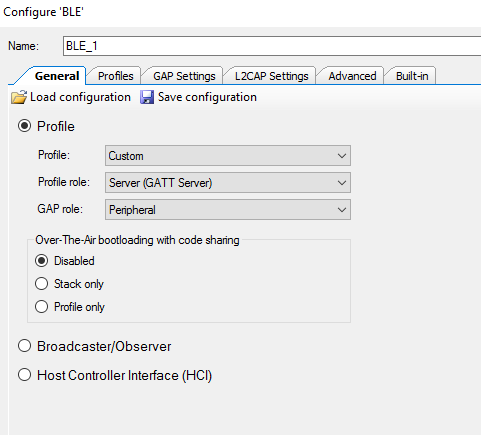
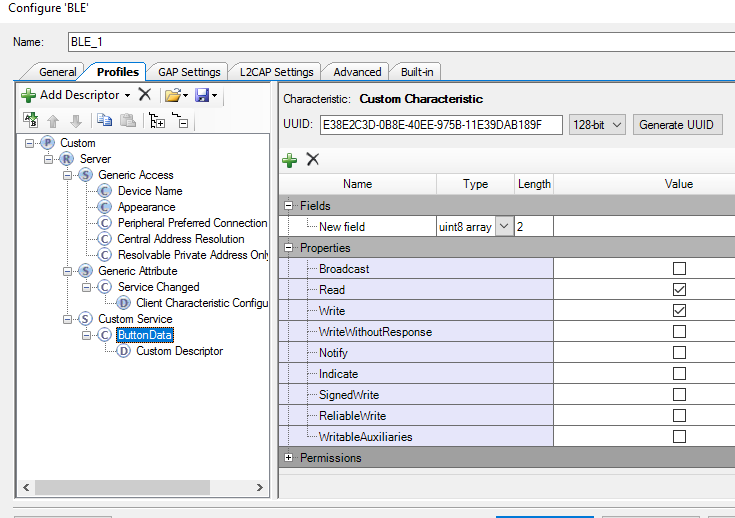
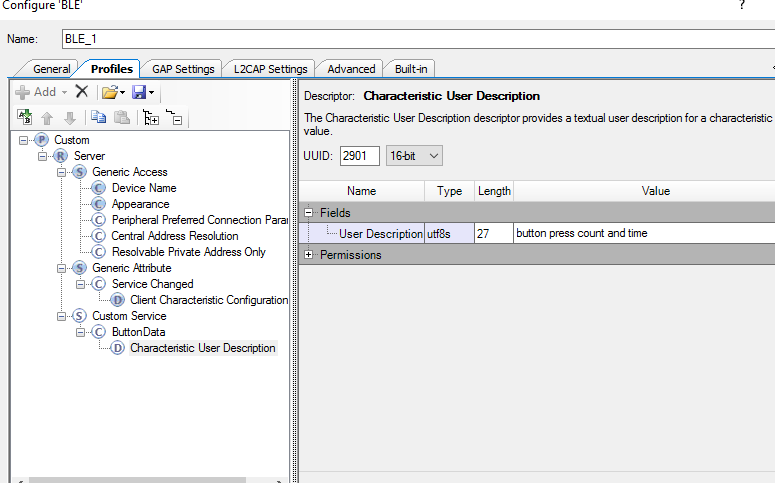
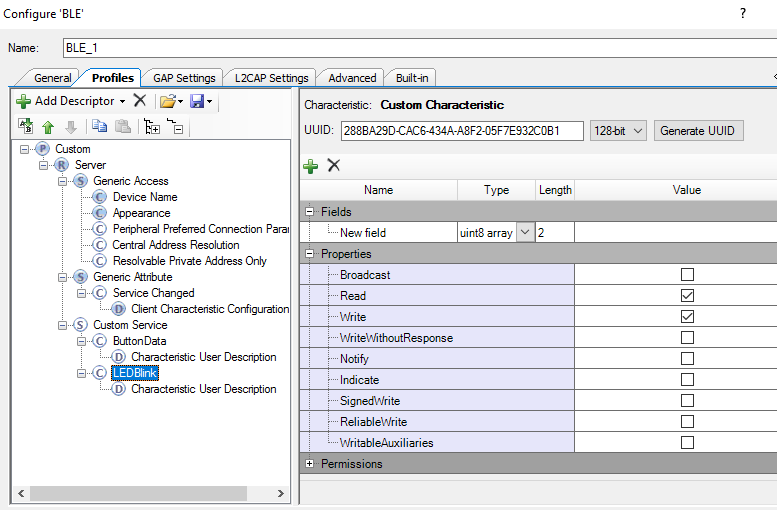
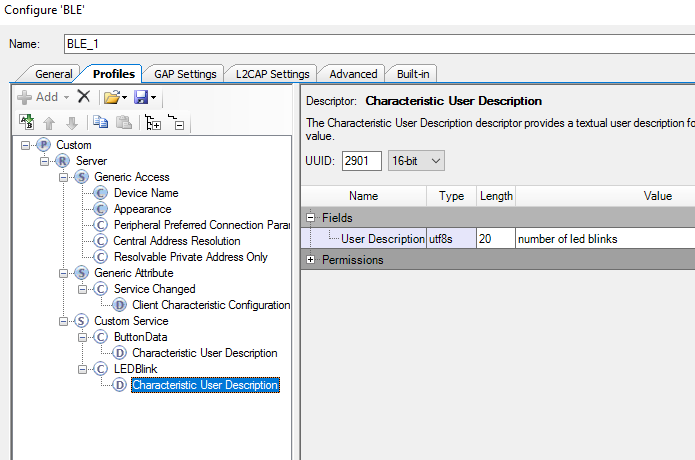
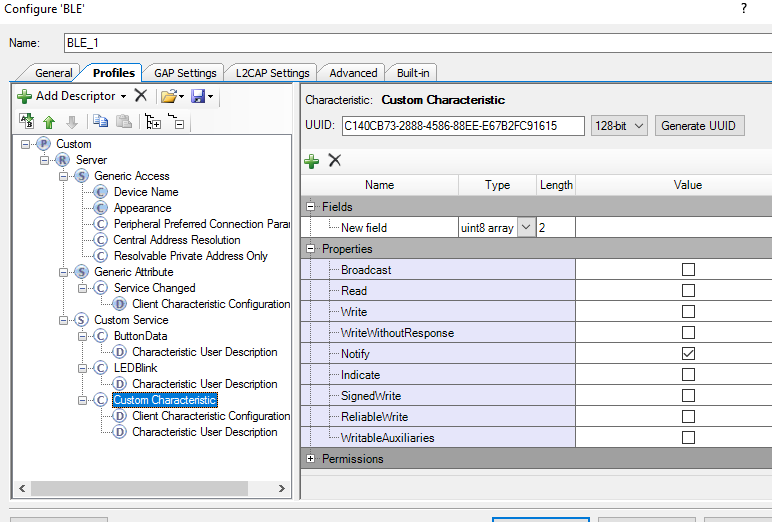
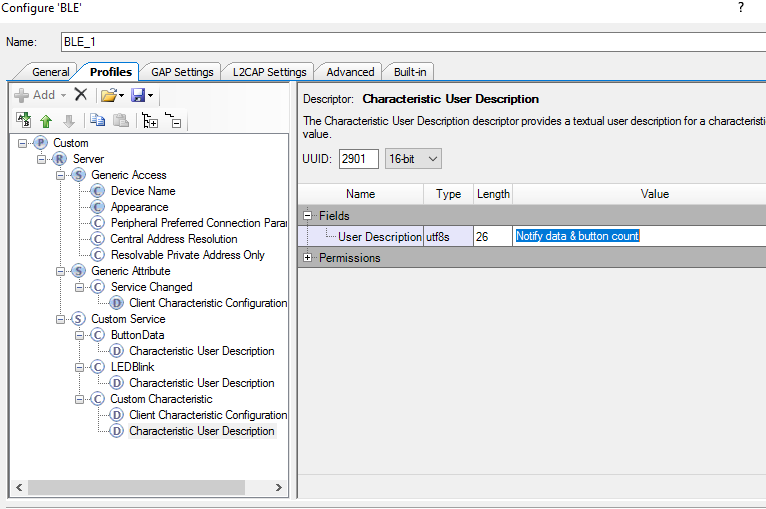
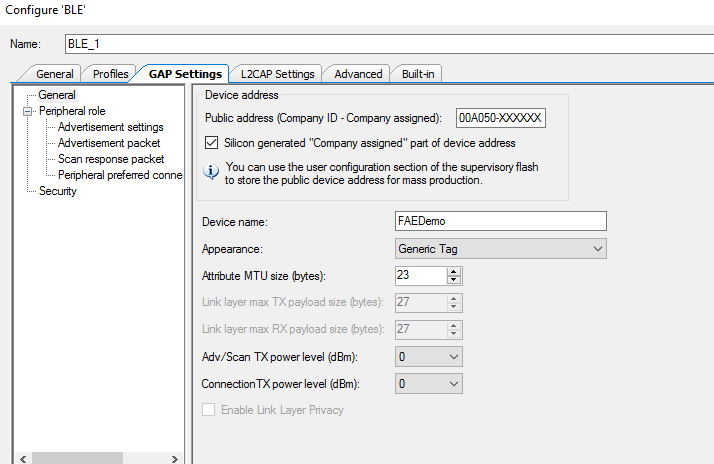
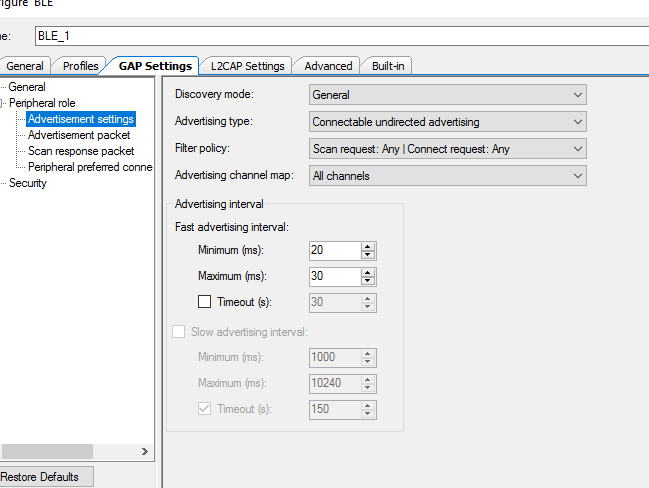
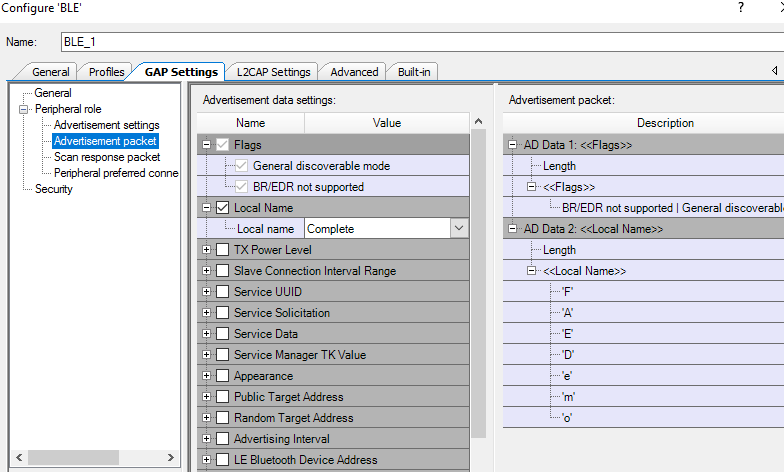
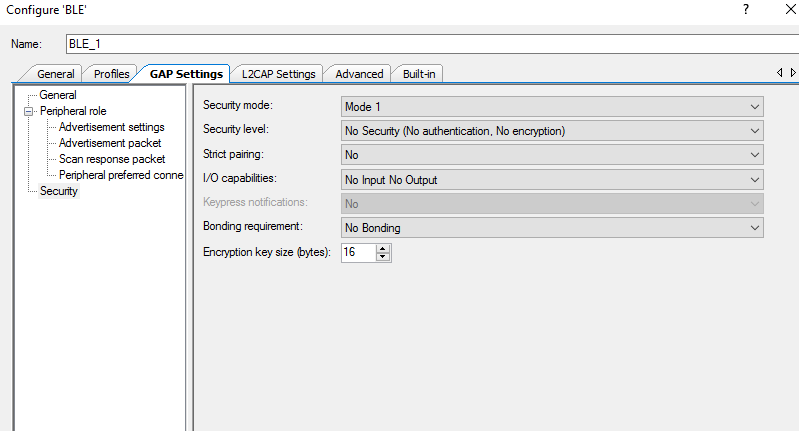
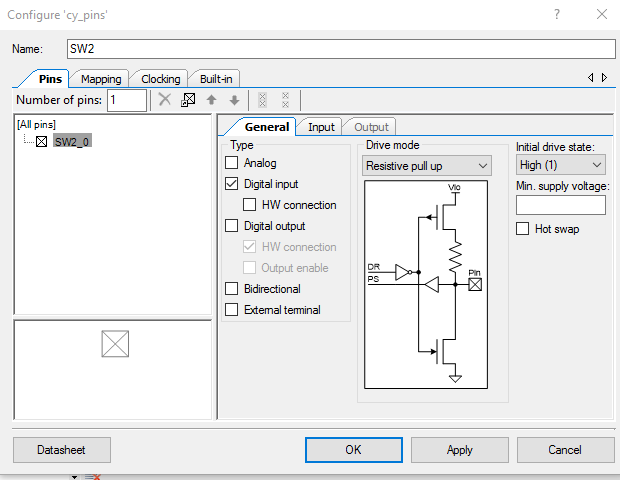
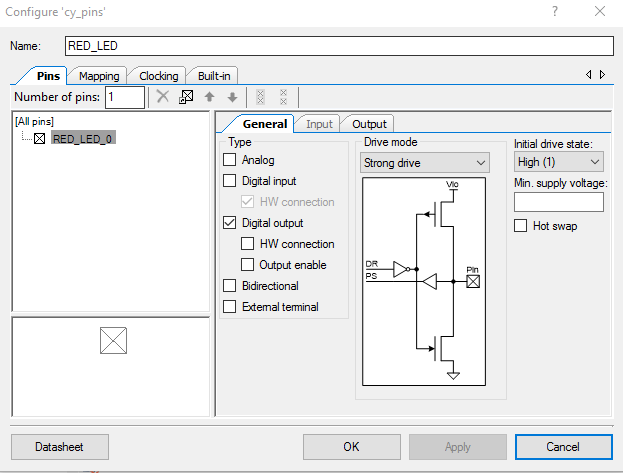
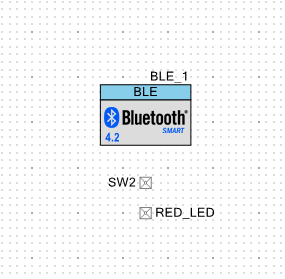
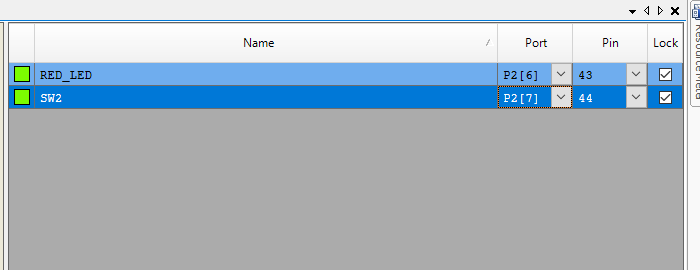
This document covers how to create a custom project in PSoC Creator for the PSoC 4 BLE devices.

The project will count button presses, blink an LED, and have read, write and notify characteristics.

**Create**

1. Open PSoC Creator, create new project, select target kit (CY8CKIT-042-BLE), next, empty schematic, name the workspace FAEDemo, finish.
2. Pull the BLE component in to the schematic, double click to configure. On the general tab, select the following settings:
   1. 
3. On the profiles tab, rename the Custom Characteristic to ButtonData, set the field to be uint8 array of size 2, and enable read and write properties:
   1. 
4. Delete the custom descriptor and add a Characteristic User Descriptor, with text explaining what it shows
   1. 
5. Add another characteristic to the Custom service, and rename this to LEDBlink. Give this the same properties and array size as the ButtonData, and another CUD with appropriate information. It should look like this:
   1. 
   2. 
6. We now need to add a 3rd characteristic for the Notify characteristic. This has different requirements to the 2 others, and it also needs a CCCD to allow us to enable or disable notifications. The configuration should look like this:
   1. 
   2. 
7. In the GAP Settings tab, we need to define a Device Name – FAEDemo, and set the appearance to Generic tag. In the advertisement settings, disable timeout in fast advertising mode. In the advertisement packet, make sure local name is ticked, and in the security tab, set I/O capabilities to No Input No Output, and bonding to No Bonding. The configuration should look like this:
   1. 
   2. 
   3. 
   4. 
8. This is the BLE configuration complete. Press apply, and then OK. We should then Build the project to generate all the configuration files we will need
9. We now need to add some inputs and outputs for our button and LED. First pull a Digital Input pin on to the schematic. Double click to configure. Rename the pin to SW2, change the drive mode to Resistive pull up (our board switch is tied to ground when closed), and disable the HW Connection:
   1. 
10. Next we need to add a Digital Output pin for our LED. Rename this as RED\_LED, set the drive mode to Strong Drive, disable the HW connection, and set the initial drive state to 1 (our LEDs are active low):
    1. 
11. Our schematic should now look like this:
    1. 
12. Next go to CYWDR, and assign SW2 to P2.7 and RED\_LED to P2.6. Then rebuild the project.
    1. 
13. We can now start our code development. Unlike WICED no code callbacks or functions are auto generated so we need to create this from scratch.
14. First we will create a function to blink the RED\_LED xx times. We start by declaring the function and then we will create it:

void LEDBlink(uint8 val);

//Function Name: LEDBlink

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Summary:

\* function to blink an led xx times

\*

\* Parameters:

\* val: number of blinks

\*

\* Return:

\* void

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//function to blink an led xx times

void LEDBlink(uint8 val)

{

uint8 i;

for(i=0; i<val;i++) //run the cycle as many times as the valu passed in to it

{

RED\_LED\_Write(0); //turn LED on

CyDelay(250);//delay 250ms

RED\_LED\_Write(1);//turn LED off

CyDelay(250);//delay 250ms

CyBle\_ProcessEvents();//in case we get stuck in a long loop process events to make sure we don't miss anything

}

}

1. This function receives a value and then writes a 0 then a 1 to the RED\_LED with a delay between the 2 writes. We will use this when the BLE Central writes to the device. We include a command to process BLE events inside this function so that if we get a large number of blinks, we don’t miss any BLE commands by being stuck in this blocking function.
2. In our main function we need to start the BLE component and register a callback to deal with BLE Stack and characteristic events. The code to do this is :

CyBle\_Start(CustomEventHandler);//start the ble component and register the custom callback

1. Once we have that we need to actually develop the callback function. This requires a number of variables and structures to store the necessary data. This includes a connection id handle, notification cccd structure, handles for the ledblink and button data, as well as structures for read and write requests. It also includes local variables for sending notifications and to store the current cccd value, alongside local count and delay variables and local data arrays. This code needs to be inserted before the callback at the top of main.c

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Variable Declarations\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 'connectionHandle' stores connection parameters \*/

CYBLE\_CONN\_HANDLE\_T connectionHandle;

/\*This flag is set when the Central device writes to CCCD of the enable notifications \*/

uint8 sendNotifications = 0;

/\* Local variable to store the current CCCD value \*/

uint8 NotifyCCCDvalue[2];

/\* Handle value to update the CCCD \*/

CYBLE\_GATT\_HANDLE\_VALUE\_PAIR\_T NotificationCCCDhandle;

//handles to create database arrays for the read and write data

CYBLE\_GATT\_HANDLE\_VALUE\_PAIR\_T LEDBlinkHandle;

CYBLE\_GATT\_HANDLE\_VALUE\_PAIR\_T ButtonDataHandle;

CYBLE\_GATTS\_WRITE\_REQ\_PARAM\_T \*wrReqParam;

CYBLE\_GATTS\_CHAR\_VAL\_READ\_REQ\_T \*rdReqParam;

uint8 Count=0; //variable to count the number of switch presses

uint16 DelayVal=0; //variable used to implement a delay between sending notifications

uint8 LEDBlinkData[2];//local variable to store led blink data

uint8 ButtonData[2];//local variable used to store button press data to be read

uint8 NotifyData[2];//local variable to store notifucation data

1. The callback itself deals with all the possible stack events that could occur. It is responsible for starting advertising, restarting should it stop and be disconnected, storing the connection handle on connection amongst other things. The main changes we need to implement relate a GATT disconnection, a read request and a write request. For a GATT disconnection we disable notifications, zero the notification data, write this back to the GATT DB. The GUI configuration creates a list of declared handles and indices which can be found in BLE\_1\_custom.h (where BLE\_1 is the component name in the GUI).
2. For the write request, we have to interrogate which characteristic was written to. If it was the notification CCCD we check if the received value was 1 or 0 and enable and disable notifications, and then store the data in the GATT DB. If it was to the LEDBlink characteristic, we copy the data to our local array, update the GATT DB and then use the first byte to blink the RED LED x times. If we receive a read request for the Button Data characteristic, we ensure the current value of the counter is loaded in the first byte, and that the second byte is incremented so that we can track the number of read requests. We then write this data to the GATT DB so that it can respond with the current data.
3. The entire callback is here:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function Name: CustomEventHandler

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Summary:

\* Call back event function to handle varios events from BLE stack

\*

\* Parameters:

\* event: event returned

\* eventParam: link to value of the events returned

\*

\* Return:

\* void

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void CustomEventHandler(uint32 event, void \* eventParam)

{

switch(event)

{

case CYBLE\_EVT\_STACK\_ON:

/\* This event is received when component is Started \*/

//funcion call to start advertsing in fast mode

CyBle\_GappStartAdvertisement(CYBLE\_ADVERTISING\_FAST);

break;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* GAP Events

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

case CYBLE\_EVT\_GAPP\_ADVERTISEMENT\_START\_STOP:

/\* If the current BLE state is Disconnected, then the Advertisement

\* Start Stop event implies that advertisement has stopped \*/

if(CyBle\_GetState() == CYBLE\_STATE\_DISCONNECTED)

{

CyBle\_GappStartAdvertisement(CYBLE\_ADVERTISING\_FAST);//if we stopped and we aren't connected restart advertising

}

break;

case CYBLE\_EVT\_GAP\_DEVICE\_CONNECTED:

/\* This event is received when device is connected over GAP layer \*/

break;

case CYBLE\_EVT\_GAP\_DEVICE\_DISCONNECTED:

/\* This event is received when device is disconnected \*/

/\*restart advertising on disconnection\*/

CyBle\_GappStartAdvertisement(CYBLE\_ADVERTISING\_FAST);

break;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* GATT Events

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

case CYBLE\_EVT\_GATT\_CONNECT\_IND:

/\* This event is received when device is connected over GATT level \*/

/\* Update attribute handle on GATT Connection\*/

connectionHandle = \*(CYBLE\_CONN\_HANDLE\_T \*)eventParam;

break;

case CYBLE\_EVT\_GATT\_DISCONNECT\_IND:

/\* This event is received when device is disconnected \*/

/\* Reset notification flag to prevent further notifications

\* being sent to Central device after next connection. \*/

sendNotifications = 0;

NotifyData[0]=0;//reset the notification data

/\* Reset the CCCD value to disable notifications \*/

/\* Write the present notification status to the local variable \*/

NotifyCCCDvalue[0] = sendNotifications;

NotifyCCCDvalue[1] = 0x00;

/\* Update CCCD handle with notification status data\*/

NotificationCCCDhandle.attrHandle = CYBLE\_CUSTOM\_SERVICE\_NOTIFYDATA\_CLIENT\_CHARACTERISTIC\_CONFIGURATION\_DESC\_HANDLE;

NotificationCCCDhandle.value.val = NotifyCCCDvalue;

NotificationCCCDhandle.value.len = sizeof(NotifyCCCDvalue);

/\* Report data to BLE component for sending data when read by Central device \*/

CyBle\_GattsWriteAttributeValue(&NotificationCCCDhandle, 0, &connectionHandle, CYBLE\_GATT\_DB\_PEER\_INITIATED);

break;

case CYBLE\_EVT\_GATTS\_WRITE\_REQ:

/\* This event is received when Central device sends a Write command on an Attribute \*/

wrReqParam = (CYBLE\_GATTS\_WRITE\_REQ\_PARAM\_T \*) eventParam;

/\* When this event is triggered, the peripheral has received a write command on the custom characteristic \*/

/\* Check if command is for correct attribute and update the flag for sending Notifications \*/

if(CYBLE\_CUSTOM\_SERVICE\_NOTIFYDATA\_CLIENT\_CHARACTERISTIC\_CONFIGURATION\_DESC\_HANDLE == wrReqParam->handleValPair.attrHandle)

{

/\* Extract the Write value sent by the Client for Notify CCCD \*/

if(wrReqParam->handleValPair.value.val[CYBLE\_CUSTOM\_SERVICE\_NOTIFYDATA\_CLIENT\_CHARACTERISTIC\_CONFIGURATION\_DESC\_INDEX] == 1)

{

sendNotifications = 1;

}

else if(wrReqParam->handleValPair.value.val[CYBLE\_CUSTOM\_SERVICE\_NOTIFYDATA\_CLIENT\_CHARACTERISTIC\_CONFIGURATION\_DESC\_INDEX] == 0)

{

sendNotifications = 0;

NotifyData[0]=0;//reset the data to start at 0

}

/\* Write the present notification status to the local variable \*/

NotifyCCCDvalue[0] = sendNotifications;

NotifyCCCDvalue[1] = 0x00;

/\* Update CCCD handle with notification status data\*/

NotificationCCCDhandle.attrHandle = CYBLE\_CUSTOM\_SERVICE\_NOTIFYDATA\_CLIENT\_CHARACTERISTIC\_CONFIGURATION\_DESC\_HANDLE;

NotificationCCCDhandle.value.val = NotifyCCCDvalue;

NotificationCCCDhandle.value.len = sizeof(NotifyCCCDvalue);

/\* Report data to BLE component for sending data when read by Central device \*/

CyBle\_GattsWriteAttributeValue(&NotificationCCCDhandle, 0, &connectionHandle, CYBLE\_GATT\_DB\_PEER\_INITIATED);

}

/\* Check if the returned handle is matching to writedata Write Attribute and extract the data\*/

if(CYBLE\_CUSTOM\_SERVICE\_LEDBLINK\_CHAR\_HANDLE == wrReqParam->handleValPair.attrHandle)

{

/\* Extract the Write value sent by the Client \*/

LEDBlinkData[0] = wrReqParam->handleValPair.value.val[0];

LEDBlinkData[1] = wrReqParam->handleValPair.value.val[1];

/\* Update writedata handle with new values \*/

LEDBlinkHandle.attrHandle = CYBLE\_CUSTOM\_SERVICE\_LEDBLINK\_CHAR\_HANDLE;

LEDBlinkHandle.value.val = LEDBlinkData;

LEDBlinkHandle.value.len = sizeof(LEDBlinkData);

/\* Send updated writedata handle as attribute for read by central device \*/

CyBle\_GattsWriteAttributeValue(&LEDBlinkHandle,0,&connectionHandle,0);

/\* Blink the Red LED the number of times equal to the value of data 0\*/

LEDBlink(LEDBlinkData[0]);

}

/\* Send the response to the write request received. \*/

CyBle\_GattsWriteRsp(connectionHandle);

break;

case CYBLE\_EVT\_GATTS\_READ\_CHAR\_VAL\_ACCESS\_REQ:

/\*This event is generated wehn the client requests to read data from the server\*/

rdReqParam = (CYBLE\_GATTS\_CHAR\_VAL\_READ\_REQ\_T \*) eventParam;//copy the details

/\*check to see if the read is to the read data charaacteristic\*/

if(CYBLE\_CUSTOM\_SERVICE\_BUTTONDATA\_CHAR\_HANDLE == rdReqParam->attrHandle)

{

ButtonData[0]=Count;//load the number of switch presses in to the first byte

ButtonData[1]++;//increment the second byte

ButtonDataHandle.attrHandle = CYBLE\_CUSTOM\_SERVICE\_BUTTONDATA\_CHAR\_HANDLE;//this then writes the data in to the database

ButtonDataHandle.value.val = ButtonData;

ButtonDataHandle.value.len = sizeof(ButtonData);

ButtonDataHandle.value.actualLen = sizeof(ButtonData);

/\* Send updated handle as attribute for read by central device \*/

CyBle\_GattsWriteAttributeValue(&ButtonDataHandle, 0, &cyBle\_connHandle, 0);

}

default:

break;

} /\* switch(event) \*/

}

1. We also have to create a custom function to send notification data. This will be sent in the main loop. The function takes the data passed to it, creates a structure containing it and the notify handle, and then sends a notification to the connected central. The function is here:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function Name: SendNotification

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Summary:

\* Send data as BLE Notifications. This function updates

\* the notification handle with data and triggers the BLE component to send

\* notification

\*

\* Parameters:

\* userData: 4bytes of data to be sent in a notification

\*

\* Return:

\* void

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void SendNotification(uint8 userData[2])

{

/\* 'notificationHandle' stores notification data parameters \*/

CYBLE\_GATTS\_HANDLE\_VALUE\_NTF\_T NotifyHandle;

/\* Update notification handle will CapSense slider data\*/

NotifyHandle.attrHandle = CYBLE\_CUSTOM\_SERVICE\_NOTIFYDATA\_CHAR\_HANDLE;

NotifyHandle.value.val = userData;

NotifyHandle.value.len = 2;

/\* Send the updated handle as part of attribute for notifications \*/

CyBle\_GattsNotification(connectionHandle,&NotifyHandle);

}

1. The final aspect of the code is the main loop itself. In here we enable global interrupts, start the BLE component and register the callback, and then enter a for loop. In here we process any ble events, and then read SW2 input to see if is pressed. If it is we wait here until it is released before incrementing the press counter. We also check to see notifications have been enabled, if so we delay a period of time and then create the notification packet with the counter value and an incrementing counter, before sending the notification message. The main loop can be found here:

/\*\*\*\*\*\*\*\*\*\*main function\*\*\*\*\*\*\*\*/

int main(void)

{

CyGlobalIntEnable; /\* Enable global interrupts. \*/

CyBle\_Start(CustomEventHandler);//start the ble component and register the custom callback

for(;;)

{

CyBle\_ProcessEvents(); //check for any ble stack events

if(SW2\_Read()==0) //read the switch and if it is pressed it will equal 0

{

while(SW2\_Read()==0)

{

CyBle\_ProcessEvents();//while the switch is pressed check for stack events

}

Count++;//on release increment the counter

}//end if sw2 read

if(sendNotifications==1)//if notifications have been enabled

{

DelayVal++;//delay for a period of time

if(DelayVal==50000)

{

DelayVal=0;//reset value

NotifyData[0]++;//increment a notifications counter

NotifyData[1] = Count;//load switch press count in to second byte

SendNotification(NotifyData);//call the function to send the notification

}

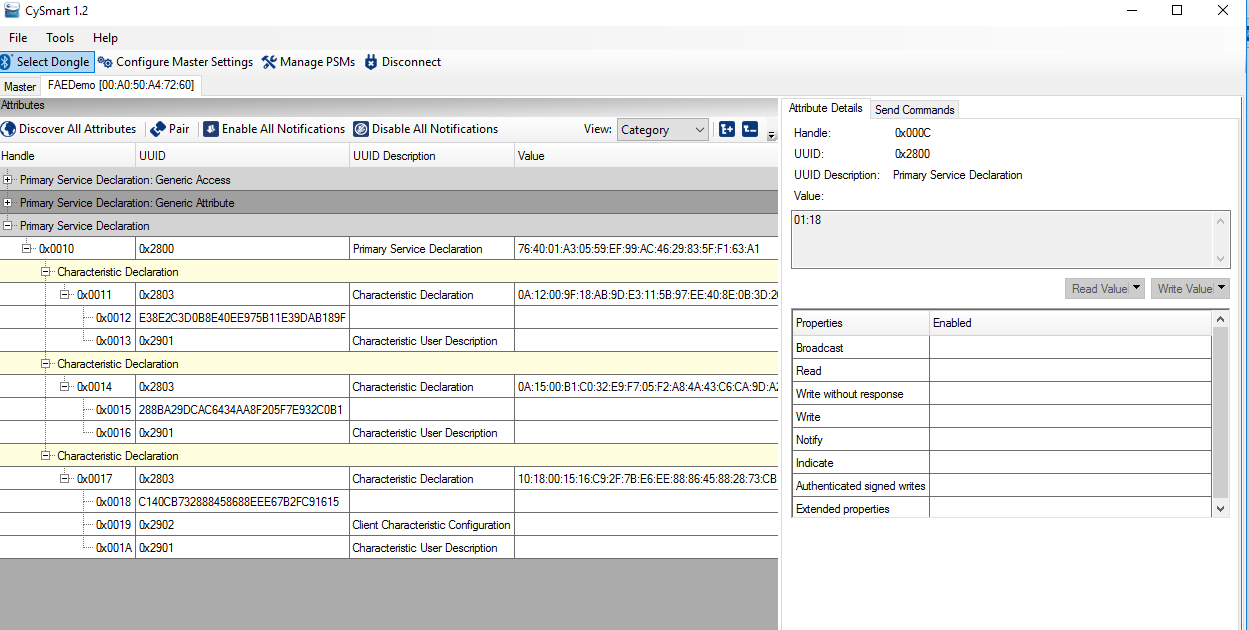
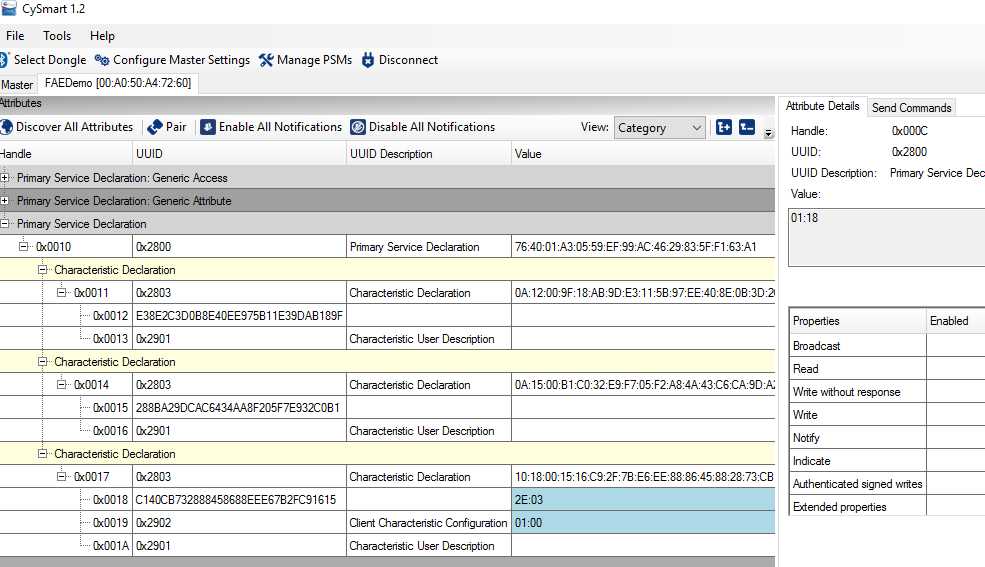
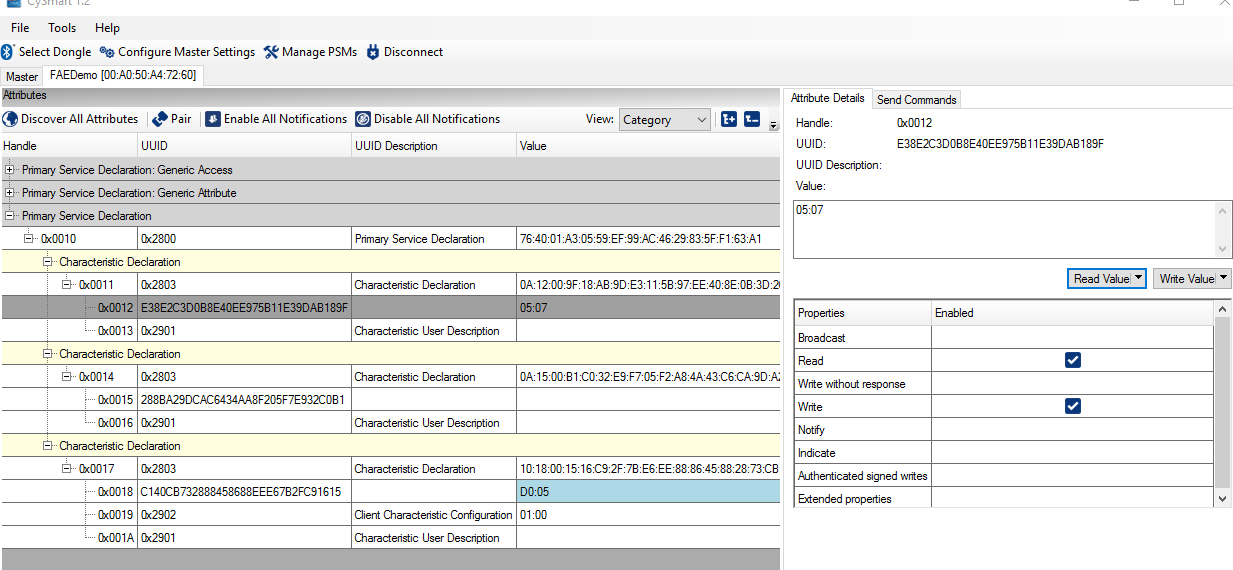
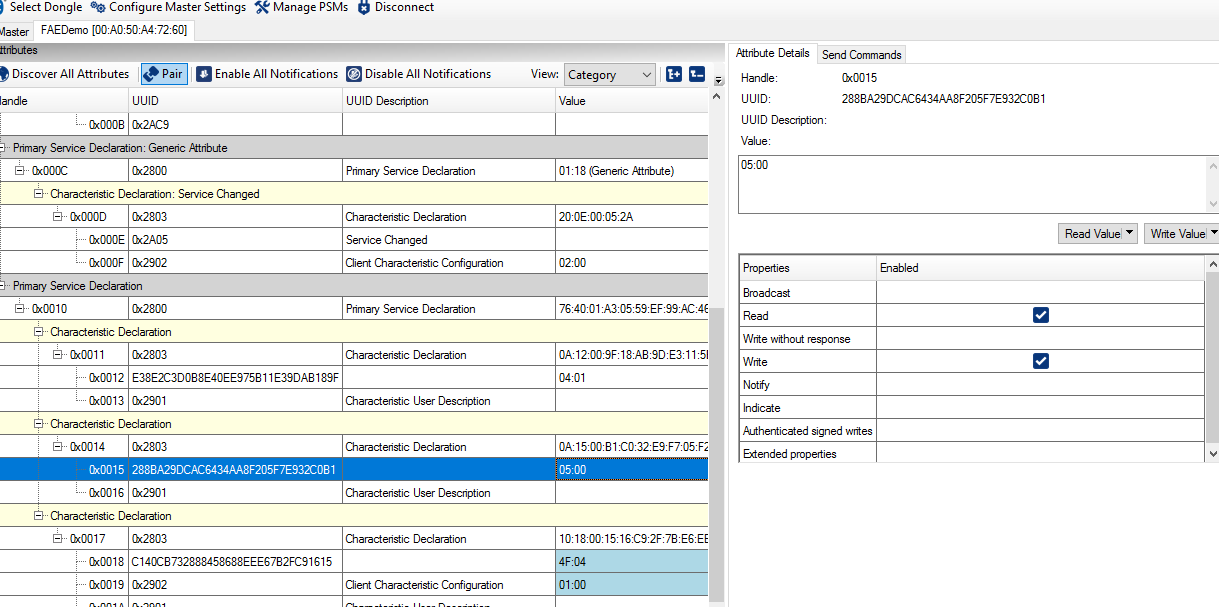
}//end if sendnotifications

}//end of for

}//end of main

1. With this done we can then build the project, ensure there are no errors, before downloading to the device.

**Testing**

1. To test, first launch CySmart 1.2 and plug a CY5677 dongle in. start scanning and we should see FAEDemo being advertised. Click connect, and then discover all attributes.
   1. 
2. If we enable all notifications at the top we should see incrementing data starting to be received. If we press SW2 a few time we should also see the second byte change. In this picture you should be able to see the notification data and that the switch has been pressed 3 times:
   1. 
3. If we read the attribute 0x0012 we should be able to see the number of button presses as well as the number of times the characteristic has been read. In the image below we should see that the switch has been pressed 5 times and we have read the characteristic 7 times:
   1. 
4. The final test is to write to attribute 0x0015 The first byte written to here should cause the RED LED to blink that many times. In this image we have written 5 to characteristic and the LED has blinked 5 times.
   1. 
5. This completes the testing. The application can be modified further to implement other features but shows the 3 main characteristic types as well as basic IO manipulation.