# Bluetooth 21Q2 Features Lab Manual

This lab manual walks you through some of the new Bluetooth SDK features that were introduced between the 20Q4 and 21Q2 release. In this lab we will create an NCP example and write a host software in Python language. Using the new dynamic GATT feature we will also build up the GATT database from the host software instead of using the GATT Configurator. Finally LE Power Control feature is presented by extending the host software.

**KEY FEATURES**

- Python BGAPI
- Dynamic GATT database
- LE Power Control

# 1 Prerequisites

To complete this lab, you will need the followings:

- Two Thunderboard BG22s or two WSTKs with any EFR32BG/EFR32MG radio board or the mix of these
- Simplicity Studio 5 installed, with Gecko SDK v3.2 including Bluetooth SDKv3.2
- A PC on which Python v3.6 or later is installed

# 2 Flashing the Target Devices for NCP Functionality

2.1 Attach your two radio boards and open Simplicity Studio 5

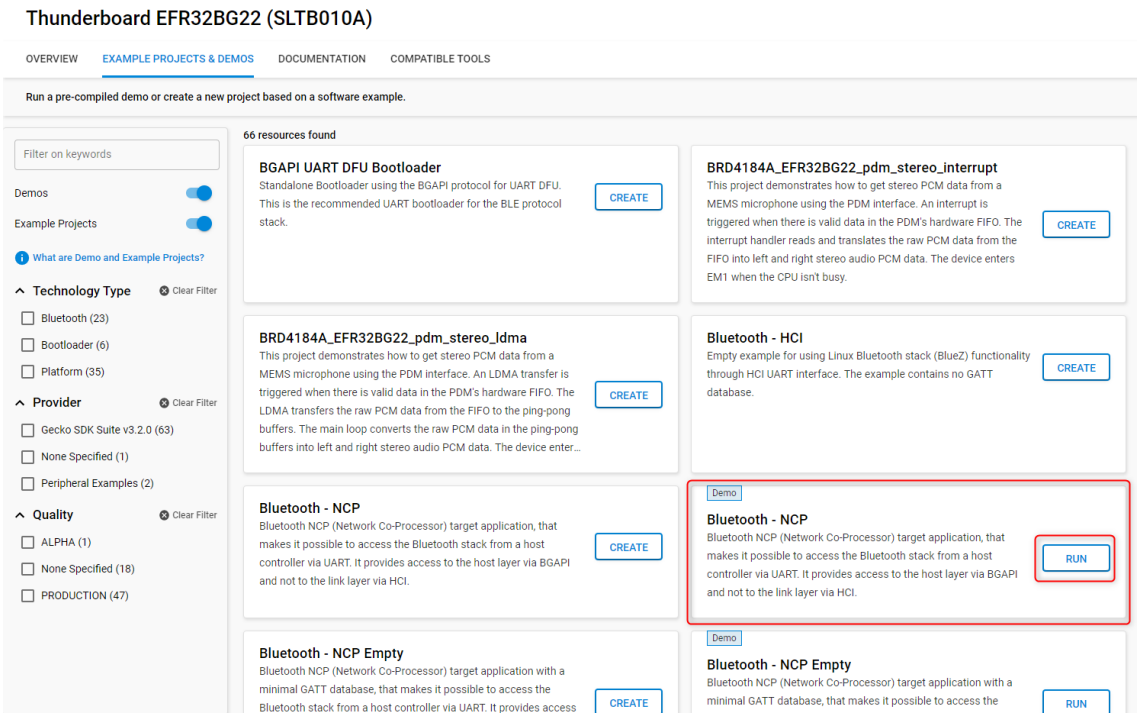
2.2 Select one of the radio boards on the Debug Adapters tab

2.3 Set the Preferred SDK to v3.2.0 on the Overview tab of the Launcher view

2.4 Open the Example Projects & Demos tab

2.5 Find the new **Bluetooth – NCP** demo. (Note: in contrast to Bluetooth – NCP Empty, this project does not include a prebuilt GATT database, but it does have the dynamic GATT API enabled, which is a prerequisite for the next sections)

2.6 Click **Run** to flash the NCP target image to the board.




2.6 Repeat the same steps for the other radio board.

# 3 Creating a Bluetooth Server Application in Python

## Getting Started

3.1 The pybgapi package provides the possibility to issue BGAPI commands toward the target device from the PC using Python programming language. To install this package type the following in the command line:

```
pip install pybgapi
```

For further information about the package visit https://pypi.org/project/pybgapi/

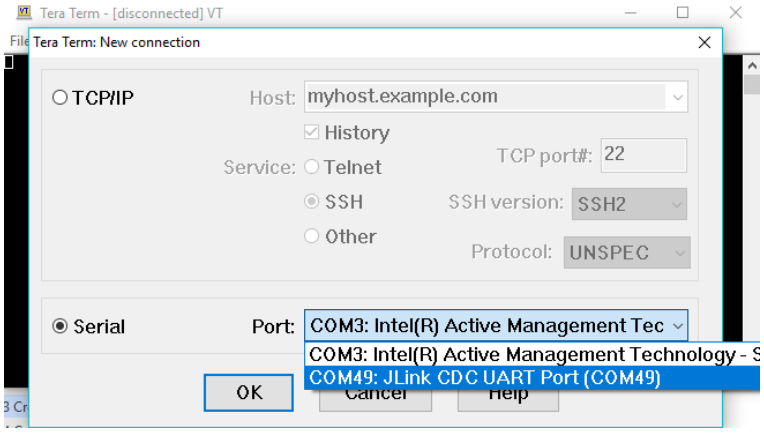
3.2 Locate the latest BGAPI definition file under *C:\SiliconLabs\SimplicityStudio\v5\developer\sdks\gecko_sdk_suite\v3.2.0\protocol\bluetooth\api\sl_bt.xapi* and copy it into your working folder.

3.3 Open the python bash (type `python` in the CLI)

3.4 Import the bgapi library with the following command:

```
>>> import bgapi
```

3.5 Find the COM port number (e.g. COM49) of one of your radio boards. You should look for the "JLink CDC UART Port" in the Device Manager or in your favorite terminal app.




3.6 Connect to your radio board:

```
>>> connection = bgapi.SerialConnector('COM49')
```

3.7 Initialize the pybgapi library for this node:

```
>>> node = bgapi.BGLib(connection,'sl_bt.xapi')
```

3.8 Open BGAPI communication toward this node:

```
>>> node.open()
```

3.9 Check if you can communicate with the board, using the system_hello() command. You should get a system_hello response:

```
>>> node.bt.system.hello()

bt_rsp_system_hello(result=0)
```

3.10 Reset your node with the following command:

```
>>> node.bt.system.reset(0)
```

3.11 Now you should get a system_boot event. To fetch the latest event, use the following command:

```
>>> evt = node.get_events(max_events=1)
>>> print(evt)
[bt_evt_system_boot(major=3, minor=2, patch=0, build=774, bootloader=17563648, hw=1, hash=1181938724)]
```

**Building the GATT database**

3.12 The Bluetooth – NCP target app does not include a prebuilt GATT database. Here we will build up the database from code. First start a session for database building:

```
>>> session = node.bt.gattdb.new_session().session
```

3.13 Add a new service to the GATT database. Here we will add the **Generic Access** service adopted by the Bluetooth SIG. This is a primary service (0x0) with no flags set (0x0) and with a 16bit UUID (0x1800).

```
>>> service = node.bt.gattdb.add_service(session, 0, 0, bytes.fromhex("0018")).service
```

3.14 Add a new characteristic to the service. Here we will add the **Device Name** characteristic to the **Generic Access** service with READ property (0x2), no security requirements (0x0), no flags (0x0), 16bit UUID (0x2a00), variable length (0x2), maximum length of 20 and with initial value of “PyBGAPI Example”:

```
>>> char = node.bt.gattdb.add_uuid16_characteristic(session, service, 2, 0, 0, bytes.fromhex('002a'), 2, 20, bytes('PyBGAPI Example','utf-8')).characteristic
```

3.15 Activate the new service:

```
>>> node.bt.gattdb.start_service(session,service)
bt_rsp_gattdb_start_service(result=0)
```

3.16 Activate the new characteristic:

```
>>> node.bt.gattdb.start_characteristic(session, char)
bt_rsp_gattdb_start_characteristic(result=0)
```

3.17 Save changes and close the database editing session:

```
>>> node.bt.gattdb.commit(session)
bt_rsp_gattdb_commit(result=0)
```

**Connecting to the Server**

3.18 Now that we have a device name in the GATT database, we can start advertising. The stack will automatically advertise the device with the name defined in its GATT database:

```
>>> advertiser_set = node.bt.advertiser.create_set().handle
>>> node.bt.advertiser.start(advertiser_set, 2, 2)
bt_rsp_advertiser_start(result=0)
```

3.19 Start EFR Connect on your phone, and find your device advertising as “PyBGAPI Example”

3.20 You can connect to the device and discover its GATT database which now has the Device Name characteristic

Note: if you want a very quick example without bothering with the GATT database, you can still flash the **Bluetooth – NCP Empty** example to your board, which has a basic prebuilt GATT database. In this case all you have to do on the host side is:

```
>>> import bgapi
>>> connection = bgapi.SerialConnector('COM49')
>>> node = bgapi.BGLib(connection,'sl_bt.xapi')
>>> node.open()
>>> advertiser_set = node.bt.advertiser.create_set().handle
>>> node.bt.advertiser.start(advertiser_set, 2, 2)
bt_rsp_advertiser_start(result=0)
```

## 4 Creating a Bluetooth Client Application in Python

4.1 Creating a client is more complicated than implementing a server. Therefore we will write a python script. Open your favorite text editor and create a new file, let's call it *client.py*

4.2 Import the followings:

```
import sys
import bgapi
```

4.3 Just like in the case of the server, we will connect to the node via UART. Use the COM port number of your second board here:

```
connection = bgapi.SerialConnector('COM51')
node = bgapi.BGLib(connection,'sl_bt.xapi')
node.open()
node.bt.system.reset(0)
```

4.4 From here, our application will be event driven. Whenever a Bluetooth event is generated by the stack, we will handle the event and drive forward the application:

```
while True:
  try:
    evt = node.get_events(max_events=1)
    if evt:
      sl_bt_on_event(evt[0])
  except (KeyboardInterrupt, SystemExit) as e:
    if node.is_open():
      node.close()
    print("Exiting...")
    sys.exit(1)
```

4.5 Let's define the event handler function and add a handler for the system_boot event, where we will start scanning for peripheral devices. Note, that this function should be defined before the while loop (and after the definition of the *node* variable).

```
def sl_bt_on_event(evt):
  global node

  if evt == 'bt_evt_system_boot':
    print("booted, start scanning...")
    node.bt.scanner.start(1,2)
  #elif evt == 'bt_evt_other_event':
    #handle other events here
  else:
    print(evt)
```

4.6 Once the scanner is started, the node will be receiving scan reports. Let's add an event handler for scan reports within the sl_bt_on_event() function. If a scan report is found with the advertised device name "Py-BGAPI Example", the client will open a connection toward that device:

```
  elif evt == 'bt_evt_scanner_scan_report':
    i = 0
    while i < len(evt.data):
      field_len = evt.data[i]
      field_type = evt.data[i+1]
      if (field_type == 0x09):
        print(evt.data[i+2 : i+field_len+1])
        if (evt.data[i+2 : i+field_len+1] == bytes('PyBGAPI Example','utf-8')):
          node.bt.scanner.stop()
          node.bt.connection.open(evt.address, evt.address_type, 1)
      i = i + field_len + 1
```

4.7 Once you reached this point it is worth checking if your client finds the server. Make sure, that you have started the advertisement on the other device, then save *client.py*, and start it from the command line. You should see something like this:

```
PS C:\SiliconLabs\SimplicityStudio\v5\developer\sdks\gecko_sdk_suite\v3.2.0-774\app\bluetooth\my_python> py .\client.py
booted, start scanning...
b'PyBGAPI Example'
bt_evt_connection_opened(address='58:8e:81:a5:50:5d', address_type=0, master=1, connection=1, bonding=255, advertiser=255)
bt_evt_connection_parameters(connection=1, interval=26, latency=0, timeout=100, security_mode=0, txsize=27)
bt_evt_connection_phy_status(connection=1, phy=1)
bt_evt_gatt_mtu_exchanged(connection=1, mtu=247)
bt_evt_connection_parameters(connection=1, interval=26, latency=0, timeout=100, security_mode=0, txsize=251)
```

4.8 The client must discover services and characteristics on the server. Here we will discover the Generic Access service and the Device Name characteristic, and finally read out the value of the Device Name characteristic. Replace your current sl_bt_on_event() function with the following code:

```
state = None
service = None
characteristic = None

def sl_bt_on_event(evt):
  global node
  global state
  global service
  global characteristic

  if evt == 'bt_evt_system_boot':
    print("booted, start scanning...")
    node.bt.scanner.start(1,2)
  elif evt == 'bt_evt_scanner_scan_report':
    i = 0
    while i < len(evt.data):
```

```
      field_len = evt.data[i]
      field_type = evt.data[i+1]
      if (field_type == 0x09):
        print(evt.data[i+2 : i+field_len+1])
        if (evt.data[i+2 : i+field_len+1] == bytes('PyBGAPI Example','utf-8')):
          node.bt.scanner.stop()
          node.bt.connection.open(evt.address, evt.address_type, 1)
      i = i + field_len + 1
  elif evt == 'bt_evt_connection_opened':
    print("connection opened")
    state = 0
  elif evt == 'bt_evt_gatt_mtu_exchanged':
    state = 1
    node.bt.gatt.discover_primary_services(evt.connection)
  elif evt == 'bt_evt_gatt_procedure_completed':
    if (state == 1):
      node.bt.gatt.discover_characteristics(evt.connection, service)
      state = 2
    elif (state == 2):
      node.bt.gatt.read_characteristic_value(evt.connection, characteristic)
      state = 3
    elif (state == 3):
      node.close()
      print("Exiting...")
      sys.exit(1)
  elif evt == 'bt_evt_gatt_service':
    if (evt.uuid == bytes.fromhex('0018')):
      print('service found')
      service = evt.service
  elif evt == 'bt_evt_gatt_characteristic':
    if (evt.uuid == bytes.fromhex('002a')):
      print('characteristic found')
      characteristic = evt.characteristic
  elif evt == 'bt_evt_gatt_characteristic_value':
    print('Device Name: ', evt.value)
  else:
    print(evt)
```

4.9 Save *client.py* and start it from the command line. You should see something like this:

```
PS C:\SiliconLabs\SimplicityStudio\v5\developer\sdks\gecko_sdk_suite\v3.2.0-774\app\bluetooth\my_python> py .\client.py
booted, start scanning...
b'oura_A038F880C90F'
b'PyBGAPI Example'
connection opened
bt_evt_connection_parameters(connection=1, interval=26, latency=0, timeout=100, security_mode=0, txsize=27)
bt_evt_connection_phy_status(connection=1, phy=1)
bt_evt_connection_parameters(connection=1, interval=26, latency=0, timeout=100, security_mode=0, txsize=251)
service found
characteristic found
Device Name:  b'PyBGAPI Example'
Exiting...
Exiting...
PS C:\SiliconLabs\SimplicityStudio\v5\developer\sdks\gecko_sdk_suite\v3.2.0-774\app\bluetooth\my_python>
```

# 5 Adding LE Power Control Feature

## Flashing the Target Devices

LE Power Control is not enabled in the Bluetooth example projects by default. To add this feature, the **Bluetooth > Feature > PowerControl software** component must be installed.

5.1 Open the launcher view of Simplicity Studio 5.

5.2 Select one of your devices in the Debug Adapters tab. Make sure the preferred SDK is v3.2.

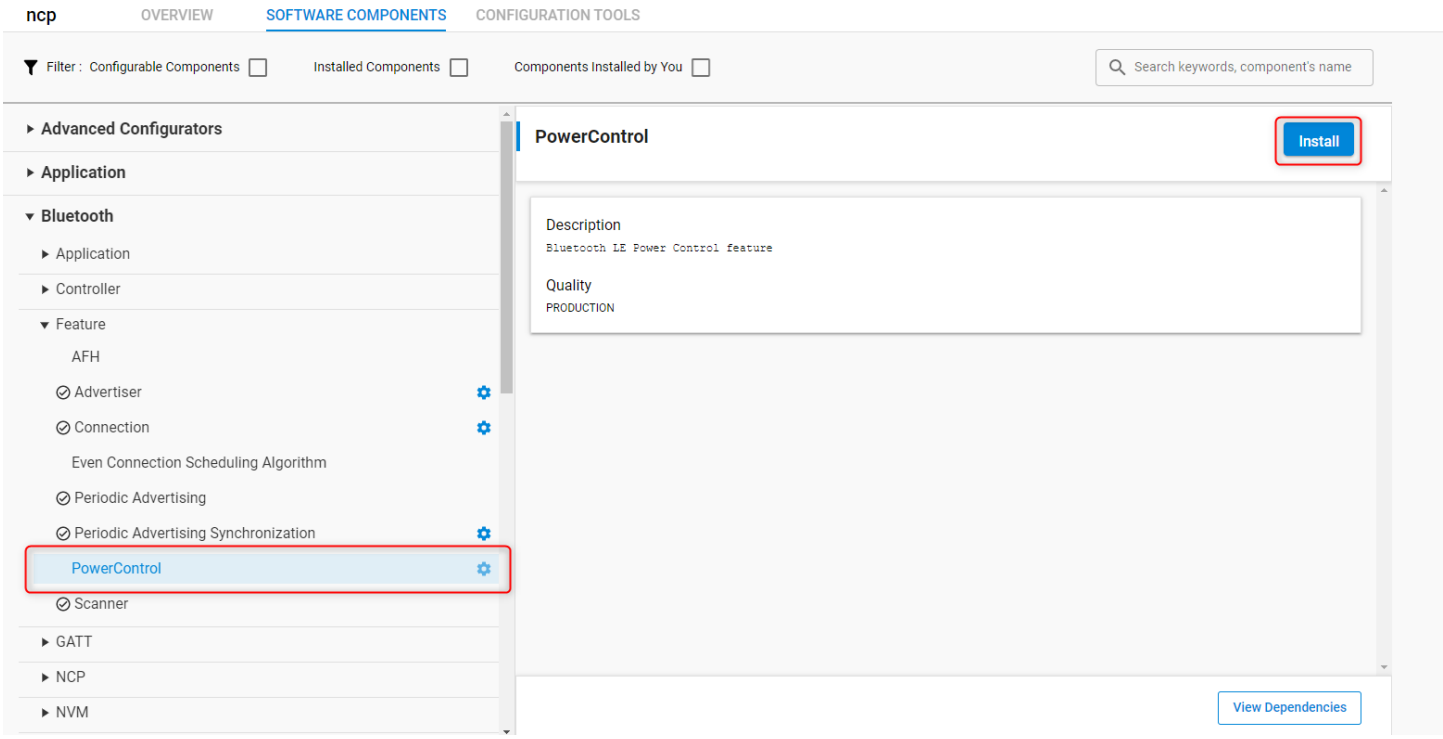
5.3 Open the Example Projects & Demos tab and find the **Bluetooth – NCP Empty** example. Press [Create] to create the project. (This time we do not want to build the GATT database, so we use NCP Empty, which has a default one.)

5.4 Open the GATT Configurator tab, select the Device Name characteristic, and overwrite the “Silabs Example” initial value with “PyBGAPI Example” (so that the client will recognize the server). Also overwrite the value length with 15.

5.5 Press ctrl-s to save the database.

5.6 In the Project Configurator open the Software Components tab.

5.7 Find the Bluetooth > Feature > PowerControl software component, and click [Install]




5.8 Click on the cogwheel next to the PowerControl software component to check the upper and lower limits of the golden range. Set the lower limit for 1M PHY to -45 (instead of -60). Although in practice this value is not optimal, it will result in more Tx power adjustments, which is good for demonstration purposes.

5.8.1 In SDK version 3.2.0, a small workaround needs to be applied to set the golden range properly: open the sl_bluetooth.c file found in the /autogen folder of your project and move the sl_bt_init_power_control(); function call BEFORE sl_bt_init_stack(&config);

```
sl_bt_init_power_control();
sl_status_t err = sl_bt_init_stack(&config);
(void) err;
sl_bt_init_classes(bt_class_table);
sl_bt_init_periodic_advertising();
```

5.9 Build the project and flash it to your board.

5.10 If your two boards are of the same type, flash the same image to the other board as well. If your second board is a different board, then repeat the above steps for the second board.

**Starting the Server and the Client**

5.11 Now again, open the Python bash, connect to your first board, and start advertising

```
>>> import bgapi
>>> connection = bgapi.SerialConnector('COM49')
>>> node = bgapi.BGLib(connection,'sl_bt.xapi')
>>> node.open()
>>> advertiser_set = node.bt.advertiser.create_set().handle
>>> node.bt.advertiser.start(advertiser_set, 2, 2)
bt_rsp_advertiser_start(result=0)
```

5.12 Modify your client application so, that it does not exit after reading out the device name. Find the following lines, and put them into a comment:

```
#elif (state == 3):
  #node.close()
  #print("Exiting...")
  #sys.exit(1)
```

5.13 Save and run your client application

```
py .\client.py
```

5.14 Place your two boards far away, then slowly move them closer to each other. Now you should see that the stack starts decreasing its power level from the default 8dBm down to -3dBm (which is the minimal Tx power by default):

```
PS C:\Users\arkalvac\Documents\lab> py client.py
booted, start scanning...
b'PyBGAPI Example'
connection opened
bt_evt_connection_parameters(connection=1, interval=17, latency=0, timeout=100, security_mode=0, txsize=27)
bt_evt_connection_phy_status(connection=1, phy=1)
bt_evt_connection_remote_used_features(connection=1, features=b'-\xc9\x00\x00\x06\x00\x00\x00')
service found
bt_evt_connection_parameters(connection=1, interval=17, latency=0, timeout=100, security_mode=0, txsize=251)
characteristic found
device name:  b'PyBGAPI Example'
bt_evt_connection_tx_power(connection=1, phy=1, power_level=5, flags=0, delta=-3)
bt_evt_connection_tx_power(connection=1, phy=1, power_level=2, flags=0, delta=-3)
bt_evt_connection_tx_power(connection=1, phy=1, power_level=6, flags=0, delta=4)
bt_evt_connection_tx_power(connection=1, phy=1, power_level=-1, flags=0, delta=-7)
bt_evt_connection_tx_power(connection=1, phy=1, power_level=3, flags=0, delta=4)
bt_evt_connection_tx_power(connection=1, phy=1, power_level=-1, flags=0, delta=-4)
bt_evt_connection_tx_power(connection=1, phy=1, power_level=-3, flags=1, delta=-2)
bt_evt_connection_tx_power(connection=1, phy=1, power_level=2, flags=0, delta=5)
bt_evt_connection_tx_power(connection=1, phy=1, power_level=-3, flags=1, delta=-5)
bt_evt_connection_tx_power(connection=1, phy=1, power_level=3, flags=0, delta=6)
bt_evt_connection_tx_power(connection=1, phy=1, power_level=-3, flags=1, delta=-6)
```