in the file **hidKeyboardProfile.c**. This service, as well as the service and characteristic UUIDs, are not compliant with any *Bluetooth* specifications and are intended for demonstration only.

The service contains one characteristic. The characteristic value (**hidKeyboardData**) is a three-byte array. The first byte (index 0) represents whether key was pressed or released, with a value of 1 indicating that a key was pressed, and a value of 0 indicating that the key was released. The second byte (index 1) represents the scan code for the key that was pressed, or is 0 if a key was released. The third byte (index 2) is a bit field representing the current state of any modifier keys, such as SHIFT, CTRL, and ALT. Each bit represents one modifier key, with a 1 bit indicating that the key is currently pressed, and a 0 indicating that it is not currently pressed. The scan codes for each key of the keyboard, as well as the modifier key bits, are all defined in the file **hal_scancodes.h**.

3.2.4 Notifications

Whenever a key is pressed or released, the application calls the profile function **HidKeyboard_SetParameter** and sets a new three-byte value for **hidKeyboardData**. The profile then checks whether a connection is active and whether notifications have been enabled for the characteristic, and if so will send a notification to the GATT client device.

It should be noted that the default behaviour of the characteristic is that notifications are enabled. This behaviour is not compliant with the *Bluetooth* core specification, in that the GATT client

device is supposed to enable notifications by writing to the client characteristic configuration descriptor.

4 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 0 and [3]).

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

C:\TexasInstruments\BLE-CC2540-1.1\Projects\ble\Thermometer\CC2540DB\thermometer.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 two configurations.

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

4.1.1 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.

4.1.2 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.

4.2 Software Description

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

4.2.1 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.

4.2.2 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.

4.2.3 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.

4.2.4 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.

4.2.5 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.

4.2.6 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.

4.2.7 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

5 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 following specifications:

- Heart Rate Profile Draft Specification d09r05
- Heart Rate Service Draft Specification d09r07
- Battery Service Draft Specification d09r09

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

C:\Texas Instruments\BLE-CC2540-1.1\Projects\ble\HeartRate\CC2540DB\heartrate.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, 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.

5.1.2 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**.

5.2 Software Description

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

5.2.1 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.

5.2.2 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.

5.2.3 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.

5.2.4 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.

6 HID Dongle

This project implements a USB HID keyboard device for use with the CC2540 USB dongle as BLE central device. When used in conjunction with the EmulatedKeyboard project as a peripheral device (see section 3), this application allows you to actually type text or enter commands on any PC with a USB port, as long as the operating system supports the USB HID (Human Interface Device) profile.

It is important to note that this project and the EmulatedKeyboard project are not compliant with any *Bluetooth* specification. They are simply intended to be used a demonstration of the HID capabilities of BLE.

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

C:\Texas Instruments\BLE-CC2540-1.1\Projects\ble\HIDDongle\CC2540\HIDDemo.eww

6.1 Project Overview

The source code structure of this project is slightly different than other projects in the BLE software development kit. This is due to the fact that the project was ported from the Texas Instruments RemoTI software development kit, which implements the ZigBee RF4CE protocol for RF remote controls. Even though the source code appears different, the application still follows the same general architecture of other projects using the BLE protocol stack.

The project includes a USB HID stack, which is provided as source code. The USB HID stack source code can be found in the following two directories:

C:\Texas Instruments\BLE-CC2540-1.1\Projects\ble\HIDDongle\Application\usbclass_hid

C:\Texas Instruments\BLE-CC2540-1.1\Projects\ble\HIDDongle\Application\usblibrary

The project contains one configuration, **CC2540USB HID Keyboard Central**, which runs on the CC2540 USB dongle included with the CC2540DK-MINI kit.

6.1.1 Basic Operation

When the USB dongle is plugged into a PC, the dongle will enumerate as a USB HID keyboard device to the PC operating system. From the standpoint of the PC, this should be no different than plugging in any USB keyboard.

After the dongle initializes, it will remain in a standby state with LED 2 turned on. By pressing the button SW1 on the dongle, it will attempt to initiate a connection with a slave device with the public device address of 22:22:22:22:22:22, which is the fixed address of the EmulatedKeyboard application. If a present EmulatedKeyboard device is advertising, a connection should be established and LED 2 on the dongle will turn off.

Once connected, the dongle will act as a GATT client and wait for key press or release notifications to be sent from the server. Each time a notifications is received, LED 2 will blink. Any notifications that are received by the dongle are all processed in the same way; the dongle will not check the attribute handle or verify that the notifications are actually coming from the HidKeyboardService (see section 3.2.3). The data from the notification will be processed as either a key press or release, and will be sent up the USB stack to the host device as a HID report.

To terminate the connection with the peripheral at any time, simply press button SW2 on the dongle.

6.2 Software Description

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

6.2.1 Initialization

The initialization of the application occurs in two phases: first, the **hidappInit** function is called by the OSAL. This function configures GAP connection parameters, initializes the GATT client, and sets an OSAL **HIDAPP_EVT_START** event. This triggers the second phase of the initialization, which can be found within the **hidappProcessEvent** function. During this phase, the **GAPCentralRole_StartDevice** function is called to initialize the device in the GAP central role.

6.2.2 Event Processing

The application has a main event processing functions, **hidappProcessEvent**.

Function **EmulatedKeyboard_ProcessEvent** handles events as follows:

- **SYS_EVENT_MSG**: Service the OSAL queue and process OSAL messages.
- **HIDAPP_EVT_START**: Start the device, as described in the previous section.
- **HIDAPP_EVT_KEY_RELEASE_TIMER**: Handles detection of key release
- **HIDAPP_EVT_PROCESS**: Sends a HID report up the USB stack after notifications of key presses are received.
- **HIDAPP_EVT_PROCESS_RELEASE**: Sends a HID report up the USB stack after notifications of key releases are detected, or when the BLE connection is terminated.

When an OSAL **GATT_MSG_EVENT** message is received, the function **processGATTMsg** is called. The only type of GATT message that is processed is a message with type **ATT_HANDLE_VALUE_NOTI**; all other GATT messages are ignored. The data from the notification gets parsed, and the key press data gets placed into the HID report buffer. A **HIDAPP_EVT_PROCESS** event gets set with a 5ms delay. Once the **HIDAPP_EVT_PROCESS** event occurs, the data in the HID report buffer gets processed with a call to **hidProcessCercCtrl**.

6.2.3 Callbacks

The HIDDongle application contains two callback functions: **hidappKeyCback** and **centralEventCB**.

The **hidappKeyCback** function is called every time one of the buttons on the dongle is pressed. This function uses the GAP central role profile to initiate and terminate the BLE connection.

The **centralEventCB** function is registered with the GAP central role profile upon initialization. Every time the GAP state of the dongle changes, this function is called. It is here that the state of LED 2 changes based on the connection state of the dongle.

7 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].

8 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 [5])
- Link Loss (for Proximity Profile, Reporter role; see [6] and [8])
- Immediate Alert (for Proximity Profile, Reporter role and Find Me Profile, Target role; see [6], [7], and [9])
- Tx Power (for Proximity Profile, Reporter role; see [10])
- Battery
- Accelerometer
- SimpleKeys

The battery, 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.

8.1 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.

8.1.1 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.

8.1.2 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>

8.1.3 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>.

8.1.4 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.

8.1.5 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 alert 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.

8.2 Software Description

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

8.2.1 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.

8.2.2 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.

8.2.3 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.

9 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].

10 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].

11 Soft Command Trigger- BLE Remote Control

This sample project implements the Soft Command and Battery profiles in a BLE peripheral device to provide an example of a Soft Command device like a remote control. The application implements the "Trigger" role of the Soft Command profile and the "Battery Reporter" role of the Battery profile. The project is based on following specification:

- Soft Command Draft Specification d05r02 (UCRDD)

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

C:\Texas Instruments\BLE-CC2540-1.1\Projects\ble\SoftCmd\CC2540DB\softcmd.eww

11.1 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.

11.1.1 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 sends "Soft Command 0" and the right button sends "Soft Command 1".

11.1.2 Basic Operation

Power up the device and press the right button to enable advertising. From a Soft Command target peer device, initiate a device discovery and connection procedure to find and connect to the Soft Command trigger device. Once connected the application performs service discovery to find the Generic Control service on the peer device. The peer device may discover the Command Enumeration service. The peer device may also discover and configure the battery service for battery level-state notifications.

If the Generic Control service is discovered then soft commands will be sent to the peer device when a button is pressed.

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

11.2 Software Description

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

11.2.1 Initialization

The initialization of the application occurs in two phases: first, the **SoftCmd_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 Soft Command service and the battery service along with standard GATT and GAP services in the attribute server. It also initializes GATT for client operation. Then it sets an OSAL **START_DEVICE_EVT** event. This triggers the second phase of the initialization, which can be found within the **SoftCmd_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.

11.2.2 Event Processing

The application has two main event processing functions, **SoftCmd_ProcessEvent** and **softCmd_ProcessOSALMsg**.

Function **SoftCmd_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.
- **BATT_PERIODIC_EVT**: Check the battery level and send notification if it changed.

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

- **KEY_CHANGE** messages: Call function **softCmd_HandleKeys** to handle keypresses.
- **GATT_MSG_EVENT** messages: Call function **softCmdProcessGATTMsg** to handle messages from GATT.

11.2.3 Callbacks

The application callback functions are as follows:

- **softCmdGapStateCB**: This is the GAP event callback. It processes GAP events for startup and link connect/disconnect.
- **softCmdPairStateCB**: This is the GAP bond manager state callback. It processes events for pairing start and pairing complete.
- **softCmdBattCB**: 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.

11.2.4 Service Discovery

The application performs service discovery for the Generic Control service. 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 **softCmdDiscStart**. 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.

The service discovery procedure is as follows:

1. Discovery the Generic Control service by UUID.

2. If found, discover all characteristics of the service to find the Generic Command Control Point and Command Enumeration characteristics.

3. If the Command Enumeration characteristic is found, read the characteristic.

If the Generic Command Control Point characteristic is discovered then discovery of the service is deemed successful.

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

The main service discovery function is **softCmdDiscGenCtrl**. This function is executed when a discovery-related GATT message response is received. The service discovery state is maintained in variable **softCmdDiscState**.

11.2.5 Sending Soft Commands

Function **softCmdSend** is called when a button is pressed during a connection. This function calls **GATT_WriteNoRsp** to sends a write command protocol message containing the soft command value to the peer device.

12 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 following profiles are implemented:

- Time Profile, Time Client role

  Based on Time Profile Draft Specification d09r08

- Alert Notification Profile, Alert Client role

  Based on Alert Notification Service Draft Specification d09r02

- Network Availability Profile, Network Monitor role

  Based on Network Availability Draft Specification d05r04 (UCRDD)

- Battery Profile, Battery Monitor role

  Based on Battery Profile Draft Specification d09r06

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

C:\Texas Instruments\BLE-CC2540-1.1\Projects\ble\TimeApp\CC2540\TimeApp.eww

12.1 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.

12.1.1 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.

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.

12.1.2 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
- 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**.

12.2 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.

12.2.1 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.

12.2.2 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.

12.2.3 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.

12.2.4 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:

4. Discovery the service by UUID.
5. If found, discover all characteristics of the service. Cache the handles of characteristics of interest.
6. 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**.

12.2.5 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.

12.2.6 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.

12.2.7 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**.

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

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**	
Audio	www.ti.com/audio	Communications and Telecom	www.ti.com/communications
Amplifiers	amplifier.ti.com	Computers and Peripherals	www.ti.com/computers
Data Converters	dataconverter.ti.com	Consumer Electronics	www.ti.com/consumer-apps
DLP® Products	www.dlp.com	Energy and Lighting	www.ti.com/energy
DSP	dsp.ti.com	Industrial	www.ti.com/industrial
Clocks and Timers	www.ti.com/clocks	Medical	www.ti.com/medical
Interface	interface.ti.com	Security	www.ti.com/security
Logic	logic.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Power Mgmt	power.ti.com	Transportation and Automotive	www.ti.com/automotive
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com	Wireless	www.ti.com/wireless-apps
RF/IF and ZigBee® Solutions	www.ti.com/lprf		
	TI E2E Community Home Page		e2e.ti.com

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