

Chapter 5: Security and Privacy

In this chapter you will learn about security and privacy in Bluetooth® LE. First, the concepts for both security and privacy will be explained. Then the method to implement them in the firmware will be described along with the stack events that will occur. Methods to store security information in non-volatile memory will also be explained.

Table of contents

5.1	Security	2
5.1.1	Pairing.....	3
5.1.2	Bonding	5
5.1.3	Pairing & Bonding process summary	5
5.1.4	Authentication, Authorization and the GATT database.....	5
5.2	Privacy.....	6
5.3	Bluetooth® configurator settings for security and privacy.....	8
5.3.1	Security.....	8
5.3.2	Privacy	9
5.4	Firmware for security and privacy	10
5.4.1	Pairing.....	10
5.4.2	Pairing + Bonding + Privacy	12
5.5	Exercises	16
	Exercise 1: Paring.....	16
	Exercise 2: Bonding.....	18
	Exercise 3: Passkey	23
	Exercise 4: Numeric comparison.....	25
	Exercise 5: Store Bonding Information for Multiple Devices.....	27
5.6	Appendix.....	30
5.6.1	Exercise 2 Answers	30
5.6.2	Exercise 3 Answers	31

Document conventions

Convention	Usage	Example
Courier New	Displays code and text commands	CY_ISR_PROTO(MyISR) ; make build
<i>Italics</i>	Displays file names and paths	<i>sourcefile.hex</i>
[bracketed, bold]	Displays keyboard commands in procedures	[Enter] or [Ctrl] [C]
Menu > Selection	Represents menu paths	File > New Project > Clone
Bold	Displays GUI commands, menu paths and selections, and icon names in procedures	Click the Debugger icon, and then click Next .

5.1 Security

In any type of system, to securely communicate between two devices you need to:

1. Authenticate that both sides know who they are talking to
2. Ensure that all access to data is Authorized
3. Encrypt all messages that are transmitted
4. Verify the Integrity of those messages.

In Bluetooth® LE, this entire security framework is built around AES-128 symmetric key encryption. This type of encryption works by combining a shared secret code and the unencrypted data (typically called plain text) to create an encrypted message (typically called cypher text).

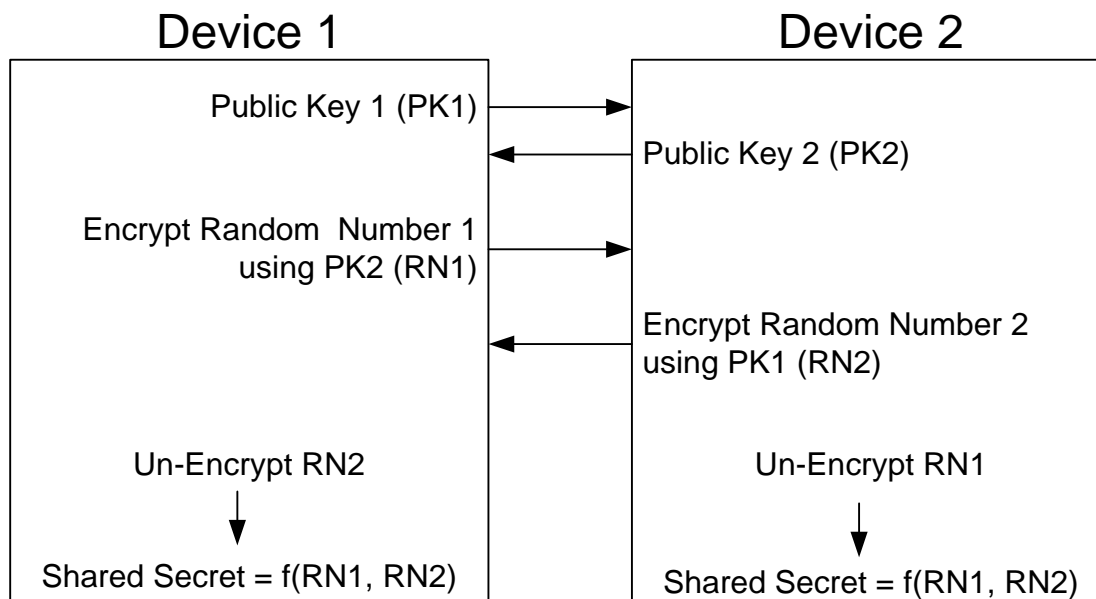
- $CypherText = F(SharedSecret, PlainText)$

There is a bunch of math that goes into AES-128, but for all practical purposes if the shared secret code is kept secret, you can assume that it is very unlikely that someone can read the original message.

If this scheme depends on a shared secret, the next question is: how do two devices that have never been connected get a shared secret that no one else can see? In Bluetooth® LE, the process for achieving this state is called Pairing. A device that is Paired is said to be Authenticated.

5.1.1 Pairing

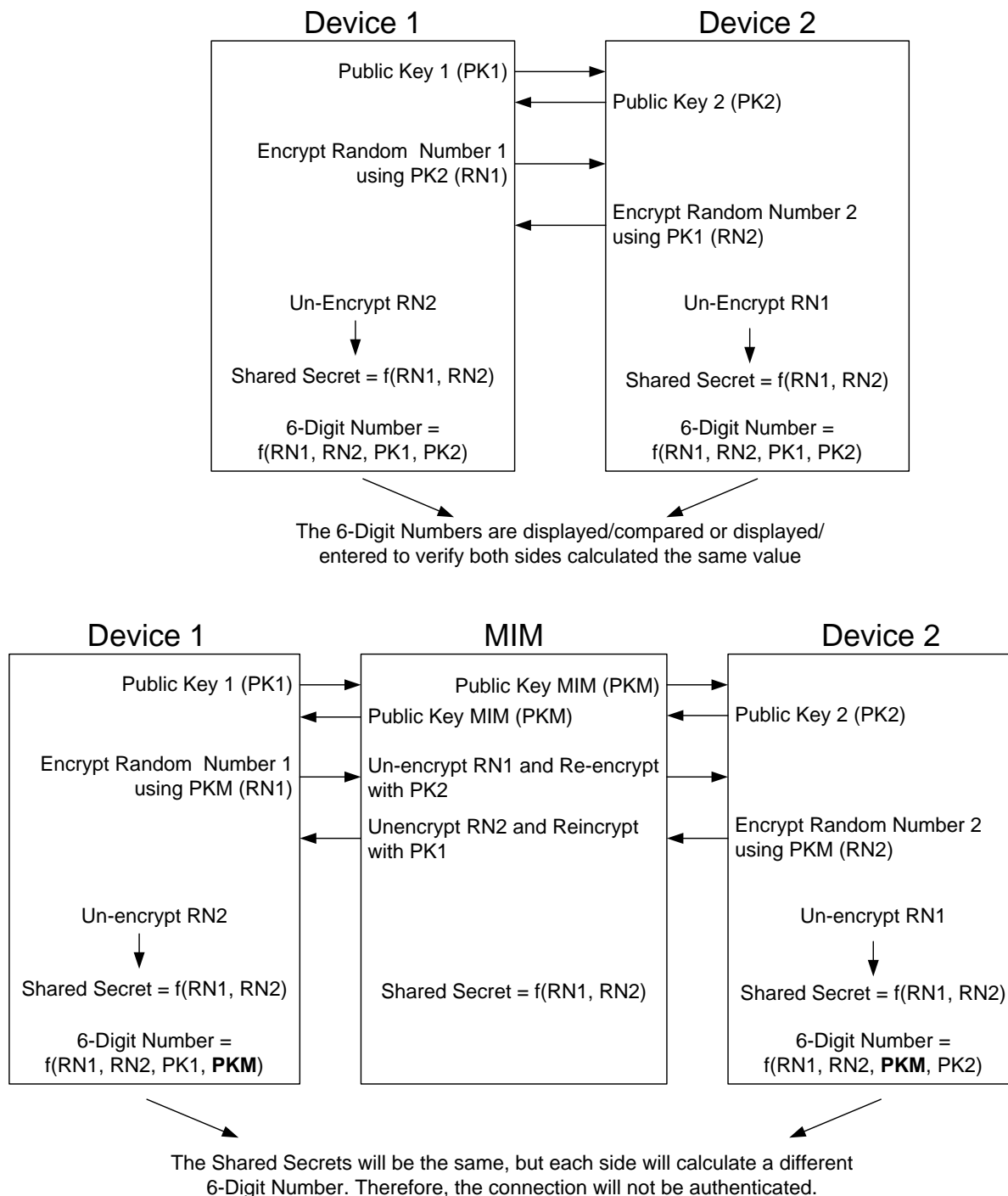
Pairing is the process of arriving at the shared secret. The basic problem continues to be how do you send a shared secret over the air, unencrypted and still have your shared secret stay a secret? The answer is that you use public key encryption. Both sides have a public/private key pair that is either embedded in the device or calculated at startup. When you want to authenticate, both sides of the connection exchange public keys. Then both sides exchange encrypted random numbers that form the basis of the shared secret.



But how do you protect against Man-In-The-Middle (MIM)? There are four possible methods.

- Method 1 is called "Just works". In this mode you have no protection against MIM.
- Method 2 is called "Out of Band". Both sides of the connection need to be able to share the PIN via some other connection that is not Bluetooth® such as NFC.
- Method 3 is called "Numeric Comparison" (V2.PH.7.2.1). In this method, both sides display a 6-digit number that is calculated with a nasty cryptographic function based on the random numbers used to generate the shared key and the public keys of each side. The user observes both devices. If the number is the same on both, then the user confirms on one or both sides. If there is a MITM, then the random numbers on both sides will be different so the 6-digit codes would not match.
- Method 4 is called "Passkey Entry" (V2.PH.7.2.3). For this method to work, at least one side needs to be able to enter a 6-digit Passkey. The other side must be able to display the Passkey. One device displays the Passkey and the user is required to enter the Passkey on the other device. Then an exchange and comparison process happens with the Passkeys being divided up, encrypted, exchanged and compared.

Pictorially, the process with no MIM and with MIM for methods 3 and 4 is shown below. If there is a man in the middle, the two sides will calculate different numbers because the number is a function of the public keys used to encrypt the random numbers.

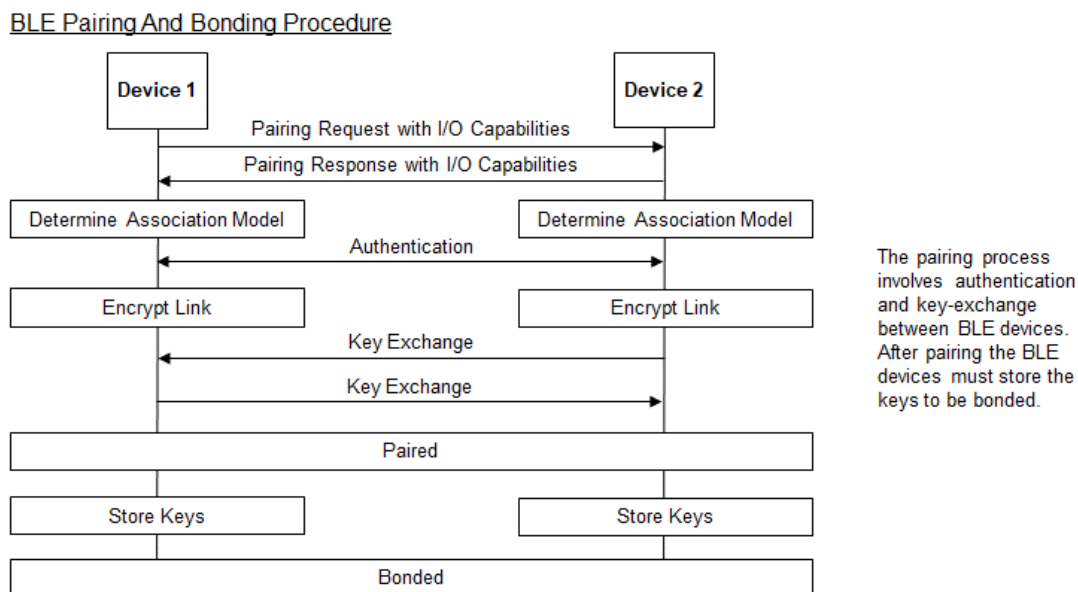


5.1.2 Bonding

The whole process of Pairing is a bit painful and time consuming. It is also the most vulnerable part of establishing security, so it is beneficial to do it only once. Certainly, you don't want to have to repeat it every time two devices connect. This problem is solved by Bonding, which just saves all the relevant information into non-volatile memory. The information is stored by both devices that are part of the connection. This allows the next connection between those two devices to happen without repeating the pairing process.

5.1.3 Pairing & Bonding process summary

The process that occurs when you pair and bond can be seen pictorially below.



5.1.4 Authentication, Authorization and the GATT database

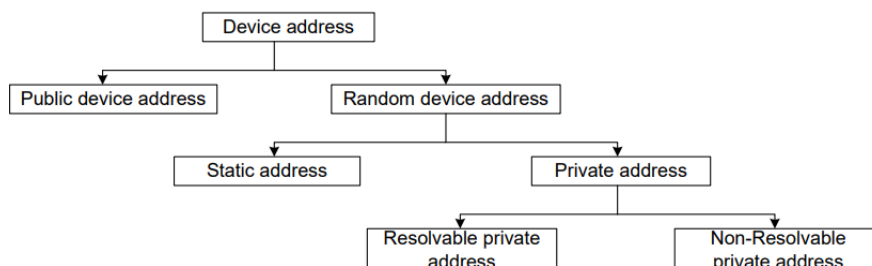
In prior chapters we talked about Attributes and the GATT Database. Each Attribute has a permissions bit field that includes bits for Encryption, Authentication, and Authorization. The stack will guarantee that you will not be able to access an Attribute that is marked for Encryption or Authentication unless the connection is Authenticated and/or Encrypted.

The Authorization flag is not enforced by the stack. The application is responsible for implementing the Authorization semantics. For example, you might not allow someone to turn off/on a switch without entering a password first.

5.2 Privacy

Bluetooth® LE devices are identified using a 48-bit device address. This device address is part of all the packets sent by the device in the advertising channels. A third device which listens on all three advertising channels can easily track the activities of a device by using its device address. Link Layer Privacy is a feature that reduces the ability to track a Bluetooth® LE device by using a private address that is generated and changed at regular intervals. Note that this is different than security (i.e. encrypting of messages).

There are a few different address types possible for Bluetooth® LE devices:



The device address can be a Public Device Address or a Random Device Address. The Public Device Addresses are comprised of a 24-bit company ID (an Organizationally Unique Identifier or OUI based on an IEEE standard) and a 24-bit company-assigned number (unique for each device); these addresses do not change over time.

There are two types of Random Addresses: Static Address and Private Address. The Static Address is a 48-bit randomly generated address with the two most significant bits set to 1. Static Addresses are generated on first power up or during manufacturing. A device using a Public Device Address or Static Address can be easily discovered and connected to by a peer device. Private Addresses change at some interval to ensure that the Bluetooth® LE device cannot be tracked. A Non-Resolvable Private Address cannot be resolved by any device so the peer cannot identify who it is connecting to. Resolvable Private Addresses (RPA) can be resolved and are used by Privacy-enabled devices.

Every Privacy-enabled Bluetooth® LE device has a unique address called the Identity Address and an Identity Resolving Key (IRK). The Identity Address is the Public Address or Static Address of the Bluetooth® LE device. The IRK is used by the Bluetooth® LE device to generate its RPA and is used by peer devices to resolve the RPA of the Bluetooth® LE device. Both the Identity Address and the IRK are exchanged during the third stage of the pairing process. Privacy-enabled Bluetooth® LE devices maintain a list that consists of the peer device's Identity Address, the local IRK used by the Bluetooth® LE device to generate its RPA, and the peer device's IRK used to resolve the peer device's RPA. This is called the Resolving List. Only peer devices that have the 128-bit identity resolving key (IRK) of a Bluetooth® LE device can determine the device's address.

A Privacy-enabled Bluetooth® LE device periodically changes its RPA to avoid tracking. The Bluetooth® LE Stack configures the Link Layer with a value called RPA Timeout that specifies the time after which the Link Layer must generate a new RPA.

Apart from this, Bluetooth® 5.0 introduced more options in the form of privacy modes. There are two modes: device privacy mode and network privacy mode. A device in device privacy mode is only concerned about the privacy of the device itself and will accept advertising physical channel PDU's (Advertising, Scanning and Initiating packets) from peer devices that contain their identity address as well as ones that contain a private address, even if the peer device has distributed its IRK in the past. In network privacy mode, a device will only accept advertising packets from peer devices that contain a private address. By default, network privacy

mode is used when private addresses are resolved and generated by the Controller. The Host can specify the privacy mode to be used with each peer identity on the resolving list.

The following table shows the logical representation of the resolving list entries. Depending on the privacy mode entry in the resolving list, the device will behave differently with each peer device.

Device	Local IRK	Peer IRK	Peer Identity Address	Identity Address Type	Privacy Mode
1	Local IRK	Peer 1 IRK	Peer 1 Identity Address	Static/Public	Network/Device
2	Local IRK	Peer 2 IRK	Peer 2 Identity Address	Static/Public	Network/Device
3	Local IRK	Peer 3 IRK	Peer 3 Identity Address	Static/Public	Network/Device

5.3 Bluetooth® configurator settings for security and privacy

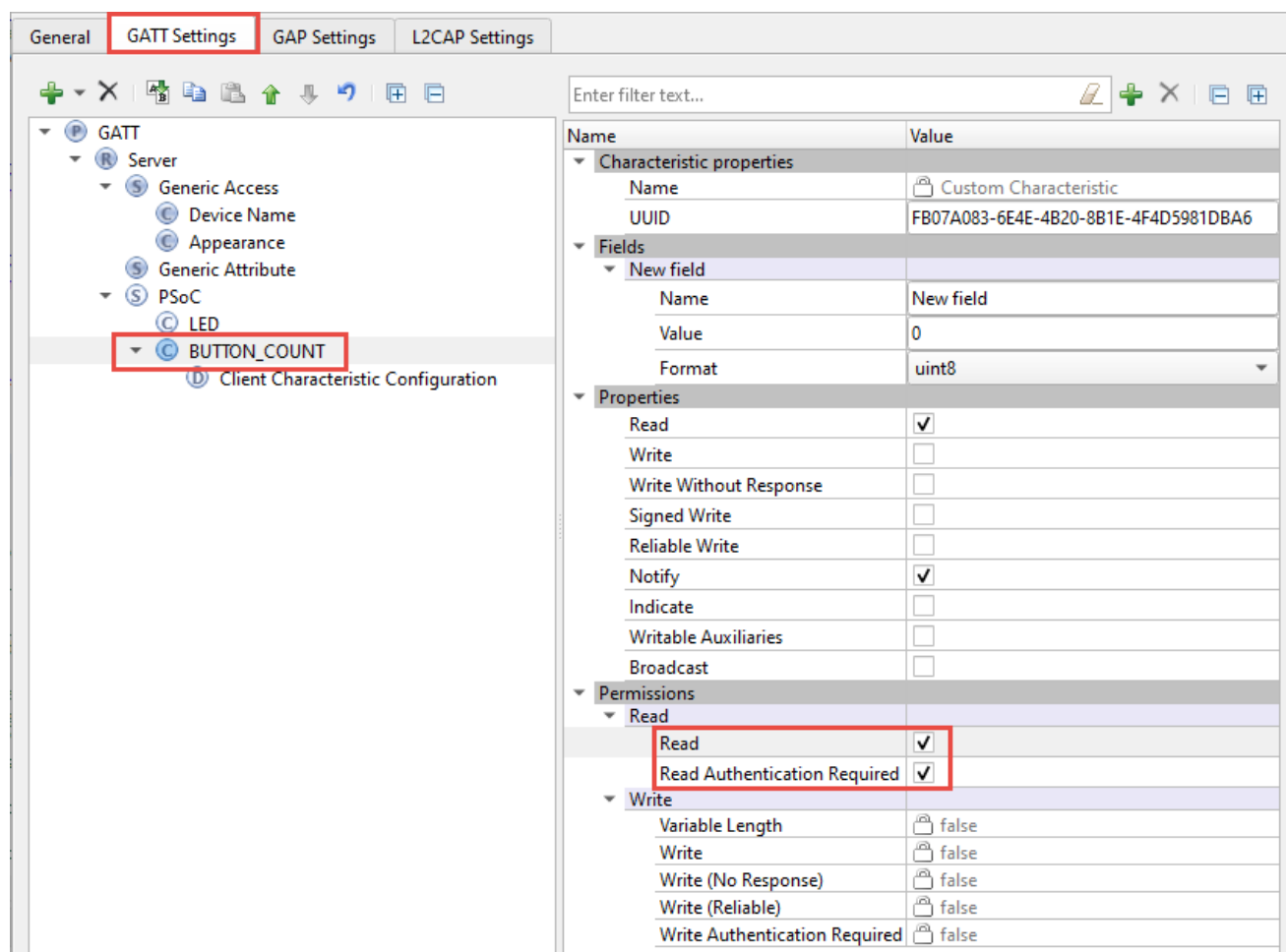
The ModusToolbox™ Bluetooth® configurator makes it simple to set up security and privacy. Examples of each are shown below.

5.3.1 Security

In order to enable security (i.e. to require pairing before allowing the client read or write of a characteristic) you just click the "Read Authentication Required" or "Write Authentication Required" button in the permissions for that characteristic. Note that this is a bitmask setting, so you must still keep "Read" and/or "Write" selected when you enable authentication. If not, reads/writes to that characteristic will fail. Enabling security works the same way for Descriptors such as the CCCD. That is, you can require authentication before allowing reads/writes of the CCCD (thereby preventing the client from turning Notifications on/off without pairing first).

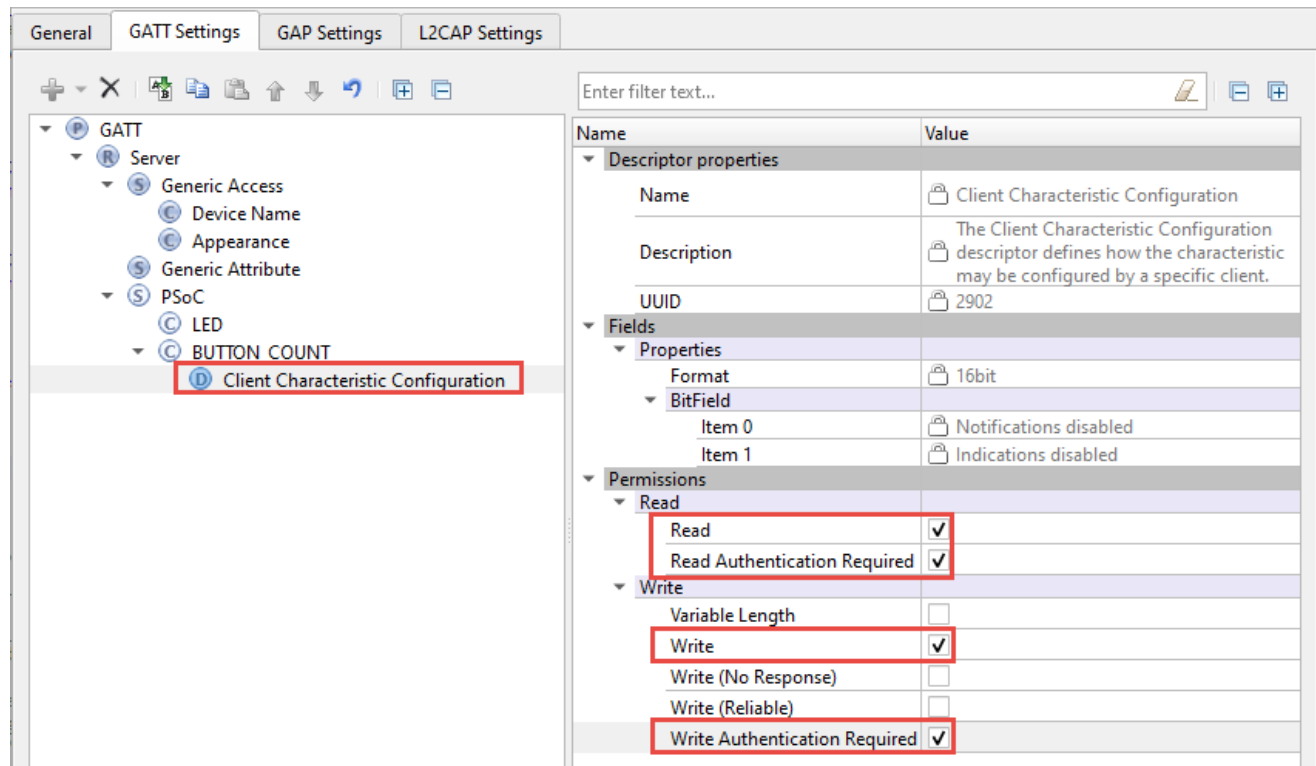
Note: The security settings for each Characteristic and its Descriptors are independent so make sure you set the permissions for everything the way you want them.

As an example, for prior exercises, the Bluetooth® configuration settings for the BUTTON_COUNT and its CCCD with security enabled look like this:



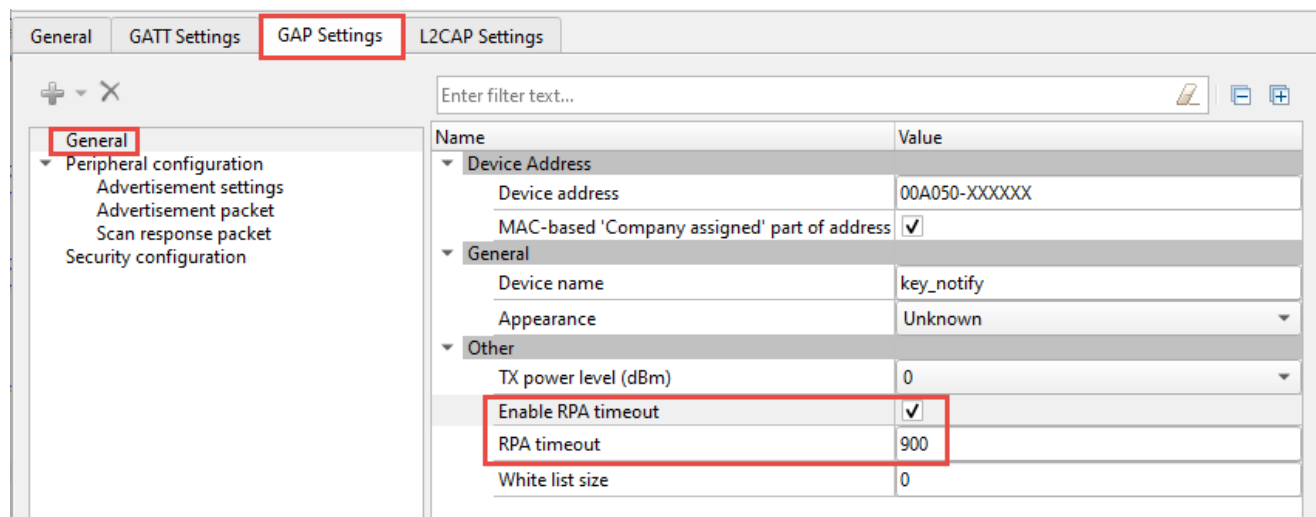
The screenshot shows the ModusToolbox Bluetooth configurator interface. The 'GATT Settings' tab is selected. In the left-hand tree view, the 'BUTTON_COUNT' characteristic is highlighted. The right-hand pane displays the configuration for this characteristic. The 'Permissions' section is expanded, and the 'Read' checkbox is checked. The 'Read Authentication Required' checkbox is also checked, indicating that security is enabled for reading this characteristic.

Name	Value
Characteristic properties	
Name	Custom Characteristic
UUID	FB07A083-6E4E-4B20-8B1E-4F4D5981DBA6
Fields	
New field	
Name	New field
Value	0
Format	uint8
Properties	
Read	<input checked="" type="checkbox"/>
Write	<input type="checkbox"/>
Write Without Response	<input type="checkbox"/>
Signed Write	<input type="checkbox"/>
Reliable Write	<input type="checkbox"/>
Notify	<input checked="" type="checkbox"/>
Indicate	<input type="checkbox"/>
Writable Auxiliaries	<input type="checkbox"/>
Broadcast	<input type="checkbox"/>
Permissions	
Read	
Read	<input checked="" type="checkbox"/>
Read Authentication Required	<input checked="" type="checkbox"/>
Write	
Variable Length	false
Write	false
Write (No Response)	false
Write (Reliable)	false
Write Authentication Required	false



5.3.2 Privacy

A Resolvable Private Address (RPA) can be enabled for a device simply by checking the **Enable RPA timeout** box in the configurator's **GAP Settings** tab. Once the box is checked, there is also an **RPA timeout** value that can be changed if desired. The default is 900 seconds (i.e. 15 minutes).



Note: Enable RPA timeout is selected in all of our exercises because some iOS devices require that RPA is enabled for peripherals to function properly.

5.4 Firmware for security and privacy

The only differences in firmware for security and privacy are additional Bluetooth® stack management events and GATT database events that your firmware needs to respond to.

5.4.1 Pairing

For a typical application that connects using a Paired link but does NOT use privacy, does NOT store bonding information in non-volatile memory (e.g. flash), and does NOT require a passkey, the callback events will look like this:

Activity	Callback Event Name (both Stack and GATT)	Reason
Powerup	BTM_LOCAL_IDENTITY_KEYS_REQUEST_EVT	At initialization, the Bluetooth LE stack looks to see if the privacy keys are available. A return value of <code>WICED_BT_ERROR</code> indicates to the stack that it should generate new privacy keys since we are not storing values yet.
	BTM_ENABLED_EVT	This occurs once the Bluetooth LE stack has completed initialization. Typically, you will start up the rest of your application here.
	BTM_BLE_ADVERT_STATE_CHANGED_EVT	This occurs when you enable advertisements. You will see a return value of 3 for fast advertisements. After a timeout, you may see this again with a return value of 4 for slow advertisements. Eventually the state changes to 0 (off) if there have been no connections, giving you a chance to save power.
	BTM_LOCAL_IDENTITY_KEYS_UPDATE_EVT	This event is called if reading of the privacy keys from flash failed (i.e. the return value from <code>BTM_LOCAL_IDENTITY_KEYS_REQUEST_EVT</code> was not <code>WICED_BT_SUCCESS</code>). Since we are not yet implementing bonding, this should just return <code>WICED_BT_SUCCESS</code> .
Connect	BTM_PAIRING_DEVICE_LINK_KEYS_REQUEST_EVT	The stack is requesting pairing keys for the connected peer. In this case, we have not saved any keys in flash so this state just returns <code>WICED_BT_ERROR</code> to tell the stack to generate new keys.
	GATT_CONNECTION_STATUS_EVT	The callback needs to determine if the event is a connection or a disconnection. For a connection, the connection ID is saved.
	BTM_BLE_ADVERT_STATE_CHANGED_EVT	Once the connection happens, the stack stops advertisements which will result in this event. You will see a return value of 0 which means advertisements have stopped.
	BTM_PAIRING_DEVICE_LINK_KEYS_REQUEST_EVT	The stack looks at security information for the previous pairing (if any). This is done to prevent a peer from reconnecting with reduced security. In this case, we have not saved any keys in flash so this state just returns <code>WICED_BT_ERROR</code> .

Activity	Callback Event Name (both Stack and GATT)	Reason
Pair (if secure link is required)	BTM_SECURITY_REQUEST_EVT	The occurs when the client requests a secure connection. When this event happens, you need to call <code>wiced_bt_ble_security_grant</code> to allow a secure connection to be established.
	BTM_PAIRING_IO_CAPABILITIES_BLE_REQUEST_EVT	This occurs when the client asks what type of capability your device has that will allow validation of the connection (e.g. screen, keyboard, etc.). You need to set the appropriate values when this event happens.
	BTM_ENCRYPTION_STATUS_EVT	This occurs when the secure link has been established.
	BTM_PAIRING_DEVICE_LINK_KEYS_REQUEST_EVT	The latest keys are requested by the stack before requesting the application to write them to flash. Again, in this case, we have not saved any keys in flash so this state just returns <code>WICED_BT_ERROR</code> .
	BTM_PAIRING_DEVICE_LINK_KEYS_UPDATE_EVT	This event is used so that you can store the paired devices keys if you are storing bonding information. If not, then this state should just return <code>WICED_BT_SUCCESS</code> .
	BTM_PAIRING_COMPLETE_EVT	This event indicates that pairing has been completed successfully.
Read Values	GATT_ATTRIBUTE_REQUEST_EVT → GATTS_REQ_TYPE_READ	The firmware must get the value from the correct location in the GATT database.
Write Values	GATT_ATTRIBUTE_REQUEST_EVT → GATTS_REQ_TYPE_WRITE	The firmware must store the provided value in the correct location in the GATT database.
Notifications	N/A	Notifications must be sent whenever an attribute that has notifications set is updated by the firmware. Since the change comes from the local firmware, there is no stack or GATT event that initiates this process.
Disconnect	GATT_CONNECTION_STATUS_EVT	For a disconnection, the connection ID is reset, all CCCD settings are cleared, and advertisements are restarted.
	BTM_BLE_ADVERT_STATE_CHANGED_EVT	Upon a disconnect, the firmware will get a GATT event handler callback for the <code>GATT_CONNECTION_STATUS_EVENT</code> (more on this later). At that time, it is the user's responsibility to determine if advertising should be re-started. If it is restarted, then you will get a Bluetooth LE stack callback once advertisements have restarted with a return value of 3 (fast advertising) or 4 (slow advertising).

5.4.2 Pairing + Bonding + Privacy

If bonding information is stored to flash, the event sequence will look like the following. The sequence is shown for three cases (each shaded differently):

1. First-time connection before bonding information is saved
2. Connection after bonding information has been saved for disconnect/re-connect without resetting the kit between connections.
3. Connection after bonding information has been saved for disconnect/reset/re-connect.

In the reconnect cases, you can see that the pairing sequence is greatly reduced since keys are already available.

Activity	Callback Event Name	Reason
1 st Powerup	BTM_LOCAL_IDENTITY_KEYS_REQUEST_EVT	When this event occurs, if privacy is enabled the firmware needs to load the privacy keys from flash, provide them to the stack and return <code>WICED_BT_SUCCESS</code> . If keys have not been previously saved for the device, then this state must return <code>WICED_BT_ERROR</code> . The non-success return value causes the stack to generate new privacy keys.
	BTM_ENABLED_EVT	<p>This occurs once the Bluetooth LE stack has completed initialization. Typically, you will start up the rest of your application here.</p> <p>During this event, the firmware needs to load keys (which also includes the <code>BD_ADDR</code>) for a previously bonded device from flash so that it will have the necessary keys to allow connecting to a bonded device. If a device has not been previously bonded, this will return values of all 0.</p> <p>If RPA is being used, you must also call <code>wiced_bt_dev_add_device_to_address_resolution_db</code> so that the random private addresses can be resolved.</p>
	BTM_BLE_ADVERT_STATE_CHANGED_EVT	This occurs when you enable advertisements. You will see a return value of 3 for fast advertisements. After a timeout, you may see this again with a return value of 4 for slow advertisements. Eventually the state changes to 0 (off) if there have been no connections, giving you a chance to save power.
	BTM_LOCAL_IDENTITY_KEYS_UPDATE_EVT	This event is called if reading of the privacy keys from flash failed (i.e. the return value from <code>BTM_LOCAL_IDENTITY_KEYS_REQUEST_EVT</code> was not <code>WICED_BT_SUCCESS</code>). During this event, the privacy keys must be saved to flash.
	BTM_LOCAL_IDENTITY_KEYS_UPDATE_EVT	This is called twice to update both the IRK and the ER in two steps.

Activity	Callback Event Name	Reason
1 st Connect	BTM_PAIRING_DEVICE_LINK_KEYS_REQUEST_EVT	The stack looks at security information for the previous pairing (if any). This is done to prevent a peer from reconnecting with reduced security. In this case, we have not saved any keys in flash so this state just returns <code>WICED_BT_ERROR</code> .
	GATT_CONNECTION_STATUS_EVT	The callback needs to determine if the event is a connection or a disconnection. For a connection, the connection ID is saved, and the BDA of the remote device is saved so that it is available during numeric comparison security requests.
	BTM_BLE_ADVERT_STATE_CHANGED_EVT	Once the connection happens, the stack stops advertisements which will result in this event. You will see a return value of 0 which means advertisements have stopped.
1 st Pair	BTM_PAIRING_DEVICE_LINK_KEYS_REQUEST_EVT	The latest keys are requested by the stack before requesting the application to write them to flash. Again, in this case, we have not saved any keys in flash yet so this state just returns <code>WICED_BT_ERROR</code> .
	BTM_SECURITY_REQUEST_EVT	This occurs when the client requests a secure connection. When this event happens, you need to call <code>wiced_bt_ble_security_grant</code> to allow a secure connection to be established.
	BTM_PAIRING_IO_CAPABILITIES_BLE_REQUEST_EVT	This occurs when the client asks what type of capability your device has that will allow validation of the connection (e.g. screen, keyboard, etc.). You need to set the appropriate values when this event happens.
	BTM_PASSKEY_NOTIFICATION_EVT	This event only occurs if the IO capabilities are set such that your device has the capability to display a value, such as <code>BTM_IO_CAPABILITIES_DISPLAY_ONLY</code> . In this event, the firmware should display the passkey so that it can be entered on the client to validate the connection.
	BTM_USER_CONFIRMATION_REQUEST_EVT	This event only occurs if the IO capabilities are set such that your device has the capability to display a value and accept Yes/No input, such as <code>BTM_IO_CAPABILITIES_DISPLAY_AND_YES_NO_INPUT</code> . In this event, the firmware should display the passkey so that it can be compared with the value displayed on the Client. This state should also provide confirmation to the Stack (either with or without user input first).
	BTM_ENCRYPTION_STATUS_EVT	This occurs when the secure link has been established. Previously saved information such as paired device <code>BD_ADDR</code> and notify settings is read. If no device has been previously bonded, this will return all 0's.
	BTM_PAIRING_DEVICE_LINK_KEYS_REQUEST_EVT	The latest keys are requested by the stack before requesting the application to write them to flash. Since this is the first pairing, we have not saved any keys in flash yet so this state just returns <code>WICED_BT_ERROR</code> .

Activity	Callback Event Name	Reason
	BTM_PAURED_DEVICE_LINK_KEYS_UPDATE_EVT	During this event, the firmware needs to store the keys of the paired device (including the BD_ADDR) into flash so that they are available for the next time the devices connect.
	BTM_PAIRING_COMPLETE_EVT	This event indicates that pairing has been completed successfully.
Read Values	GATT_ATTRIBUTE_REQUEST_EVT → GATTS_REQ_TYPE_READ	The firmware must get the value from the correct location in the GATT database.
Write Values	GATT_ATTRIBUTE_REQUEST_EVT → GATTS_REQ_TYPE_WRITE	The firmware must store the provided value in the correct location in the GATT database.
Notifications	N/A	Notifications must be sent whenever an attribute that has notifications set is updated by the firmware. Since the change comes from the local firmware, there is no stack or GATT event that initiates this process.
Disconnect	BTM_BLE_ADVERT_STATE_CHANGED_EVT	Upon a disconnect, the firmware will get a GATT event handler callback for the GATT_CONNECTION_STATUS_EVENT (more on this later). At that time, it is the user's responsibility to determine if advertising should be re-started. If it is restarted, then you will get a Bluetooth LE stack callback once advertisements have restarted with a return value of 3 (fast advertising) or 4 (slow advertising).
Re-Connect	BTM_PAURED_DEVICE_LINK_KEYS_REQUEST_EVT	The stack looks at security information for the previous pairing (if any). This is done to prevent a peer from reconnecting with reduced security. In this case, we have saved keys in flash so this state returns WICED_BT_SUCCESS. If RPA is being used, you must also call <code>wiced_bt_dev_add_device_to_address_resolution_db</code> so that the random private addresses can be resolved.
	GATT_CONNECTION_STATUS_EVT	The callback needs to determine if the event is a connection or a disconnection. For a connection, the connection ID is saved, and the BDA of the remote device is saved so that it is available during numeric comparison security requests.
	BTM_BLE_ADVERT_STATE_CHANGED_EVT	Advertising off.
Re-Pair	BTM_ENCRYPTION_STATUS_EVT	In this state, the firmware reads the state of the server from flash. For example, the saved state of any notify settings may be read. Since the paired device BD_ADDR and keys were already available, no other steps are needed to complete pairing.
Read Values	GATT_ATTRIBUTE_REQUEST_EVT → GATTS_REQ_TYPE_READ	The firmware must get the value from the correct location in the GATT database.
Write Values	GATT_ATTRIBUTE_REQUEST_EVT → GATTS_REQ_TYPE_WRITE	The firmware must store the provided value in the correct location in the GATT database.

Activity	Callback Event Name	Reason
Notifications	N/A	Notifications must be sent whenever an attribute that has notifications set is updated by the firmware. Since the change comes from the local firmware, there is no stack or GATT event that initiates this process.
Disconnect	BTM_BLE_ADVERT_STATE_CHANGED_EVT	Advertising on.
Reset	BTM_LOCAL_IDENTITY_KEYS_REQUEST_EVT	Local keys are loaded from flash.
	BTM_ENABLED_EVT	Stack is enabled. Paired device keys (including the BD_ADDR) are loaded from flash and the device is added to the address resolution database.
	BTM_BLE_ADVERT_STATE_CHANGED_EVT	Advertising on.
Re-Connect	BTM_PAIRING_DEVICE_LINK_KEYS_REQUEST_EVT	This event is called by the stack so that the firmware can load the paired device's keys from flash. Since keys are available, this state must return <code>WICED_BT_SUCCESS</code> . If RPA is being used, you must also call <code>wiced_bt_dev_add_device_to_address_resolution_db</code> so that the random private addresses can be resolved.
	GATT_CONNECTION_STATUS_EVT	The callback needs to determine if the event is a connection or a disconnection. For a connection, the connection ID is saved, and the BDA of the remote device is saved so that it is available during numeric comparison security requests.
	BTM_BLE_ADVERT_STATE_CHANGED_EVT	Advertising off.
Re-Pair	BTM_ENCRYPTION_STATUS_EVT	In this state, the firmware reads the state of the server from flash. For example, the saved state of any notify settings may be read. Since the paired device BD_ADDR and keys were already available in flash, no other steps are needed to complete pairing.
Read Values	GATT_ATTRIBUTE_REQUEST_EVT → GATTS_REQ_TYPE_READ	The firmware must get the value from the correct location in the GATT database.
Write Values	GATT_ATTRIBUTE_REQUEST_EVT → GATTS_REQ_TYPE_WRITE	The firmware must store the provided value in the correct location in the GATT database.
Notifications	N/A	Notifications must be sent whenever an attribute that has notifications set is updated by the firmware. Since the change comes from the local firmware, there is no stack or GATT event that initiates this process.
Disconnect	BTM_BLE_ADVERT_STATE_CHANGED_EVT	Advertising on.

5.5 Exercises

Exercise 1: Paring

In this exercise, you will add Pairing and Security (Encryption) to the completed notification exercise from the previous chapter.

Application Creation



1. Create a new application with the BSP you are using.

On the application template page, use the **Browse** button to start from the completed application for the *ch04_ex01_notify* exercise from the previous chapter. If you did not complete that exercise, the solution can be found in *Projects/key_ch04_ex01_notify*.

Name the new application *ch05_ex01_pair*.



2. Open the Bluetooth® Configurator.



3. On the GAP Settings tab, change the device name to `<init>_pair`.



4. On the GATT Settings tab and make the following changes:

- a. In the BUTTON_COUNT characteristic, set the "Read Authentication Required" permission, which will make the peripheral reject read requests unless the devices are paired first.
- b. Update the Client Characteristic Configuration Descriptor to require authenticated read and write. This will cause the application to require pairing to view or change the notification settings.



5. Save your edits and close the configurator.



6. In *main.c*, make the following changes.

- a. Look for the call to `wiced_bt_set_pairable_mode` mode and set the first argument to `WICED_TRUE` to allow pairing.

Note: Leave the second argument as `WICED_FALSE` - when set to true, this argument indicates that ONLY previously paired devices are allowed to connect.

- b. In the `BTM_PAIRING_IO_CAPABILITIES_BLE_REQUEST_EVT` management case tell the central that you want MITM protection, but the device has no IO capabilities. The required code is shown below.

```
p_event_data->pairing_io_capabilities_ble_request.auth_req =  
BTM_LE_AUTH_REQ_SC_MITM_BOND;  
p_event_data->pairing_io_capabilities_ble_request.init_keys =  
BTM_LE_KEY_PENC|BTM_LE_KEY_PID;  
p_event_data->pairing_io_capabilities_ble_request.local_io_cap =  
BTM_IO_CAPABILITIES_NONE;  
p_event_data->pairing_io_capabilities_ble_request.max_key_size = 0x10;  
p_event_data->pairing_io_capabilities_ble_request.resp_keys =  
BTM_LE_KEY_PENC|BTM_LE_KEY_PID;  
p_event_data->pairing_io_capabilities_ble_request.oob_data = BTM_OOB_NONE;
```

- c. In the `BTM_SECURITY_REQUEST_EVT` management case grant the authorization to the central by using the following code:

```
wiced_bt_ble_security_grant( p_event_data->security_request.bd_addr,  
WICED_BT_SUCCESS );
```


Testing

- ☐ 1. Open a UART terminal and connect to the kit.
- ☐ 2. Program your application to your kit.
- ☐ 3. Open the AIROC™ Connect app and connect to your device.
- ☐ 4. Watch the UART messages during the next steps to see which Bluetooth® events occur.
- ☐ 5. Connect to the device.
- ☐ 6. Open the GATT browser, navigate to the LED Characteristic (the one with Read and Write Properties) and read/write the value.

Notice the events that occur. Remember that we did NOT require authentication for this Characteristic which is reflected in the events that you will see.
- ☐ 7. Now navigate to the BUTTON_COUNT Characteristic (the one with Read and Notify Properties) and read the value. When requested, accept the invitation to pair the devices.

Notice the events that occur. Remember that this Characteristic requires authentication.
- ☐ 8. Disconnect from the AIROC™ Connect app.
- ☐ 9. Go to the phone's Bluetooth® settings and remove the <init>_pair device from the paired devices list.

Note: This is necessary so that when you reset or re-program the kit, the phone won't have stale bonding information stored which could prevent you from re-connecting. In the next exercise we'll store bonding information on the Bluetooth® LE device so that you will be able to leave the devices paired if you desire.

Note: If you are using the Android version of AIROC™ Connect, you can remove bonding information by clicking on the "Paired" button next to your device's name rather than going to the phone's settings screen.

Exercise 2: Bonding

In this exercise, you will try out and review an application that saves bonding information to the flash. This exercise has been fully implemented in the template.

Note: *The application stores the bonding information using the kv-store library. That library supports using both internal and external flash. For the CYW920829M2EVK-02 kit, external flash is used since it does not have internal flash. For other kits, the internal flash is used. See the kv-store library documentation for details on its usage.*

Note: *The data stored in NV memory by the kv-store library is not erased when the device is reprogrammed so you must manually erase it if necessary. That can be done by calling the `mtb_kvstore_delete` function to remove individual key value pairs or by calling `mtb_kvstore_reset` to erase everything. The Templates directory has an application called `ch05_ex02_erase_nv` that can be used to completely erase the contents of the NV memory.*

User LED2 has been updated to indicate whether bonding information has been saved or not. The states are:

OFF	Not advertising
Slow Blink	Advertising, not bonded
Fast Blink	Advertising, bonded
ON	Connected

Once bonding information has been stored for a device, it can be erased from the device by entering [e] in the UART terminal window. This can only be done when the device is not connected. Erasing bonding information also causes advertising to restart if the device is not already advertising.

Application Creation



1. Create a new application with the BSP you are using.

On the application template page, use the **Browse** button to start from the template provided in *Templates/ch05_ex02_bond*.

Keep the name as the default of *ch05_ex02_bond*.



2. Open the Bluetooth® Configurator.



3. On the GAP Settings tab Change the device name to <init>_bond.



4. Save your edits and close the configurator.



5. Review the code to see how bonding information is stored and retrieved using the kv-store and serial-flash libraries.

Note: *An overview of the changes made from the prior application is provided later in this exercise.*

Testing

- ☐ 1. Open a UART terminal and connect to the kit.

Note: The retarget-io library is configured with flow control. This is necessary for the CYW920829M2EVK-02 kit to operate properly when using the UART for input. Therefore, you must keep a UART terminal open to prevent the Tx buffer from filling up, which may cause the firmware to hang.

- ☐ 2. Program your application to your kit.
- ☐ 3. Open the AIROC™ Connect app and connect to your device.
- ☐ 4. Watch the UART messages during the next steps to see which Bluetooth® events occur.
- ☐ 5. Connect to the device.
- ☐ 6. Connect to your device, open the GATT browser, click on the Service, and then on the BUTTON_COUNT Characteristic. Click "Read".
- ☐ 7. If requested, accept the invitation to pair the devices.
- ☐ 8. Note down the Stack events that occur during pairing. This information is displayed in the UART.
- ☐ 9. Disconnect from the device. Do NOT remove the device from the phone's list of paired devices this time.

Note: You will notice that the LED is blinking at 5 Hz. The firmware was written to do this when it is not connected but has bonding information stored.

- ☐ 10. Re-scan and find your device in the list.
- ☐ 11. Re-connect to your device and read the BUTTON_COUNT Characteristic.
- ☐ 12. Once again note down the Stack events that occur during pairing. You will notice that fewer steps are required this time.
- ☐ 13. Disconnect again.
- ☐ 14. Reset or power cycle the board.

Note: If you power cycle the board, you will need to either reset or re-open the UART terminal window.

- ☐ 15. Start a scan, find your device in the list, connect to your device for a third time and then read the BUTTON_COUNT Characteristic.
- ☐ 16. Note down the Stack events that occur this time during pairing. Compare to the previous two connections.
- ☐ 17. Disconnect again.
- ☐ 18. Remove the device from the list of bonded devices in the Phone's Bluetooth® settings or by clicking the "Paired" button in AIROC™ Connect (Android only).
- ☐ 19. Start a scan and find your device.
- ☐ 20. Connect to your device and try to read the BUTTON_COUNT Characteristic.

Note: Pairing will not complete because AIROC™ Connect no longer has the required keys to use. You will not be able to read the Counter value because it requires an authenticated connection.

Note: If you look in the UART window you will see a message about the security request being denied.

☐ 21. Disconnect from the device. Wait until the pairing request times out so that advertising restarts or just reset the kit.

☐ 22. Press **[e]** in the UART window to erase bonding information and reset the kit.

Note that `CYBSP_USER_LED` begins blinking at 2 Hz. This indicates that the bonding information has been cleared from the device and it will now allow a new connection.

☐ 23. Scan, Connect, and attempt to read the `BUTTON_COUNT` Characteristic again. Allow pairing if requested. This time it should work.

☐ 24. Note the steps that the firmware goes through this time.

☐ 25. Disconnect from the device.

☐ 26. Press **[e]** in the UART window to erase bonding information from the kit.

☐ 27. Remove the device from the phone's paired Bluetooth® devices so that the saved bonding information won't interfere with any future tests.

Note: You should clear the bonding information from both the kit and phone whenever you complete an exercise to that it does not interfere with later exercises.

Overview of Changes

- There are a lot of messages printed in this example for learning purposes. In a real application, most if not all these messages would be removed. For example, the keys are printed to the UART at powerup. That would never be done in a real application.
- The *kv-store* and *serial-flash* libraries are added as dependencies. The header files and initializations are included in *main.c*. The *serial-flash* library is only used when accessing external flash.
- The required *kv-store* functions for internal flash (using the HAL driver) are provided in *app_kv-store_internal.c*. The required *kv-store* functions for external flash (using *serial-flash*) are provided in *app_kv-store_external.c*. The header file *app_kv-store_common.h* is used for both cases. The selection between internal and external flash is made in the application's Makefile based on the BSP name. The functions in these files are based on the Bluetooth LE Hello Sensor code example.
- The header file for the FreeRTOS Queue functions is included in *main.c*.
- A global variable called `bond_mode` is created that can either be set to `BONDED` or `BONDING` to indicate the device's bonding status.
- The status LED is updated to indicate connection status. The PWM operates the LED as follows:

Advertising?	Connected?	Bonded?	LED
No	No	N/A	OFF
No	Yes	N/A	ON
Yes	No	No	Blinking at 2 Hz

Advertising?	Connected?	Bonded?	LED
Yes	No	Yes	Blinking at 5 Hz
Yes	Yes	N/A	N/A - this case doesn't occur

- Global variables `link_keys`, `cccd`, and `identity_keys` are created to hold the link keys for the bonded device (which also includes the BDA of the bonded device), the value of the Button CCCD, and the identity keys (which are used for RPA) respectively. The BDA of the bonded device is used to determine when we have reconnected to the same device while the CCCD value is saved so that the state of notifications can be retained across connections for bonded devices.
- In `BTM_ENABLED_EVENT`, before initializing the GATT database, the NV memory is examined to see if data is available for a bonded device. If there is, the remote device is added to the address resolution database and `bond_mode` is set to `BONDED`.
- In `BTM_SECURITY_REQUEST_EVENT` look to see if `bond_mode` is `BONDING`. Security is only granted if a device is not already bonded.
- In `BTM_PAIRING_COMPLETE_EVT`, if bonding was successful set `bonded` to `TRUE`.
- In `BTM_ENCRYPTION_STATUS_EVT`, if the device is bonded (i.e. `bond_mode` is `BONDED`), set the CCCD value to what was read from NV memory during `BTM_ENABLED_EVENT`.
 - This reads `bondinfo` upon a subsequent connection when devices were previously bonded.
- In `BTM_PAIRING_DEVICE_LINK_KEYS_UPDATE_EVT`, save the keys for the peer device to flash.
- In `BTM_PAIRING_DEVICE_LINK_KEYS_REQUEST_EVT`, read the keys for the peer device from flash. If no keys have been stored, return `WICED_BT_ERROR` so the stack will generate new keys.
- In `BTM_LOCAL_IDENTITY_KEYS_UPDATE_EVT`, save the identity keys used for RPA to flash.
- In `BTM_LOCAL_IDENTITY_KEYS_REQUEST_EVT`, read the identity keys used for RPA from flash. If no keys have been stored, return `WICED_BT_ERROR` so the stack will generate new keys.
- In the GATT connect event handler, for a disconnection, reset the CCCD to 0 so that if a new device is bonded it will not start out with notifications enabled.
- In the GATT write handler function, save the Button CCCD value to the `cccd` variable whenever it is updated and write the value into flash.
- The UART is configured to accept input with a receive callback. This is done by reconfiguring the object that is created by the `retarget-io` library. The `rx_cback` function sends the received character to a UART task using an RTOS queue. The UART task is necessary because you cannot call any Bluetooth® Stack functions from inside the ISR.
- The initialization function for the `retarget-io` library is changed to `cy_retarget_io_init_fc` so that flow control is enabled. This is necessary for the CYW920829M2EVK-02 to work properly because flow control is enabled on the KitProg on that kit. You should keep a UART terminal open when using the application so that output messages don't fill up the UART Tx buffer.
- The UART task looks for the key `[e]`. If it has been sent, it sets `bond_mode` to `BONDING`, removes the bonded device from the list of bonded devices, removes the device from the address resolution database, and clears out the `link_keys` and `cccd` values stored in flash.
- Because the UART does not function during Deep Sleep, the device configurator was used to set the PSoC™ 6 **System > Power > System Idle Power Mode** to **CPU Sleep**. This is done by providing a custom *design.modus* file for each supported BSP inside the application's *templates* directory. The correct

design.modus file is copied into the BSP's *config* directory during application creation. Template processing is discussed in more detail in the Level 1 Getting Started class.

- `Aprint_array` function is created to simplify printing out arrays of values such as keys.

Questions

Here are a few questions to answer to make sure you understand how the application works.

☐

1. What items are stored in flash?

☐

2. Which event stores each piece of information?

☐

3. Which event retrieves each piece of information?

Exercise 3: Passkey

In this exercise, you will modify the previous exercise to require a passkey to be entered to pair the devices the first time. The passkey will be randomly generated by the stack and will be displayed by the UART. The passkey will need to be entered in your Phone's Bluetooth® connection settings before Pairing/Bonding will be allowed.

Application Creation

- ☐ 1. Create a new application with the BSP you are using.
- ☐ 2. On the application template page, use the **Browse** button to start from the completed application for the previous exercise, `ch05_ex02_bond`. If you did not complete that exercise, you can use the template for it from `Templates/ch05_ex02_bond`.
Name the new application `ch05_ex03_passkey`.
- ☐ 3. Open the Bluetooth Configurator.
 - a. Change the **Device Name** to `<init>_passkey`
 - b. Save the edits and close the configurator.
- ☐ 4. Open `main.c` and change the `pairing_io_capabilities_ble_request.local_iop_cap` from `BTM_IO_CAPABILITIES_NONE` to `BTM_IO_CAPABILITIES_DISPLAY_ONLY`.
- ☐ 5. Create a new event case statement for `BTM_PASKEY_NOTIFICATION_EVT` in `app_bt_management_callback` to print the passkey value provided by the stack to the UART. Be sure to set `result = WICED_BT_SUCCESS` inside the case.

Note: `BTM_PASKEY_NOTIFICATION_EVT` is not in the template code.

Note: Make sure you print something such as a string of asterisks around the value so that it is easy to find in the terminal window.

Note: The passkey is 6 digits so print leading 0's if the value is less than 6 digits. (i.e., use `%06d`).

Note: The passkey is passed to the callback event as:
`p_event_data->user_passkey_notification.passkey`

Testing

- ☐ 1. Open a UART terminal window.

Note: The `retarget-io` library is configured with flow control. This is necessary for the CYW920829M2EVK-02 kit to operate properly when using the UART for input. Therefore, you must keep a UART terminal open to prevent the Tx buffer from filling up, which may cause the firmware to hang.
- ☐ 2. Build the application and program your kit.
- ☐ 3. Open the AIROC™ Connect app.
- ☐ 4. Attempt to connect to the device and then navigate down to the `BUTTON_COUNT` Characteristic in the GATT browser.

You will see a notification from the Bluetooth system asking for the Passkey to be entered. Find the Passkey on the UART terminal window and enter it into the device.

☐

5. Once Pairing and Bonding completes, verify that the application still works.

☐

6. Disconnect, reconnect and read the `BUTTON_COUNT` Characteristic again.

Observe that the key does not need to be entered to Pair this time.

☐

7. Disconnect, then remove the device from the phone's remembered Bluetooth device settings or by clicking the "Paired" button in AIROC™ Connect (Android only).

☐

8. Press [**e**] in the UART terminal to put the kit into Bonding mode (i.e. erase the stored bonding information) and then reconnect.

Read the `BUTTON_COUNT` Characteristic and observe that the key must be entered again.

☐

9. Disconnect again.

☐

10. Remove the bonding information from the kit.

☐

11. Remove the bonding information from the phone's Bluetooth settings or by clicking the "Paired" button in AIROC™ Connect (Android only).

Questions

Here are some questions to answer to test your understanding.

☐

1. Other than `BTM_IO_CAPABILITIES_NONE` and `BTM_IO_CAPABILITIES_DISPLAY_ONLY`, what other choices are available? What do they mean?

☐

2. What additional stack callback event occurs compared to the previous exercise? At what point does it get called?

Exercise 4: Numeric comparison

In this exercise, you will modify the previous exercise to require the user to compare a 6-digit number on both devices to pair the first time. After comparing that both numbers are the same, the user needs to click **Yes** in the Phone's Bluetooth connection settings and type [y] in the UART terminal window for the kit before Pairing/Bonding will be allowed.

Note: The code to implement the passkey will not be removed since it is up to the central to decide which method to use. That is, we will set the device capabilities to `BTM_IO_CAPABILITIES_DISPLAY_AND_YES_NO_INPUT`. Since the device has both a display and Yes/No input, the central can choose to use either passkey or numeric comparison.

Application Creation



1. Create a new application with the BSP you are using.

On the application template page, use the **Browse** button to start from the completed application for the previous exercise `ch05_ex03_passkey`. If you did not complete that exercise, the solution can be found in `Projects/key_ch05_ex03_passkey`.

Name the new application `ch05_ex04_numeric`.



2. Open the Bluetooth Configurator.
 - a. Change the **Device Name** to `<init>_numeric`
 - b. Save the edits and close the configurator.



3. Open `main.c` and change the `pairing_io_capabilities_ble_request.local_iop_cap` to `BTM_IO_CAPABILITIES_DISPLAY_AND_YES_NO_INPUT`.



4. Add a global variable of type `wiced_bt_device_address_t` called `tempRemoteBDA` to temporarily hold the remote BDA. This address will be needed when the user confirms the connection request.



5. Create a new event case statement for `BTM_USER_CONFIRMATION_REQUEST_EVT` in `app_bt_management_callback` to do the following:
 - a. print the value of the numeric code that is provided by the stack.
 - b. Use `memcpy` to save the remote BDA in `tempRemoteBDA`.
 - c. Set `result = WICED_BT_SUCCESS`.

Note: `BTM_USER_CONFIRMATION_REQUEST_EVT` is not in the template code.

Note: Make sure you print something such as a string of asterisks around the value so that it is easy to find in the terminal window.

Note: The numeric code is 6 digits so print leading 0's if the value is less than 6 digits. (i.e. use `%06d`).

*Note: The numeric code is passed to the callback event as:
`p_event_data->user_confirmation_request.numeric_value`*

*Note: The remote_bda is passed to the callback event as:
`p_event_data->user_confirmation_request.bd_addr`*



6. In the UART task, add cases for a 'y' or an 'n' character and send the appropriate response back to the stack.

If the response is 'y' send `WICED_BT_SUCCESS` and if the response is 'n' send `WICED_BT_ERROR`. You must provide the BDA of the remote device which was captured in the `BTM_USER_CONFIRMATION_REQUEST_EVT`. For example:

```
wiced_bt_dev_confirm_req_reply( WICED_BT_SUCCESS, tempRemoteBDA );  
wiced_bt_dev_confirm_req_reply( WICED_BT_ERROR, tempRemoteBDA );
```

Testing



1. Open a UART window.

Note: The retarget-io library is configured with flow control. This is necessary for the CYW920829M2EVK-02 kit to operate properly when using the UART for input. Therefore, you must keep a UART terminal open to prevent the Tx buffer from filling up, which may cause the firmware to hang.



- 2.



3. Build the application and program your kit.



4. Open the AIROC™ Connect app.



5. Attempt to Connect and then Pair to the device.

You will see a notification on the phone asking for you to verify the number printed by both devices is the same. Find the number on the UART terminal window. If the values match, click **Yes** on the phone, and press [y] in the UART.



6. Once Pairing and Bonding completes, verify that the application still works.



7. Disconnect, reconnect and read the `BUTTON_COUNT` Characteristic.



8. Observe that the number does not need to be verified to pair this time.



9. Disconnect, then remove the device from the phone's remembered Bluetooth® device settings or by clicking the "Paired" button in AIROC™ Connect (Android only).



10. Enter [e] in the UART window to put the kit into Bonding mode and then reconnect.



11. Connect again and read the `BUTTON_COUNT` Characteristic. Observe that the numeric comparison must be done again to connect.



12. Disconnect again.



13. Remove the bonding information from the kit.



14. Remove the bonding information from the phone's Bluetooth settings or by clicking the "Paired" button in AIROC™ Connect (Android only).

Exercise 5: Store Bonding Information for Multiple Devices

In this exercise, you will review and test an application that has all the features of the previous exercise, but it will allow up to 4 devices to be bonded. Note that while this application stores bonding information for multiple devices, it does NOT allow multiple simultaneous Bluetooth LE connections. That is also possible, but it's not demonstrated here.

Each new device that is bonded has its information stored in a separate flash location. When a connection is made, the device will search to see if the device is one of the stored devices.

When the application starts, it will be in bonding mode (just like the earlier applications). Once at least one device is bonded, the [b] key in the UART is used to toggle between bonding mode and bonded mode. You must enter bonding mode to bond a new device and you must be in bonded mode to connect to an existing device. If you have an existing device and delete its bonding information from the client, you can re-bond to it by entering bonding mode and then connecting to the device. This will replace the existing bonding entry for that device. When you change the mode, if advertising has stopped, it will be restarted.

If you try to bond a new device and all the available slots are already taken, the bonding information for the oldest device will be removed and then replaced with the new device.

The [l] key in the UART can be used to print information on the number of bonded devices, the next flash location that will be used to store bonding information, and the list of BDAs for the bonded devices.

The [E] key in the UART can be used to erase bonding information for all devices.

The [?] key in the UART will list the available commands.

Application Creation



1. Create a new application with the BSP you are using.

On the application template page, use the **Browse** button to start from the template provided in *Templates/ch05_ex06_multi*.

Keep the name as the default of *ch05_ex06_multi*.



2. Open the Bluetooth® Configurator.



3. On the GAP Settings tab Change the device name to <init>_multi.



4. Save your edits and close the configurator.



5. Review the code to see how bonding information is stored and retrieved using the kv-store and serial-flash libraries.

Note: An overview of the changes made from the prior application is provided later in this exercise.

Testing

- ☐ 1. Open a UART terminal and connect to the kit.

Note: The retarget-io library is configured with flow control. This is necessary for the CYW920829M2EVK-02 kit to operate properly when using the UART for input. Therefore, you must keep a UART terminal open to prevent the Tx buffer from filling up, which may cause the firmware to hang.

- ☐ 2.
- ☐ 3. Program your application to the kit.
- ☐ 4. Open the AIROC™ Connect mobile app.
- ☐ 5. Scan for your device and connect to it.
- ☐ 6. Open the GATT browser, go to the BUTTON_COUNT Characteristic, and click Read. Complete pairing by entering [y] in the UART and accepting pairing on the phone.
- ☐ 7. Test the functionality of the application to make sure it still works the same way.
- ☐ 8. Disconnect from the device.
- ☐ 9. Press the [l] key in the UART window to see that one device is bonded.
- ☐ 10. Press the [b] key in the UART window to put the device into bonding mode so that a second device can be bonded.
- ☐ 11. Use a different phone to open AIROC™ Connect.
- ☐ 12. Scan for your device and connect to it.
- ☐ 13. Open the GATT browser, go to the BUTTON_COUNT Characteristic, and click Read. Complete pairing by entering [y] in the UART and accepting pairing on the phone.
- ☐ 14. Test the functionality of the application to make sure it still works the same way.
- ☐ 15. Disconnect from the device.
- ☐ 16. Press the [l] key in the UART window to see that two devices are now bonded.
- ☐ 17. Scan and connect again from both AIROC™ Connect clients to verify that bonding occurs without requiring you to do the numeric comparison. Verify that the functionality still works.
- ☐ 18. Disconnect and then remove bonding information from AIROC™ Connect on one of the phones.
- ☐ 19. Connect and then attempt to pair from the device and see that it fails because the device is not in bonding mode and the client does not have the bonding information available. Disconnect again.
- ☐ 20. Wait for advertising to restart (the bonding request must timeout first) and then press the [e] key in the UART window to put the device into bonding mode so that the prior device can be re-bonded.
- ☐ 21. Reconnect and pair. Notice that you are now required to do numeric comparison again since the bonding information is no longer available on the client.
- ☐ 22. Disconnect again and press [l] in the UART. Notice that the bonding information has been updated in the existing flash slot rather than bonding a new device.

- ☐ 23. Disconnect again. Connect and pair additional devices if you desire to see them stored in unique locations. If you bond more than 4 devices, the oldest bonded device will be replaced.
- ☐ 24. Disconnect again and clear the bonding information from all devices.

Overview of Changes

- A `#define` is added for `BOND_MAX` which is the maximum number of devices that can be bonded at a time (default is set to 4).
- A `#define` is added for `NONE` to use when a device does not have bonding information stored yet.
- The `link_keys` and `cccd` variables are updated to arrays that can hold information for up to `BOND_MAX` devices.
- Variables `num_bonded` and `next_slot` are added to keep track of the number of devices currently bonded and the next location (slot) to use for new bonding information respectively. Their values will be stored in NV memory.
- A global variable `current_slot` is added to keep track of which device from the saved bonded devices list is currently connected.
- A global variable `new_device` is added to keep track of whether a device being bonded is a new device or is replacing an existing device's information.
- The UART task is updated so that it just toggles whether it is in bonding mode or not when [**b**] is pressed. This can only be done when not connected.
- The UART task is updated so that when [**l**] is pressed, it will list information for the bonded devices.
- The UART task is updated so that when [**E**] is pressed bonding information for all devices is removed.
- The UART task is updated so that when [**?**] is pressed a list of commands is printed.
- Update the `BTM_ENABLED_EVT` to read information from all previously bonded devices and add them to the address resolution database.
- Update `BTM_ENCRYPTION_STATUS_EVT` to update the CCCD information in the GATT database from the bonded device's last saved information.
- Update `BTM_PAIRING_DEVICE_LINK_KEYS_REQUEST_EVT`:
 - Search for the device in the list of bonded devices:
 - If found, set the `current_slot` variable to the location containing the connected device's information and provide keys for that slot to the stack.
 - If not found, return `WICED_BT_ERROR` so new keys will be generated.
- Update `BTM_PAIRING_DEVICE_LINK_KEYS_UPDATE_EVT` so that it stores the newly bonded device's encryption key information into the correct array location in flash. It also increments the number of bonded devices and the next slot location to use if this is a new device being bonded.
- Update the GATT `app_bt_write_handler` function so that it stores any changes to the CCCD value to the correct slot location.

5.6 Appendix

Answers to the questions asked in the exercises above are provided here.

5.6.1 Exercise 2 Answers

1. What items are stored in flash?

Identity keys for the device (`identity_keys`)

Security keys for the bonded device (`link_keys`)

CCCD value for the bonded device (`cccd`)

2. Which event stores each piece of information?

The identity keys are stored by `BTM_LOCAL_IDENTITY_KEYS_UPDATE_EVT`.

The link keys are stored by `BTM_PAIRED_DEVICE_LINK_KEYS_UPDATE_EVT` when a device is bonded.

The CCCD value is stored in the GATT write handler when a new CCCD value is written and by `BTM_PAIRED_DEVICE_LINK_KEYS_UPDATE_EVT` when a device is bonded.

3. Which event retrieves each piece of information?

The variable `identity_keys` is read during `BTM_LOCAL_IDENTITY_KEYS_REQUEST_EVT` to determine if new keys must be generated.

The variable `link_keys` is read during `BTM_ENABLED_EVT` so that it can be added to the address resolution database and during `BTM_PAIRED_DEVICE_LINK_KEYS_REQUEST_EVT` to provide keys to the stack for a previously bonded device.

The variable `cccd` is read during `BTM_PAIRED_DEVICE_LINK_KEYS_REQUEST_EVT` so that a previously connected device retains its CCCD setting when reconnected.

5.6.2 Exercise 3 Answers

1. Other than `BTM_IO_CAPABILITIES_NONE` and `BTM_IO_CAPABILITIES_DISPLAY_ONLY`, what other choices are available? What do they mean?

`BTM_IO_CAPABILITIES_DISPLAY_AND_YES_NO_INPUT`

The device can display values and can accept a Yes/No input

`BTM_IO_CAPABILITIES_KEYBOARD_ONLY`

The device cannot display values, but has a way for the user to enter numeric values

`BTM_IO_CAPABILITIES_BLE_DISPLAY_AND_KEYBOARD_INPUT`

The device can display values and has a way for the user to enter numeric values

2. What additional stack callback event occurs compared to the previous exercise? At what point does it get called?

`BTM_PASKEY_NOTIFICATION_EVT`

This event is called when the Stack has generated a 6-digit value that needs to be displayed to the user.

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Published by
Infineon Technologies AG
81726 Munich, Germany

© 2024 Infineon Technologies AG.
All Rights Reserved.

IMPORTANT NOTICE

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics ("Beschaffenhheitsgarantie").

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer's compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer's products and any use of the product of Infineon Technologies in customer's applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (www.infineon.com).

WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.