BLUETOOTH® SMART CABLE REPLACEMENT

APPLICATION NOTE

Monday, 15 October 2012

Version 1.5

VERSION HISTORY

Version	Comment
1.0	First version
1.1	BGScript instructions added
1.3	Compilation and installation instructions added
1.4	iPhone example added
1.5	BGScript section updated

TABLE OF CONTENTS

1 Introduction

This application note discusses how to build a *Bluetooth* 4.0 Cable Replacement Profile (CRP) sensor using Bluegiga's *Bluetooth* 4.0 software development kid, for use with a DKBLE112 hardware evaluation board and an Apple iPhone. The application note contains a practical example on how to build GATT based Cable Replacement Profile and how to make a standalone sensor (DevA) device using BGScript scripting language.

Notice that Cable Replacement profile is a Bluegiga proprietary profile and is not a profile standardized by the *Bluetooth* SIG.

2 What is *Bluetooth* Smart technology?

Bluetooth low energy (*Bluetooth* 4.0) is a new, open standard developed by the *Bluetooth* SIG. It's targeted to address the needs of new modern wireless applications such as ultra-low power consumption, fast connection times, reliability and security. *Bluetooth* low energy consumes 10-20 times less power and is able to transmit data 50 times quicker than classical *Bluetooth* solutions.

Bluetooth low energy is designed for new emerging applications and markets, but it still embraces the very same benefits we already know from the classical, well established *Bluetooth* technology:

- **Robustness and reliability** - The adaptive frequency hopping technology used by *Bluetooth* low energy allows the device to quickly hop within a wide frequency band, not just to reduce interference but also to identify crowded frequencies and avoid them. On addition to broadcasting *Bluetooth* low energy also provides a reliable, connection oriented way of transmitting data.
- **Security** - Data privacy and integrity is always a concern is wireless, mission critical applications. Therefore *Bluetooth* low energy technology is designed to incorporate high level of security including authentication, authorization, encryption and man-in-the-middle protection.
- **Interoperability** - *Bluetooth* low energy technology is an open standard maintained and developed by the *Bluetooth* SIG. Strong qualification and interoperability testing processes are included in the development of technology so that wireless device manufacturers can enjoy the benefit of many solution providers and consumers can feel confident that equipment will communicate with other devices regardless of manufacturer.
- **Global availability** - Based on the open, license free 2.4GHz frequency band, *Bluetooth* low energy technology can be used in world wide applications.

There are two types of *Bluetooth* 4.0 devices:

- ***Bluetooth* 4.0 single-mode** devices that only support *Bluetooth* low energy and are optimized for low-power, low-cost and small size solutions.
- ***Bluetooth* 4.0 dual-mode** devices that support *Bluetooth* low energy and classical *Bluetooth* technologies and are interoperable with all the previously *Bluetooth* specification versions.

Key features of *Bluetooth* low energy wireless technology include:

- Ultra-low peak, average and idle mode power consumption
- Ability to run for years on standard, coin-cell batteries
- Low cost
- Multi-vendor interoperability
- Enhanced range

Bluetooth low energy is also meant for markets and applications, such as:

- Automotive
- Consumer electronics
- Smart energy
- Entertainment
- Home automation
- Security & proximity
- Sports & fitness

3 Cable Replacement profile

3.1 Description

Cable Replacement Profile enables a Cable Replacement DevB to connect and exchange data with a Cable Replacement DevA. The Profile provides a reliable and transparent way to send data from the sensor device's UART interface over *Bluetooth* low energy connection to a collector device.

Cable Replacement Profile defines two roles:

1. **The Cable Replacement DevA**

The Cable Replacement DevA receives data from its UART interface and transparently exposes it via a Cable Replacement Service. The DevA can also receive data from the Cable Replacement DevB and transparently route the data to the UART interface. The Cable Replacement DevA is the GATT server.

2. **The Cable Replacement DevB**

The Cable Replacement DevB accesses the information exposed by the Cable Replacement DevA and can for example display it to the end user or store it on non-volatile memory for later analysis. The Cable Replacement DevB is the GATT client and typically a device like a mobile phone or a PC.

The figure below shows the relationship of these two roles.

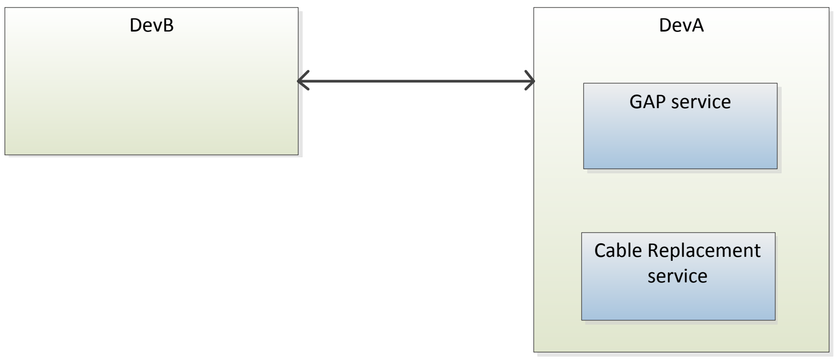


Figure 1: Cable Replacement Profile roles

3.2 GATT Server: Service requirements

The table below describes the service requirements.

Service	UUID	Cable Replacement DevA
GAP service	1800	Mandatory
Cable Replacement Service	0bd51666-e7cb-469b-8e4d-2742f1ba77cc	Mandatory

Table 1: Service requirements

3.3 GATT Server: Attribute requirements

The table below describes the structure and requirements for the attribute used in Cable Replacement Service

Characteristic	UUID	Length	Type	Support	Security	Properties
Cable Replacement Data	e7add780-b042-4876-aae1-112855353cc1	Variable (max 20B)	user data	Mandatory	Optional	Indicate Write

Table 2: Cable Replacement Service structure

- **UUID** is a manufacturer proprietary 128-bit ID for the characteristic
- The characteristic **length** is variable and has a maximum length of 20 bytes as that is the maximum payload for indicated values
- By default the security is not required, but one can optionally implement it if the application requires security
- By default the characteristic has **indicate** and **write** properties value.

3.4 Other requirements

The Cable Replacement DevA includes the Cable Replacement Service UUID in the advertisement data. This allows the DevB to filter and recognize only the DevA during the scanning process. This is especially valid when DevB is an iPhone4S/5 or and iPad 3.

The Cable Replacement DevA should include the device name in the advertisement or scan response data.

The Cable Replacement DevA may support write property for the device's local name characteristic, allowing the Collector to edit the friendly name.

3.5 Recommended connection establishment procedures

3.5.1 Un-bonded devices

Advertisement duration	Parameter	Value
First 30 seconds	Advertising interval	20ms to 30ms
After 30 seconds	Advertising interval	1000ms to 2500ms

Table 3: Advertising parameters for un-bonded Cable Replacement DevA

- The Cable Replacement DevA shall accept any valid values for connection interval and slave latency set by the DevB until service discovery, bonding and/or encryption are complete. After this the DevA may request the change of connection parameters.
- If the connection is not established within a time limit, the sensor may exit GAP Connectable mode.
- If bonded the Cable Replacement DevA should write the address of the Collector to the white list and should set the filtering policy so that scan and connection requests are only accepted from devices on the white list.
- When Cable Replacement DevA is disconnected by the Collector and ready to receive a connection (i.e. senses the Cable Replacement) it should initiate the connection procedure.

3.5.2 Bonded devices

The following produce is uses for bonded devices:

- The Cable Replacement DevA should use GAP General discoverable mode with connectable undirected advertisement events.
- For the first 10 seconds the white list should be used to allow only connections from bonded devices. After 10 seconds the white list should not be used to allow connections from other devices.
- The advertisement parameters should be as in Table 3.
- The Cable Replacement DevA shall accept any valid values for connection interval and slave latency set by the DevB until service discovery, bonding and/or encryption have are complete. After this the sensor may request the change of connection parameters.
- If the connection is not established within a time limit, the sensor may exit GAP Connectable mode.
- When Cable Replacement DevA is disconnected by the Collector and ready to receive a connection (i.e. senses the Cable Replacement) it should initiate the connection procedure.

3.5.3 Link loss procedure

When connection is terminated due to link loss the sensor should attempt reconnection with the Collector by entering the GAP connectable mode using the recommended parameters from Table 3.

3.6 Security requirements

The Cable Replacement DevA may bond with the Collector.

When bonding is not used:

1. The Cable Replacement DevA should use the *Slave Security Request* procedure to inform the Collector of its security requirements.

When bonding is used:

1. The Cable Replacement DevA shall use LE security Mode 1 and either Security Level 2 or 3.
2. The Cable Replacement DevA shall use the *Slave Security Request* procedure.
3. All supported characteristics specified by the Cable Replacement Service shall be set to Security Mode 1 and either Security Level 2 or 3.
4. All supported characteristics specified by the Device Information Service should be set to the same security mode and level as the characteristics in the Cable Replacement Service.

4 Implementing a Cable Replacement DevA

The chapter contains step by step instructions how to implement a stand-alone Cable Replacement DevA with Bluegiga's *Bluetooth* 4.0 Software Development Kit. The chapter is split into following steps:

1. Creating a project
2. Defining hardware configuration
3. Building Cable Replacement and Device Information Services with Profile Toolkit
4. Writing a BGScript code
5. Compiling the GATT database and BGScript into a binary firmware
6. Installing the firmware into BLE112 or DKBLE112 hardware

The actual project comes as an example with the Bluegiga's *Bluetooth* low energy Software Development Kit v.1.1 or newer under the \example\cable_replacement\ directory.

4.1 Creating a project

The Cable Replacement DevA implementation is started by first creating a project file (**project.xml**), which defines the resources use by the project and the firmware output file.

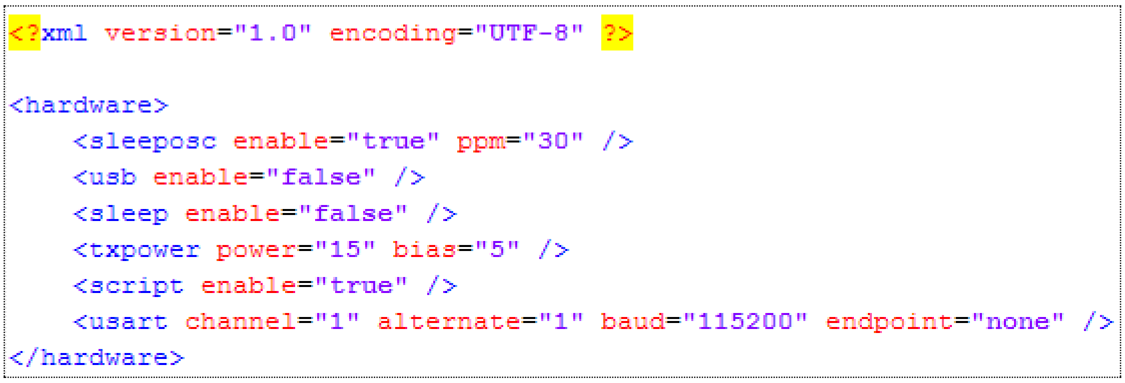
```xml
<?xml version="1.0" encoding="UTF-8" ?>
<project>
    <gatt in="gatt.xml" />
    <hardware in="evkit_hardware.xml" />
    <script in="evkit_conn_slave_gatt_server.bgs" />
    <config in="config.xml" />
    <image out="out.hex" />
</project>
```

Figure 2: Project file

• **<gatt>**	Defines the XML-file containing the GATT database.
• **<hardware>**	Defines the XML-file containing the hardware configuration.
• **<script>**	Defines the BGScript-file which contains the BGScript code.
• **<config>**	Defines the application configuration file.
• **<image>**	Defines the output HEX file containing the firmware image.

4.2 Hardware configuration

The **hardware.xml** file contains the hardware configuration for BLE112 device. It describes which interfaces and functions are in used and their properties.



```xml
<?xml version="1.0" encoding="UTF-8" ?>

<hardware>
    <sleeposc enable="true" ppm="30" />
    <usb enable="false" />
    <sleep enable="false" />
    <txpower power="15" bias="5" />
    <script enable="true" />
    <usart channel="1" alternate="1" baud="115200" endpoint="none" />
</hardware>
```

Figure 3: Hardware configuration for Cable Replacement DevA

• **<sleeposc>**	The 32.768KHz sleep oscillator is enabled. Sleep oscillator allows the device to enter power mode 1 or 2 between *Bluetooth* operations, for example between connection intervals.
• **<usb>**	USB interface is disabled to save power and allow the device to go to low power modes.
• **<sleep>**	Disables the power saving modes.
• **<txpower>**	TX power is set to +3dBm value. Every step represents roughly a 1dBm change and the range of the parameter is 15 to 0, corresponding TX power values from +3dBm to -24dBm.
• **<script>**	Scripting is enabled as the Cable Replacement application is implemented with BGScript scripting language.
• **<usart>**	Enables the USART interface used for data communications. In this configuration USART 1 is used in alternate configuration 1 and with 115200 bps baud rate. RTS/CTS flow control is enabled.

The example is designed to work with the DKBLE112 development kit in the default configuration, so it can be easily trialled with the DKBLE112 and an iPhone running for example Bluegiga's demo application.

4.3 GATT database for Cable Replacement

This section describes how to define the Cable Replacement Profile's services using Bluegiga's Profile Toolkit.

The Cable Replacement Profile contains two services:

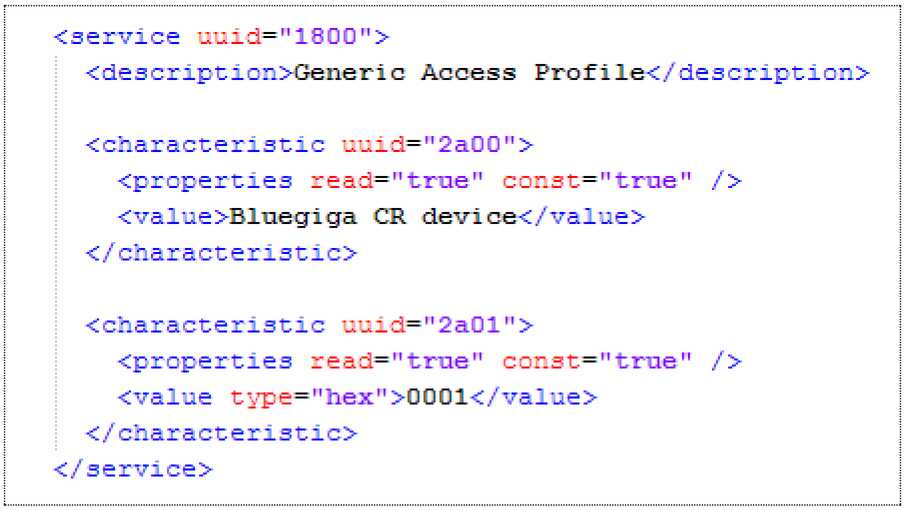
1. Generic Access Profile (GAP) service
2. Cable Replacement Service (GRS)

Optionally if the application requires it other services can be implanted, like Device Information or Battery Status services.

4.3.1 Generic Access Profile (GAP) service

Every *Bluetooth* low energy device needs to implement a GAP service. The GAP service is very simple and consists of only two characteristics. An example implementation of GAP service is show below.

The service has two characteristics, which are explained in Table 4. In this example the characteristics are read-only, so they are also marked as **const.** Constant values are stored on the flash of BLE112 and the value is defined in the GATT database. Constant values cannot be changed.



```xml
<service uuid="1800">
  <description>Generic Access Profile</description>

  <characteristic uuid="2a00">
    <properties read="true" const="true" />
    <value>Bluegiga CR device</value>
  </characteristic>

  <characteristic uuid="2a01">
    <properties read="true" const="true" />
    <value type="hex">0001</value>
  </characteristic>
</service>
```

Figure 4: GAP service

Characteristic	UUID	Type	Support	Security	Properties
Device name	2a00	UTF8	Mandatory	none	Read (optionally write)
Appearance	2A01	16bit	Mandatory	none	Read

Table 4: GAP service description

If the device name needs to be changeable by the remote device, then the write property should be enabled.

4.3.2 Cable Replacement Service

DevA is the side implementing the GATT Server, so it must also implement the Cable Replacement Service. Cable Replacement Service is defined as below:

Characteristic	UUID	Length	Type	Support	Security	Properties
Cable Replacement Data	e7add780-b042-4876-aae1-112855353cc1	Variable (max 20B)	user data	Mandatory	Optional	Indicate Write

Table 5: Cable Replacement Service description

The Cable Replacement Service is created by adding the code below to the gatt.xml:

```
<service uuid="0bd51666-e7cb-469b-8e4d-2742f1ba77cc" advertise="true">
    <description>Cable replacement service</description>

    <characteristic uuid="e7add780-b042-4876-aae1-112855353cc1" id="xgatt_data">
        <description>Data</description>
        <properties write="true" indicate="true" />
        <value variable_length="true" length="20" type="user" />
    </characteristic>
</service>
```

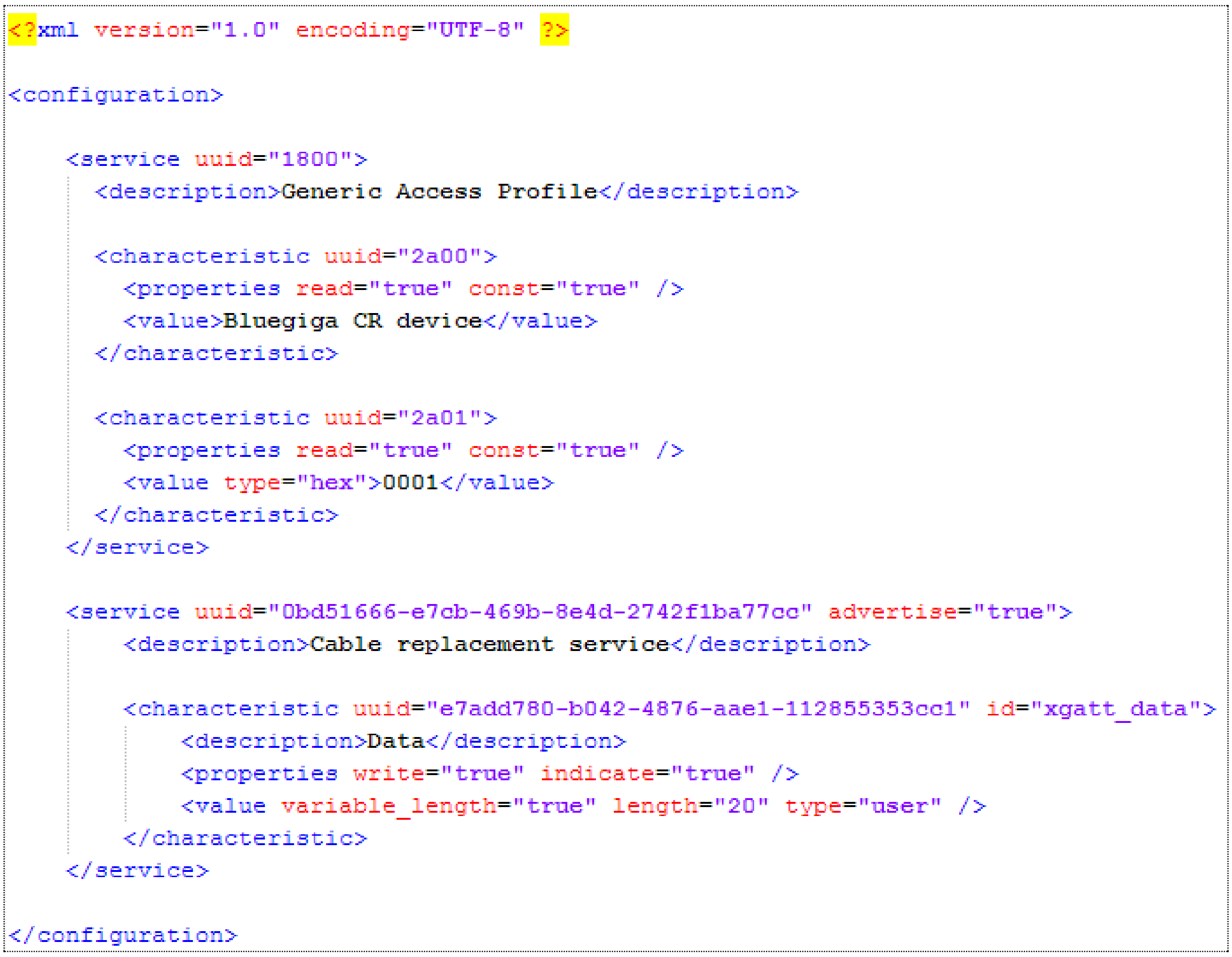
Figure 5: Minimal Cable Replacement Service

The cable replacement service is explained below:

- First of all the **advertise="true"** option is needed for the Cable Replacement Service. When the DevA advertises, the UUID of the CR service is included in the data field of the advertisement packets, so the device can be easily identified by the demote devices.
- The **id="xgatt_data"** defines the attribute ID, which the BGScript application can use to update the Cable Replacement data.
- **Indicate** and **write** properties are enabled for the CR service. Indicate allows the Cable Replacement DevA to indicate the data automatically to the DevB when it changes, so there is no need for the DevB to actively poll the data, and significant power savings can be achieved. Indicate also provides acknowledgements of data, so the data gets reliably transmitted to the remote end. Write property on the other hand allows the Cable Replacement DevB reliably to write data to the DevA.
- As indication and write only allow 20 bytes to be transmitted at a time, the maximum length of the attribute is selected to be 20 bytes, but it is also marked as variable length to allow transmission of smaller amounts of data more efficiently.

4.3.3 Summary

The full GATT database implementation is shown below.



```xml
<?xml version="1.0" encoding="UTF-8" ?>

<configuration>

    <service uuid="1800">
      <description>Generic Access Profile</description>

      <characteristic uuid="2a00">
        <properties read="true" const="true" />
        <value>Bluegiga CR device</value>
      </characteristic>

      <characteristic uuid="2a01">
        <properties read="true" const="true" />
        <value type="hex">0001</value>
      </characteristic>
    </service>

    <service uuid="0bd51666-e7cb-469b-8e4d-2742f1ba77cc" advertise="true">
        <description>Cable replacement service</description>

        <characteristic uuid="e7add780-b042-4876-aae1-112855353cc1" id="xgatt_data">
            <description>Data</description>
            <properties write="true" indicate="true" />
            <value variable_length="true" length="20" type="user" />
        </characteristic>
    </service>

</configuration>
```

Figure 6: Cable Replacement Profile GATT database

4.4 BGScript for Cable Replacement

The example implements a standalone Cable Replacement DevA device where no external host processor is needed. The Cable Replacement DevA application is created as a BGScript script application and the BGScript code is explained in this chapter.

BGScript uses an event based programming approach. The script is executed when an event takes place, and the programmer may register listeners for various events.

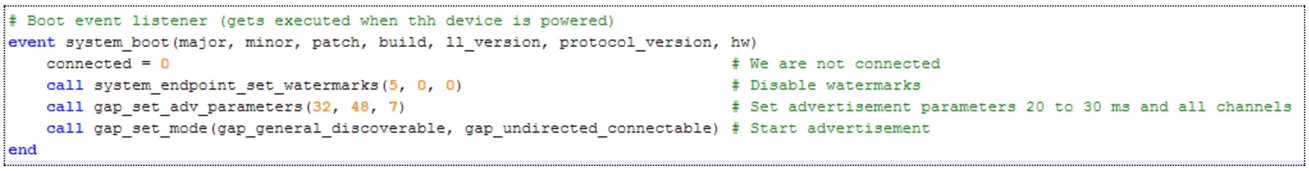
The Cable Replacement DevA BGScript uses the following event listeners:

4.4.1 System: Boot event

When the system is started or reset a boot event is generated and this event listener should be the entry point for all the BGScript applications.

The boot event code for the cable replacement application is described below.

In the example above, when the system is started, the unit starts to advertise, enables bonding mode, and starts a timer.



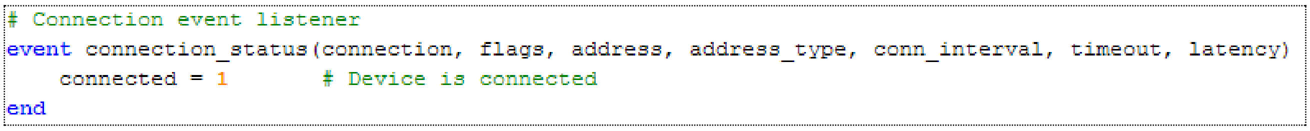
```
# Boot event listener (gets executed when thh device is powered)
event system_boot(major, minor, patch, build, ll_version, protocol_version, hw)
    connected = 0                                                           # We are not connected
    call system_endpoint_set_watermarks(5, 0, 0)                            # Disable watermarks
    call gap_set_adv_parameters(32, 48, 7)                                  # Set advertisement parameters 20 to 30 ms and all channels
    call gap_set_mode(gap_general_discoverable, gap_undirected_connectable) # Start advertisement
end
```

Figure 7: system_boot(...) event listener

The system boot code simply initializes the device state and starts advertisements, so the DevA can be connected.

4.4.2 *Bluetooth*: Connection event

When the *Bluetooth* connection is established a connection event occurs. For this purpose an event listener is added to the BGScript and it simply changes the connections status flag.

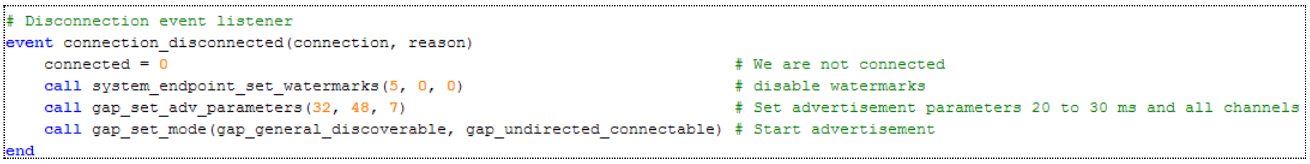


```
# Connection event listener
event connection_status(connection, flags, address, address_type, conn_interval, timeout, latency)
    connected = 1        # Device is connected
end
```

Figure 8: Connection event listener

4.4.3 *Bluetooth*: Disconnection event

If the *Bluetooth* connection is a lost a disconnection event occurs. For this purpose an event listener is added to the BGScript and it does almost the same as the boot event listener, so it sets the application state and restarts advertisement procedure.



```
# Disconnection event listener
event connection_disconnected(connection, reason)
    connected = 0                                                      # We are not connected
    call system_endpoint_set_watermarks(5, 0, 0)                       # disable watermarks
    call gap_set_adv_parameters(32, 48, 7)                             # Set advertisement parameters 20 to 30 ms and all channels
    call gap_set_mode(gap_general_discoverable, gap_undirected_connectable) # Start advertisement
end
```

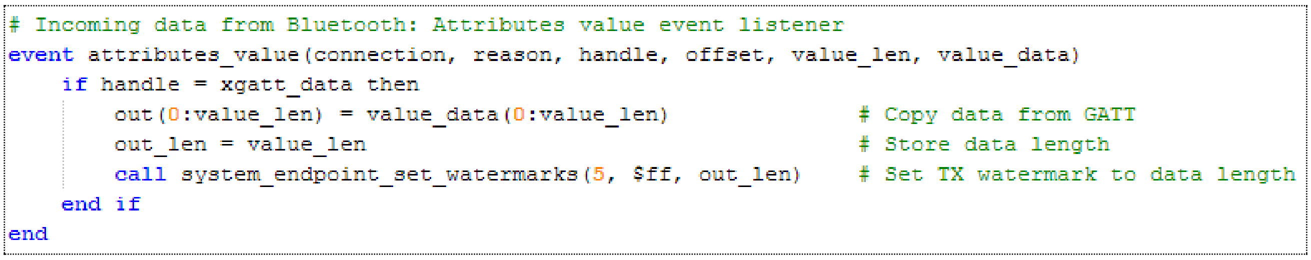
Figure 9: Disconnection event listener

4.4.4 Data: Receiving data from a *Bluetooth* connection

The ATTribute protocol is used to transmit data over a *Bluetooth* connection. A remote device can use ATT write operation to write up to 20 bytes of data into the Cable Replacement DevA over a *Bluetooth* connection.

When the ATT write operation is made an event, which can be captured with BGScript, is generated. This event listener catches the data written to **xgatt_data** attribute and prepares the UART write operation.

The BGScript code is for receiving data from a *Bluetooth* connection is shown below.



```
# Incoming data from Bluetooth: Attributes value event listener
event attributes_value(connection, reason, handle, offset, value_len, value_data)
    if handle = xgatt_data then
        out(0:value_len) = value_data(0:value_len)             # Copy data from GATT
        out_len = value_len                                    # Store data length
        call system_endpoint_set_watermarks(5, $ff, out_len)   # Set TX watermark to data length
    end if
end
```

Figure 10: Attribute value event listener

As seen above the code is fairly simple. It only checks the proper attribute handle is written and copies the received data and data length to a local attribute.

The code also configures the UART0 endpoint watermark to the value of the received data length. This will generate another event, which again can be captured with BGScript and the data can be written to the UART interface.

4.4.5 Data: Writing data to UART

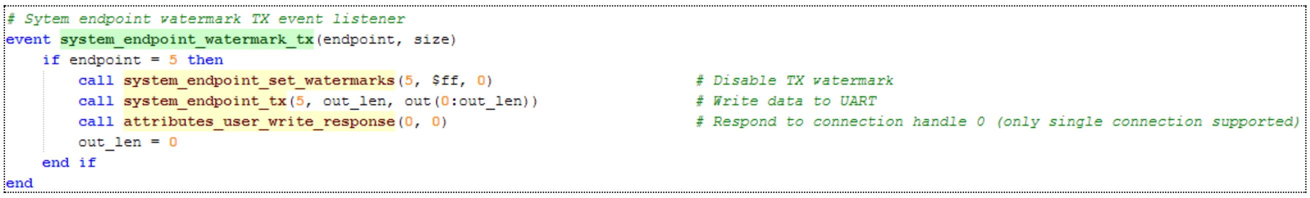
The Bluegiga *Bluetooth* Smart stack uses a concept called endpoint for receiving or transmitting data to interfaces like UART or USB. When data is received from an endpoint an event is generated and an event listener can be written to read and handle this data. On the other hand when data needs to be sent to an endpoint an endpoint TX function can be called.

The endpoint flow control is managed with a concept called watermarks. Watermarks allow users to configure how much data needs to be received from an endpoint for an event to be generated. These events can also be disabled when the data is being processed in order to prevent data overflow from happening.

When this event occurs can be configured by defining the number of bytes, which need to be received from an endpoint by setting a watermark for the desired number of bytes.

The BGScript code below is executed when an endpoint watermark TX event caught i.e. when there is data in the buffer that needs to be written in the UART or in other words when remote *Bluetooth* device has performed an attribute write.

The BGScript code first disables the TX watermark, so no more data can will be received, then writes the data to the UART interface and finally acknowledges the data reception to the remote *Bluetooth* device.



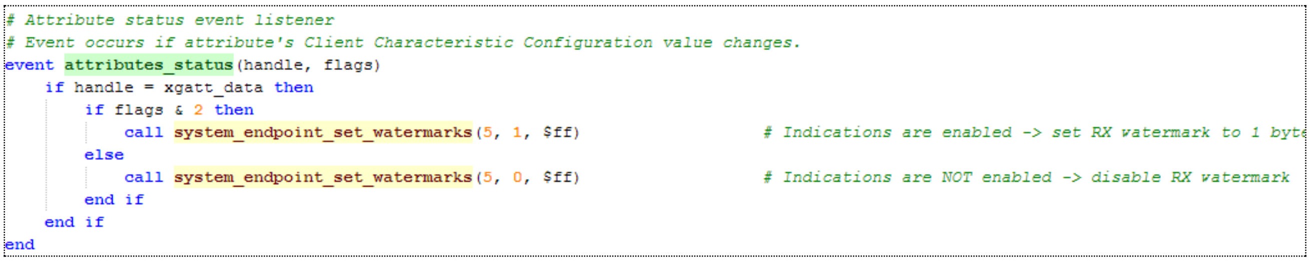
```
# Sytem endpoint watermark TX event listener
event system_endpoint_watermark_tx(endpoint, size)
    if endpoint = 5 then
        call system_endpoint_set_watermarks(5, $ff, 0)                 # Disable TX watermark
        call system_endpoint_tx(5, out_len, out(0:out_len))            # Write data to UART
        call attributes_user_write_response(0, 0)                      # Respond to connection handle 0 (only single connection supported)
        out_len = 0
    end if
end
```

Figure 11: Writing data to UART

4.4.6 Data: Enabling data reception from UART

In order to send data from the local UART to a remote *Bluetooth* device additional BGScript code needs to be written. The Cable Replacement code uses indications to reliably send the data received from UART to the remote *Bluetooth* device.

The BGScript below is called when the remote *Bluetooth* device enables or disables the indications for the **Cable Replacement Data** attribute and it enables or disables the RX watermark event which indicates to the BGScript that data is available from the UART. The logic of this functionality is that if there is no device listening for the data received from UART, the data can be lost.



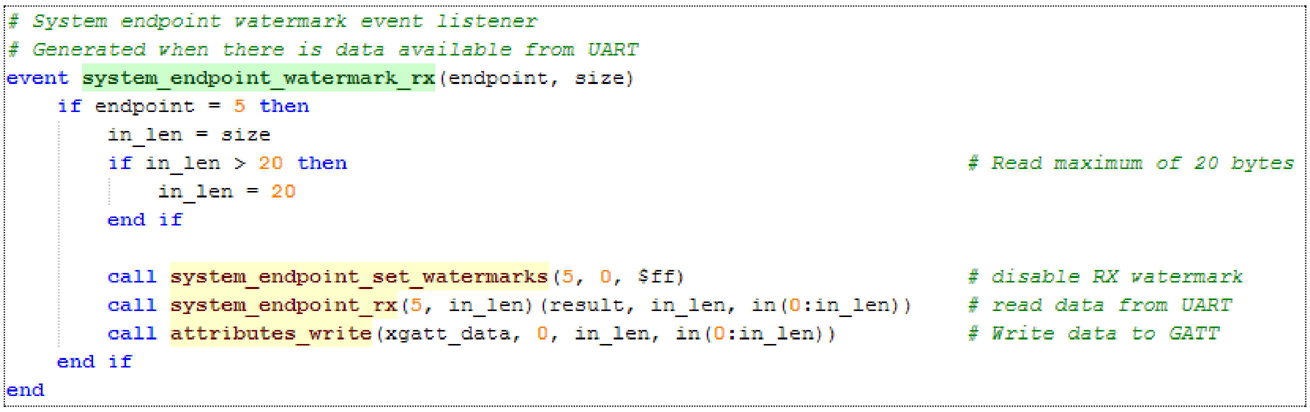
```
# Attribute status event listener
# Event occurs if attribute's Client Characteristic Configuration value changes.
event attributes_status(handle, flags)
    if handle = xgatt_data then
        if flags & 2 then
            call system_endpoint_set_watermarks(5, 1, $ff)                  # Indications are enabled -> set RX watermark to 1 byt
        else
            call system_endpoint_set_watermarks(5, 0, $ff)                  # Indications are NOT enabled -> disable RX watermark
        end if
    end if
end
```

Figure 12: Catching indication enable/disable event

4.5 Data: Receiving data to UART and forwarding it to *Bluetooth* connection

When data is indicated from a remote *Bluetooth* device an event is generated and it can be caught with BGScript. When such an event is caught the endpoint RX watermark is enabled and shown in the script example below.

This section of the BGScript first disables the RX watermark, so no more data can be received and then it reads the maximum amount (20 bytes) of data from the UART and writes it to the GATT database. The attribute then gets indicated to the remote *Bluetooth* device.



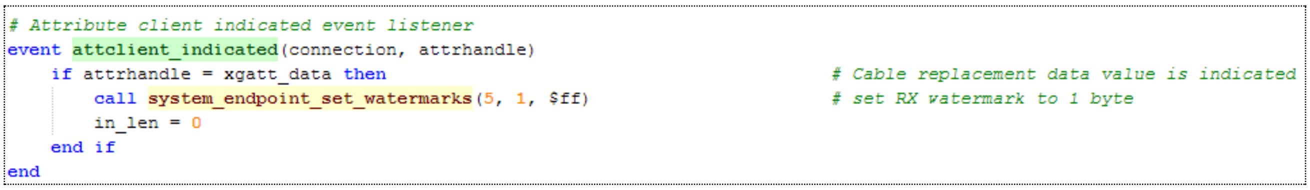
```
# System endpoint watermark event listener
# Generated when there is data available from UART
event system_endpoint_watermark_rx(endpoint, size)
    if endpoint = 5 then
        in_len = size
        if in_len > 20 then                                                     # Read maximum of 20 bytes
            in_len = 20
        end if

        call system_endpoint_set_watermarks(5, 0, $ff)                          # disable RX watermark
        call system_endpoint_rx(5, in_len)(result, in_len, in(0:in_len))        # read data from UART
        call attributes_write(xgatt_data, 0, in_len, in(0:in_len))              # Write data to GATT
    end if
end
```

Figure 13: Data reception from UART

The final section of the script is executed when the UART data is indicated to the *Bluetooth* connection and the remote device confirms the data reception. When the indication is acknowledged by the remote device an event is generated, which can be caught with BGScript.

The script below catches the acknowledgement and enables the endpoint watermark event, so more data can be received from UART.



```
# Attribute client indicated event listener
event attclient_indicated(connection, attrhandle)
    if attrhandle = xgatt_data then                                   # Cable replacement data value is indicated
        call system_endpoint_set_watermarks(5, 1, $ff)                # set RX watermark to 1 byte
        in_len = 0
    end if
end
```

Figure 14: Indication event listener

4.6 Compiling and installing the firmware

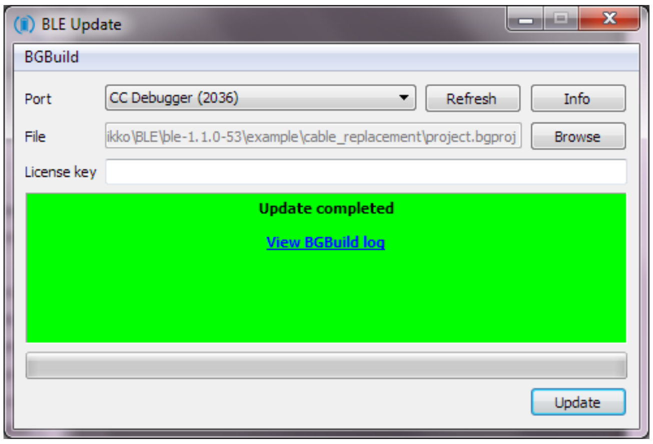
4.6.1 Using BLE Update tool

When you want to test your project, you need to compile the hardware settings, the GATT data base and BGScript code into a firmware binary file. The easiest way to do this is with the BLE Update tool that can be used to compile the project and install the firmware to a BLE112 module using a CC debugger.

In order to compile and install the project:

1. Connect CC debugger to the PC via USB
2. Connect the CC debugger to the debug interface on the BLE112
3. Press the button on CC debugger and make sure the led turns green
4. Start **BLE Update** tool
5. Make sure the CC debugger is shown in the **Port** drop down list
6. Use Browse to locate your **project.xml** file
7. Press **Update**

   BLE Update tool will compile the project and install it into the target device.



Port: CC Debugger (2036)

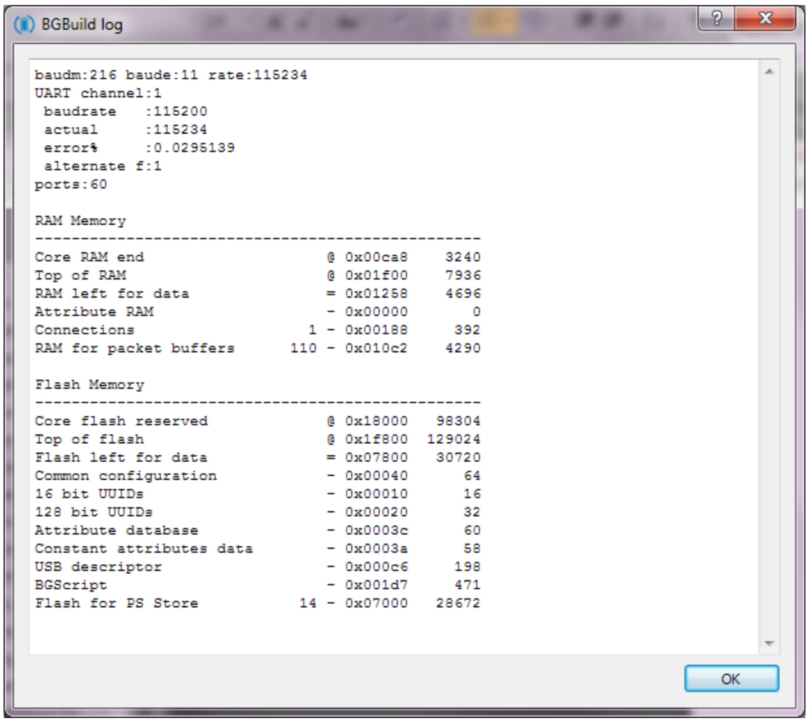
File: ikko\BLE\ble-1.1.0-53\example\cable_replacement\project.bgproj

License key:

Update completed

Figure 15: Compile and install with BLE Update tool

The **View Build Log** opens up a dialog that shows the bgbuild compilere output and the RAM and Flash memory allocations.


BGBuild log

```
baudm:216 baude:11 rate:115234
UART channel:1
 baudrate   :115200
 actual     :115234
 error%     :0.0295139
 alternate f:1
ports:60

RAM Memory
-------------------------------------------------
Core RAM end                    @ 0x00ca8    3240
Top of RAM                      @ 0x01f00    7936
RAM left for data               = 0x01258    4696
Attribute RAM                   - 0x00000       0
Connections                   1 - 0x00188     392
RAM for packet buffers      110 - 0x010c2    4290

Flash Memory
-------------------------------------------------
Core flash reserved             @ 0x18000   98304
Top of flash                    @ 0x1f800  129024
Flash left for data             = 0x07800   30720
Common configuration            - 0x00040      64
16 bit UUIDs                    - 0x00010      16
128 bit UUIDs                   - 0x00020      32
Attribute database              - 0x0003c      60
Constant attributes data        - 0x0003a      58
USB descriptor                  - 0x000c6     198
BGScript                        - 0x001d7     471
Flash for PS Store           14 - 0x07000   28672
```

Figure 16: BLE Update build log

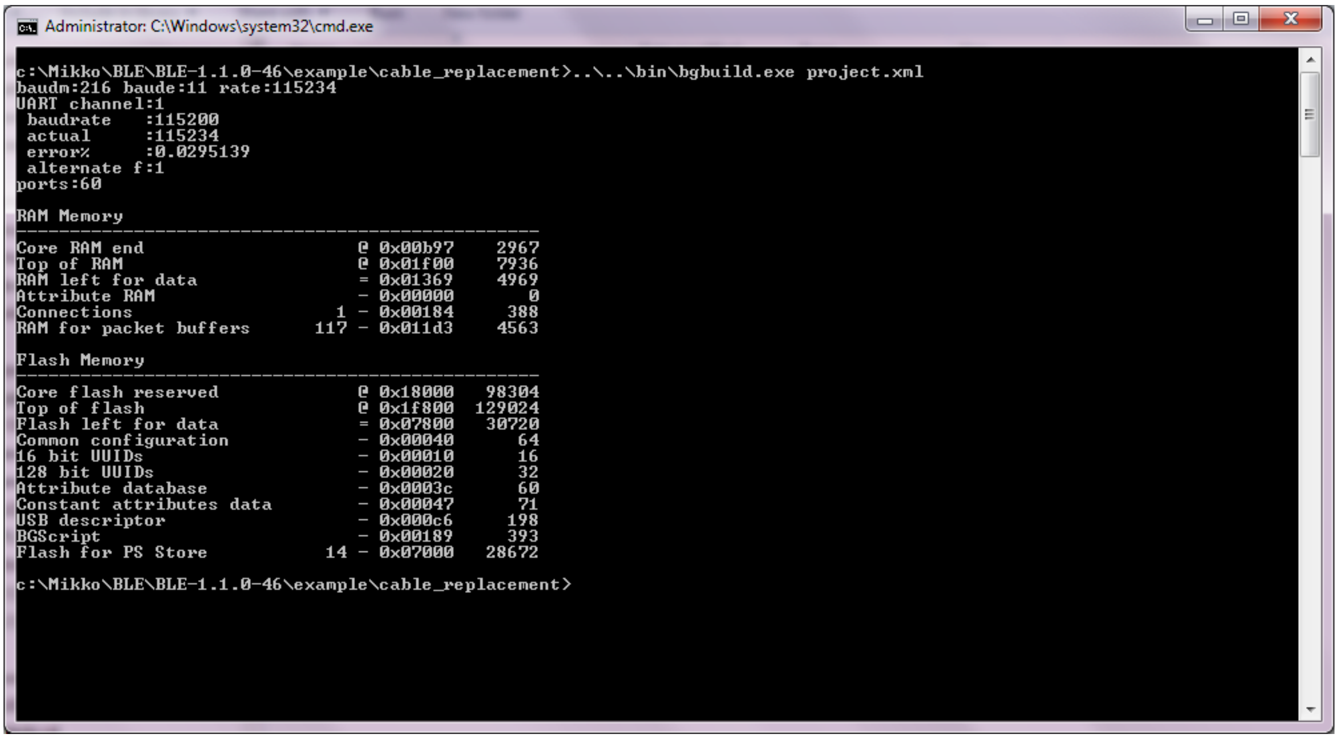
4.6.2 Compiling using the bgbuild.exe

The project can also be compiled with the **bgbuild.exe** command line compiler. The BGBuild compiler simply generates the firmware image file, which can be installed to the BLE112.

In order to compile the project using BGBuild:

1. Open Windows Command Prompt (cmd.exe)
2. Navigate to the directory where your project is
3. Execute BGbuild.exe compiler

   Syntax: ***bgbuild.exe <project file>***



```
c:\Mikko\BLE\BLE-1.1.0-46\example\cable_replacement>..\..\bin\bgbuild.exe project.xml
baudm:216 baude:11 rate:115234
UART channel:1
 baudrate   :115200
 actual     :115234
 error%     :0.0295139
 alternate f:1
ports:60

RAM Memory
-------------------------------------------------
Core RAM end                    @ 0x00b97    2967
Top of RAM                      @ 0x01f00    7936
RAM left for data               = 0x01369    4969
Attribute RAM                   - 0x00000       0
Connections                   1 - 0x00184     388
RAM for packet buffers      117 - 0x011d3    4563

Flash Memory
-------------------------------------------------
Core flash reserved             @ 0x18000   98304
Top of flash                    @ 0x1f800  129024
Flash left for data             = 0x07800   30720
Common configuration            - 0x00040      64
16 bit UUIDs                    - 0x00010      16
128 bit UUIDs                   - 0x00020      32
Attribute database              - 0x0003c      60
Constant attributes data        - 0x00047      71
USB descriptor                  - 0x000c6     198
BGScript                        - 0x00189     393
Flash for PS Store           14 - 0x07000   28672

c:\Mikko\BLE\BLE-1.1.0-46\example\cable_replacement>
```

Figure 17: Compiling with BGBuild.exe

If the compilation is successful a .HEX file is generated, which can be installed into a BLE112 module.

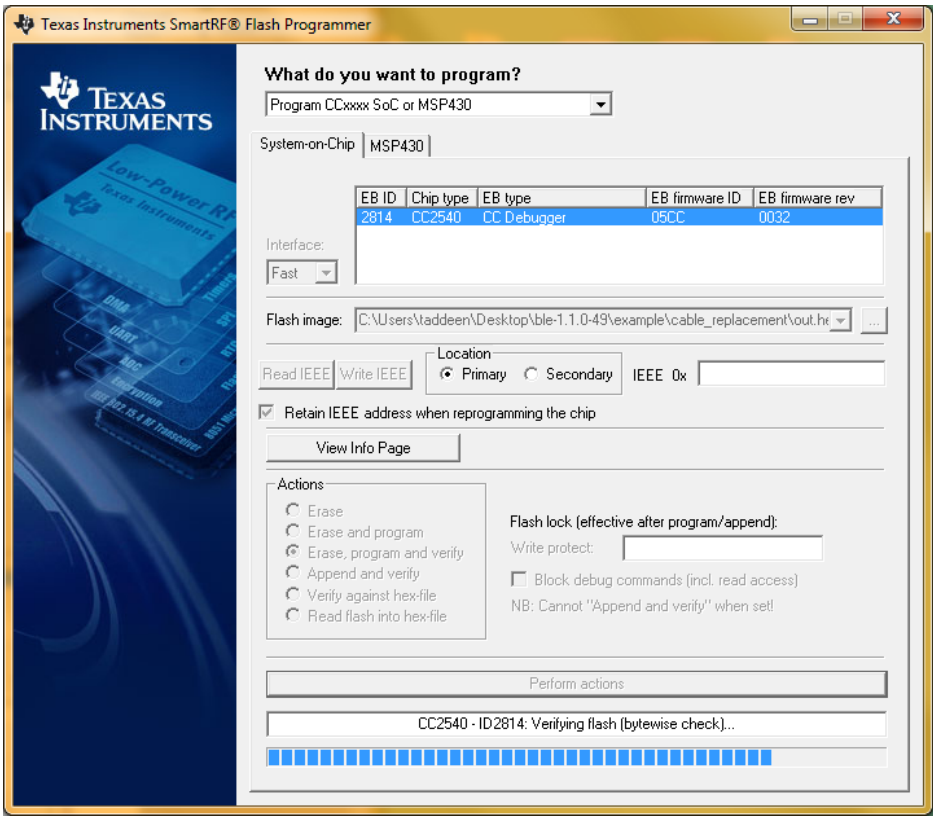
On the other hand if the compilation fails due to syntax errors in the BGScript or GATT files, and error message is printed.

4.6.3 Installing the firmware with TI's flash tool

Texas Instruments flash tool can also be used to install the firmware into the target device using the CC debugger.

In order to install the firmware with TI flash tool:

1. Connect CC debugger to the PC via USB
2. Connect the CC debugger to the debug interface on the BLE112
3. Press the button on CC debugger and make sure the led turns green
4. Start **TI flash tool** tool
5. Select program **CCxxxx SoC or MSP430**
6. Make sure the target device is recognized and displayed in the System-on-Chip field
7. Make sure **Retain IEEE address..** field is checked
8. Select the .HEX file you want to program to the target device
9. Select **Erase, Program and Verify**
10. Finally press **Perform actions** and make sure the installation is successful


Texas Instruments SmartRF® Flash Programmer

What do you want to program?

Program CCxxxx SoC or MSP430

System-on-Chip | MSP430

EB ID	Chip type	EB type	EB firmware ID	EB firmware rev
2814	CC2540	CC Debugger	05CC	0032

Interface: Fast

Flash image: C:\Users\taddeen\Desktop\ble-1.1.0-49\example\cable_replacement\out.he

Read IEEE | Write IEEE | Location: [✓ Primary] Secondary | IEEE 0x

[✓] Retain IEEE address when reprogramming the chip

View Info Page

Actions:
- Erase
- Erase and program
- [✓ Erase, program and verify]
- Append and verify
- Verify against hex-file
- Read flash into hex-file

Flash lock (effective after program/append):

Write protect:

Block debug commands (incl. read access)

NB: Cannot "Append and verify" when set!

Perform actions

CC2540 - ID2814: Verifying flash (bytewise check)...

Figure 18: TI's flash programmer tool

Note:

TI Flash tool should **NOT** be used with the Bluegiga *Bluetooth* Smart SDK v.1.1 or newer, but BLE Update tool should be used instead. The BLE112 and BLED112 devices contain a security key, which is needed for the firmware to operate and if the device is programmed with TI flash tool, this security key will be erased.

5 Testing the Cable Replacement device

5.1 Testing with BLEGUI software

This section describes how to test the Cable Replacement DevA application with BLEGUI software.

5.1.1 Discovering the Cable Replacement DevA

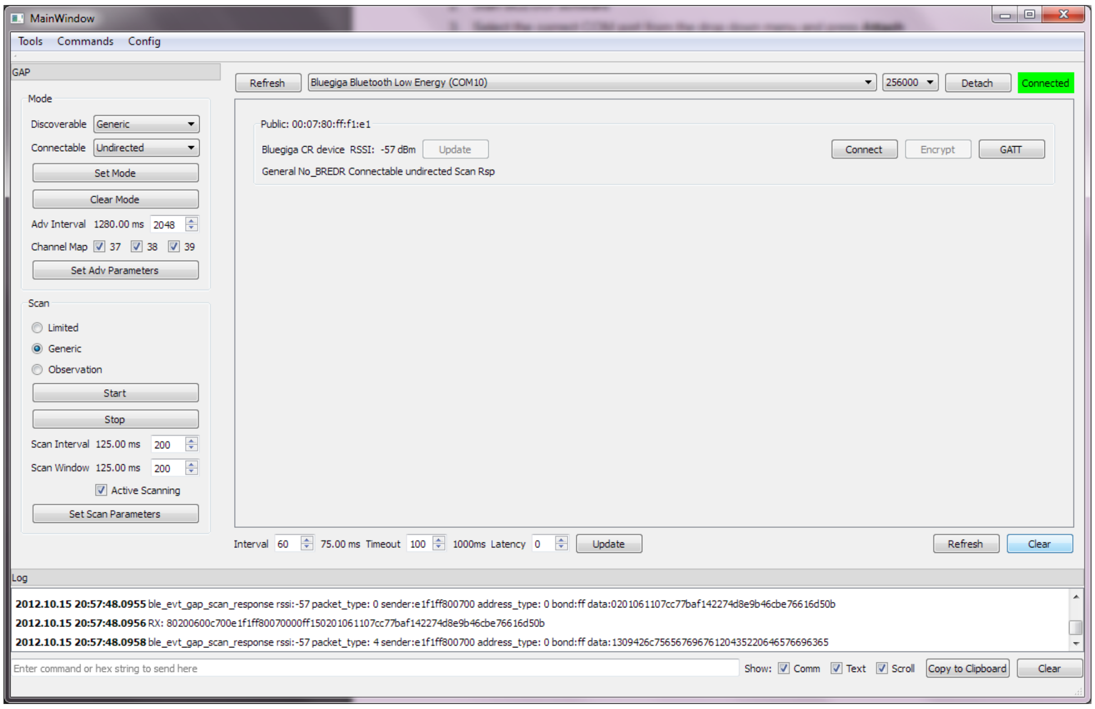
As soon as the Cable Replacement DevA is powered on it starts to advertise itself. For example a BLED112 USB dongle can be used to detect for the DevA by using the BLEGUI software.

Preparations:

1. Connect BLED112 USB dongle to a PC
2. Start BLEGUI software
3. Select the correct COM port from the drop down menu and press **Attach**
4. Execute for example **Command – Info** command to make sure the communication works

Discovering DevA:

1. Set desired scan parameters, check **Active Scanning** box and press **Set Scan Parameter** button
2. Select **Generic** scanning mode and **Start** scanning
3. If the DevA is power on, in range and not connected, it should appear in the main view.



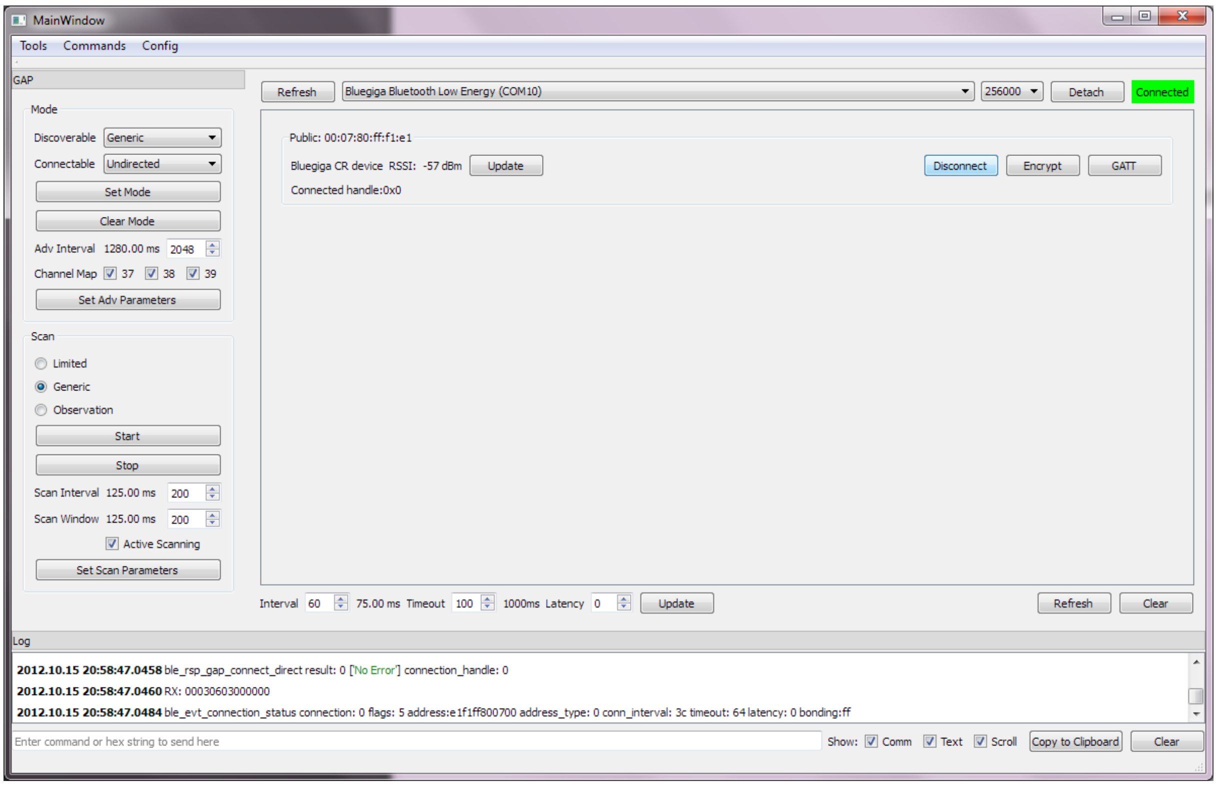
```
Public: 00:07:80:ff:f1:e1
Bluegiga CR device  RSSI:  -57 dBm
General No_BREDR Connectable undirected Scan Rsp
```

```
2012.10.15 20:57:48.0955 ble_evt_gap_scan_response rssi:-57 packet_type: 0 sender:e1f1ff800700 address_type: 0 bond:ff data:0201061107cc77baf142274d8e9b46cbe76616d50b
2012.10.15 20:57:48.0956 RX: 80200600c700e1f1ff80070000ff150201061107cc77baf142274d8e9b46cbe76616d50b
2012.10.15 20:57:48.0958 ble_evt_gap_scan_response rssi:-57 packet_type: 4 sender:e1f1ff800700 address_type: 0 bond:ff data:1309426c7565676967612043522064657669636365
```

Figure 19: Scanning for Cable Replacement DevA

5.1.2 Establishing a *Bluetooth* connection

1. Press the **Connect** button located next to the device you want to connect to
   a. If the connection is successful the connect button will change to **Disconnect**
   b. If the connection fails an error message is displayed in the **Log** view



Public: 00:07:80:ff:f1:e1
Bluegiga CR device RSSI: -57 dBm
Connected handle:0x0

Interval 60 75.00 ms Timeout 100 1000ms Latency 0

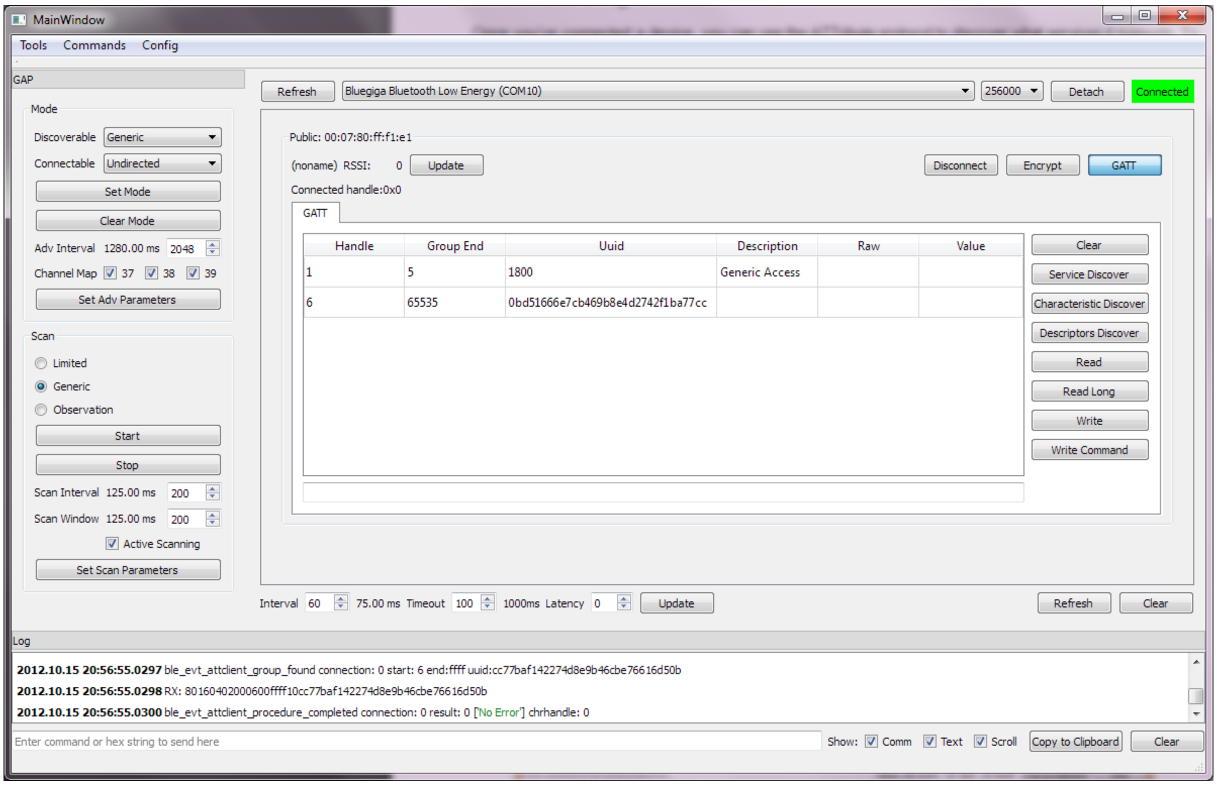
```
2012.10.15 20:58:47.0458 ble_rsp_gap_connect_direct result: 0 ['No Error'] connection_handle: 0
2012.10.15 20:58:47.0460 RX: 00030603000000
2012.10.15 20:58:47.0484 ble_evt_connection_status connection: 0 flags: 5 address:e1f1ff800700 address_type: 0 conn_interval: 3c timeout: 64 latency: 0 bonding:ff
```

Figure 20: Connected to Cable Replacement DevA

5.1.3 Discovering services

Once you've connected a device, you can use the ATTribute protocol to discover what services it supports. To discover the services of the Cable Replacement DevA:

1. Press the ***GATT*** button of the device you've just connected in order to start GATT tool
2. Press ***Service Discover*** button to start a GATT primary service discovery procedure



Handle	Group End	Uuid	Description	Raw	Value
1	5	1800	Generic Access		
6	65535	0bd51666e7cb469b8e4d2742f1ba77cc			

```
2012.10.15 20:56:55.0297 ble_evt_attclient_group_found connection: 0 start: 6 end:ffff uuid:cc77baf142274d8e9b46cbe76616d50b
2012.10.15 20:56:55.0298 RX: 80160402000600ffff10cc77baf142274d8e9b46cbe76616d50b
2012.10.15 20:56:55.0300 ble_evt_attclient_procedure_completed connection: 0 result: 0 ['No Error'] chrhandle: 0
```

Figure 21: GATT service discovery

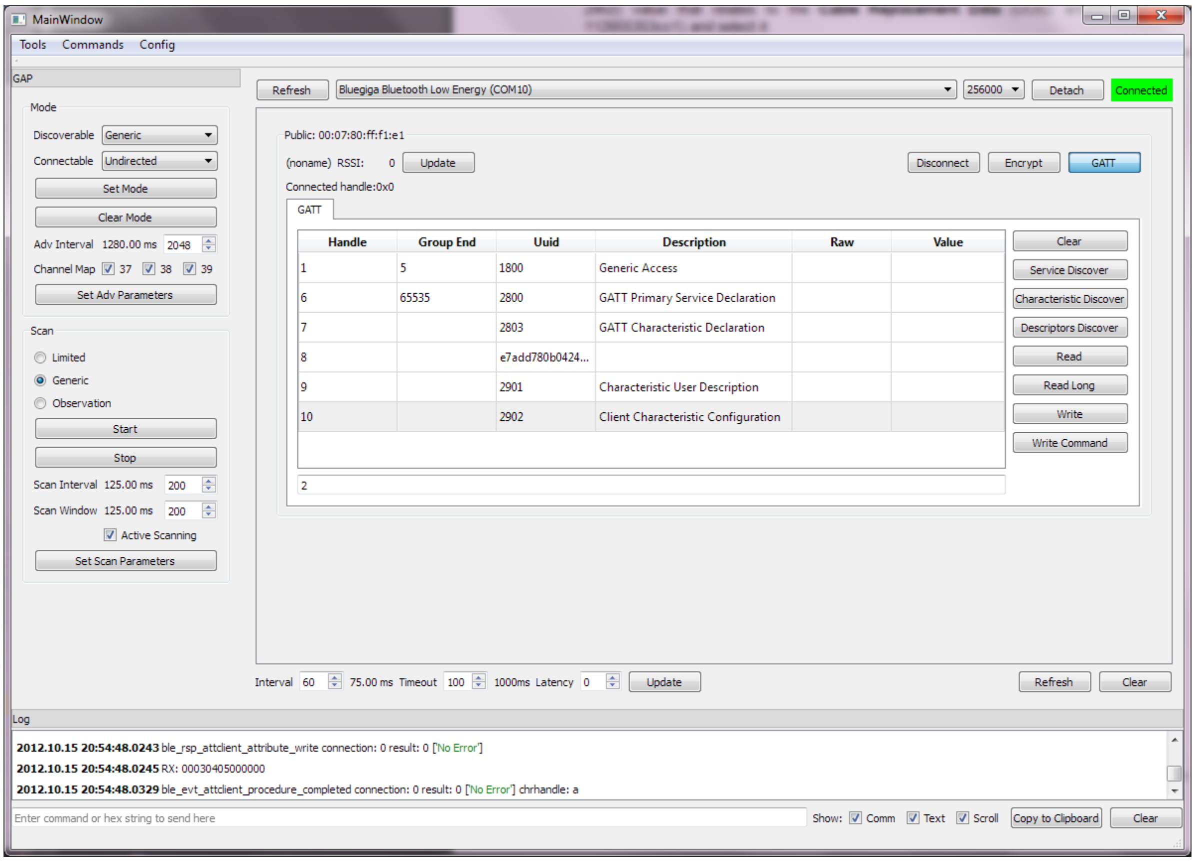
The two services defined in the **gatt.xml** should be visible in the GATT tool.

5.1.4 Receiving data from DevA

Data can be received from DevA with the attribute protocol's indication operation. Indication is a reliable way of transmitting data from the ATT Server to the ATT client and they are generated whenever the ATT server has some new data to send to the client. Indications can carry up to 20 bytes of application data.

Indications are not sent by default, but the ATT client needs to enable them first. In order to enable the indications:

1. Select the Cable Replacement Service in the BLEGUI's GATT view
2. Press **Descriptors discover** in order to see all characteristics and descriptors in the CR service.
3. Once the descriptors discovery is complete, select the **Client Characteristics Configuration** (UUID: 2902) value that relates to the **Cable Replacement Data** (UUID: e7add780-b042-4876-aae1-112855353cc1) and select it
4. In order to enable indications for the **Cable Replacement Data,** use **Write** button to write value 2 to the **Client Characteristics Configuration.**
5. Finally make sure the write operation is executed properly (see Log)



Handle	Group End	Uuid	Description	Raw	Value
1	5	1800	Generic Access		
6	65535	2800	GATT Primary Service Declaration		
7		2803	GATT Characteristic Declaration		
8		e7add780b0424...			
9		2901	Characteristic User Description		
10		2902	Client Characteristic Configuration		

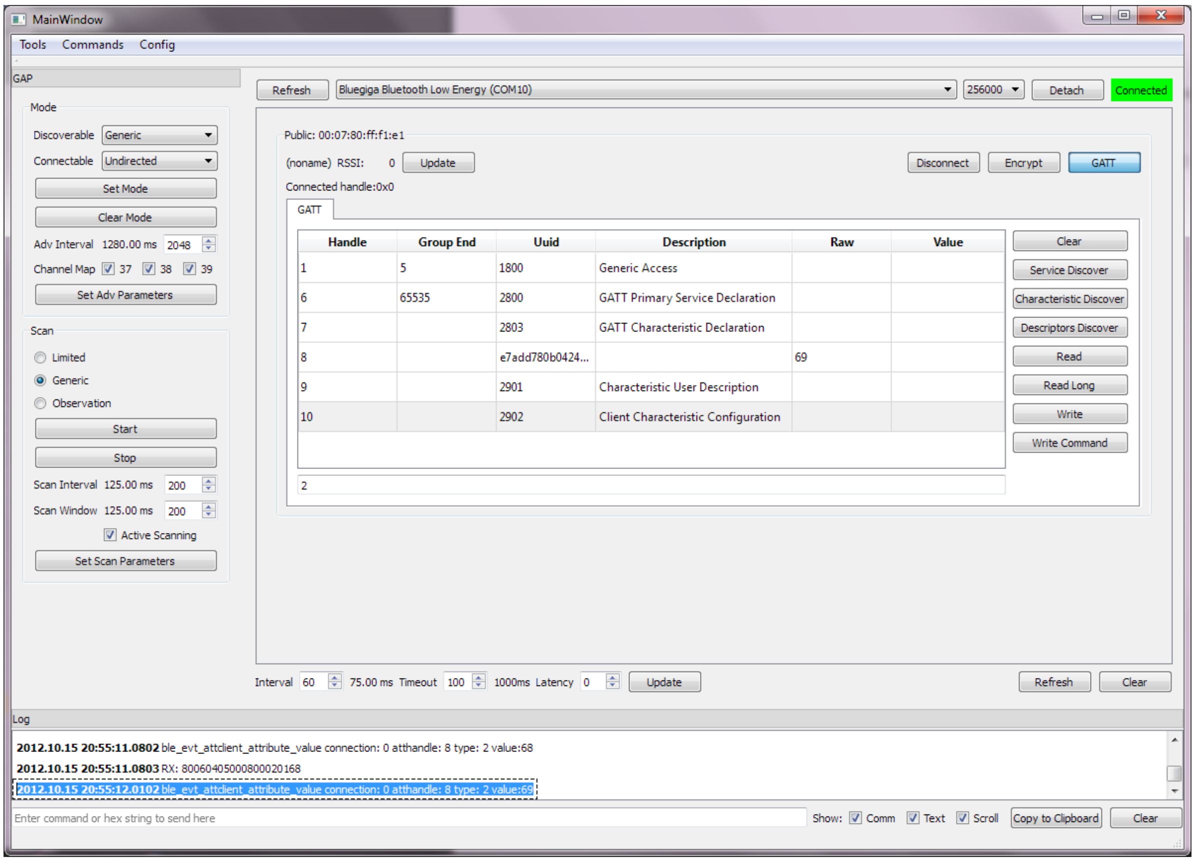
2

```
2012.10.15 20:54:48.0243 ble_rsp_attclient_attribute_write connection: 0 result: 0 ['No Error']
2012.10.15 20:54:48.0245 RX: 00030405000000
2012.10.15 20:54:48.0329 ble_evt_attclient_procedure_completed connection: 0 result: 0 ['No Error'] chrhandle: a
```

Figure 22: Enabling indications

Once the indications have been enabled data can be sent form the DevA to the DevB. In order to send data to the BLEGUI, do the following steps:

1. If you are using DKBLE112 as the DevB, make use the RS232 driver is enabled
    a. Notice that RS232 driver cannot be used with a CR2032 battery, but the DKBLE112 must be power via USB
2. Connect the RS232 interface for example to a PC
3. Open a terminal software with the settings you have defined in **hardware.xml**
    a. **Default:** 115200bps, 8 data bits, no parity, 1 stop bit and flow control enabled
4. Send data from the terminal software
5. Monitor BLEGUI software and the **Cable Replacement Data** attribute



Handle	Group End	Uuid	Description	Raw	Value
1	5	1800	Generic Access		
6	65535	2800	GATT Primary Service Declaration		
7		2803	GATT Characteristic Declaration		
8		e7add780b0424...		69	
9		2901	Characteristic User Description		
10		2902	Client Characteristic Configuration		

```
2012.10.15 20:55:11.0802 ble_evt_attclient_attribute_value connection: 0 atthandle: 8 type: 2 value:68
2012.10.15 20:55:11.0803 RX: 800604050008000020168
2012.10.15 20:55:12.0102 ble_evt_attclient_attribute_value connection: 0 atthandle: 8 type: 2 value:69
```

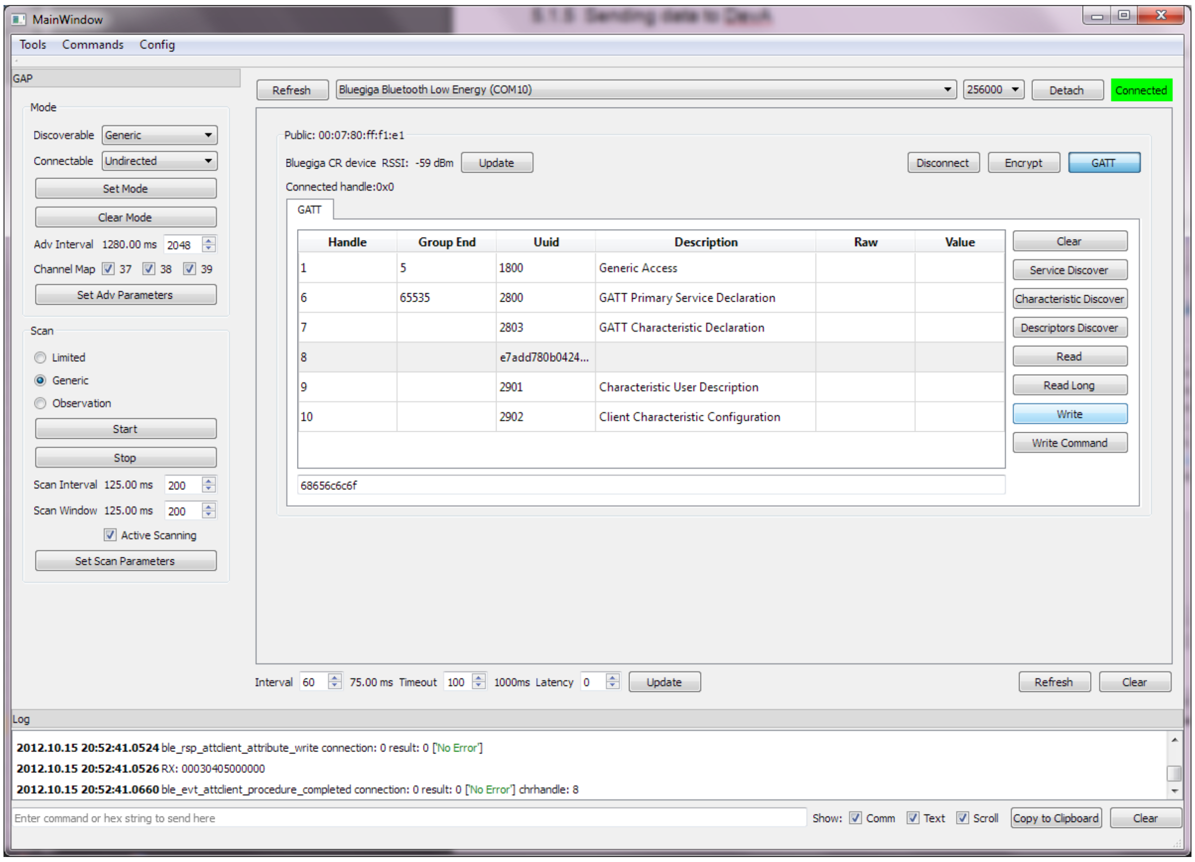
Figure 23: Data indication events

5.1.5 Sending data to DevA

The **Cable Replacement Data** attribute has a write property, which allows one to reliable **write** up to 20 bytes of application data to the DevA.

In order to send data to DevA, simply:

1. Select the **Cable Replacement Data** attribute from the BLEGUI
2. Type the data you want to send to the DevA in to the text filed below the GATT table.
3. Press the ***Write*** button, in order to execute the attribute write procedure
4. Verify from Log view that no error is received



Discoverable: Generic
Connectable: Undirected
Adv Interval 1280.00 ms 2048
Channel Map ☑ 37 ☑ 38 ☑ 39

○ Limited
● Generic
○ Observation

Scan Interval 125.00 ms 200
Scan Window 125.00 ms 200
☑ Active Scanning

Bluegiga Bluetooth Low Energy (COM10) 256000

Public: 00:07:80:ff:f1:e1
Bluegiga CR device RSSI: -59 dBm
Connected handle:0x0

Handle	Group End	Uuid	Description	Raw	Value
1	5	1800	Generic Access		
6	65535	2800	GATT Primary Service Declaration		
7		2803	GATT Characteristic Declaration		
8		e7add780b0424...			
9		2901	Characteristic User Description		
10		2902	Client Characteristic Configuration		

68656c6c6f

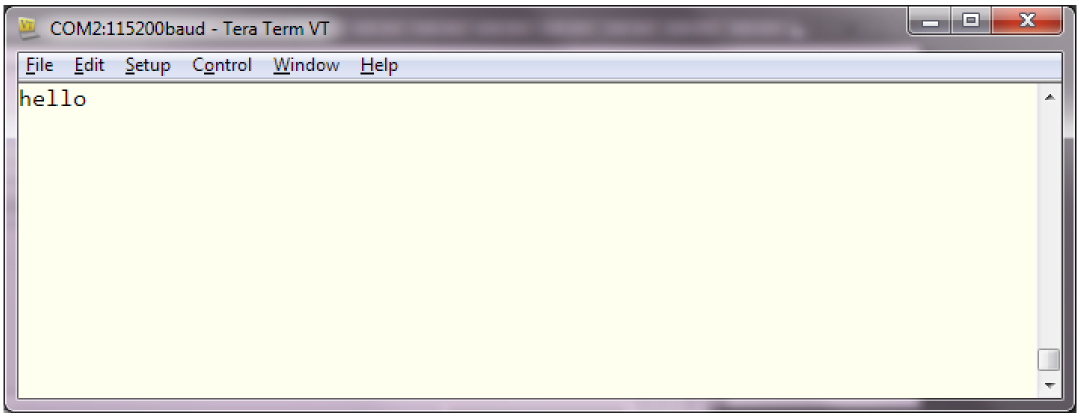
Interval 60 75.00 ms Timeout 100 1000ms Latency 0

```
2012.10.15 20:52:41.0524 ble_rsp_attclient_attribute_write connection: 0 result: 0 ['No Error']
2012.10.15 20:52:41.0526 RX: 00030405000000
2012.10.15 20:52:41.0660 ble_evt_attclient_procedure_completed connection: 0 result: 0 ['No Error'] chrhandle: 8
```

Enter command or hex string to send here
Show: ☑ Comm ☑ Text ☑ Scroll

Figure 24: Sending data to DevA

If DKBLE112 is used to receive data over RS232 interface and the DKBLE112 is connected to a terminal software, the transmitted data can be observerd on the terminal.


COM2:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
hello
```

Figure 25: Terminal connected to DevA

5.2 Testing with iPhone or iPad

Bluegiga provides a simple example application for iPhone 4S and 5 and iPad 3, which can be used to test the cable replacement application and exchange data.

This section briefly describes how to test the Cable Replacement application using an iPhone.

5.2.1 Getting the App

Bluegiga provides the cable replacement App as a source code and it can be downloaded from Bluegiga's Tech Forum.

You need to have Apple xCode SDK in order to compile and install the application into your iPhone or iPad.

5.2.2 Testing the App

Once you've installed the App to your iPhone or iPad, the following steps are required:

1. Power of the DevA (for example DKBLE112)
2. Start the iPhone application
3. Once the App has started, turn on the scanner in order to discover the DevA

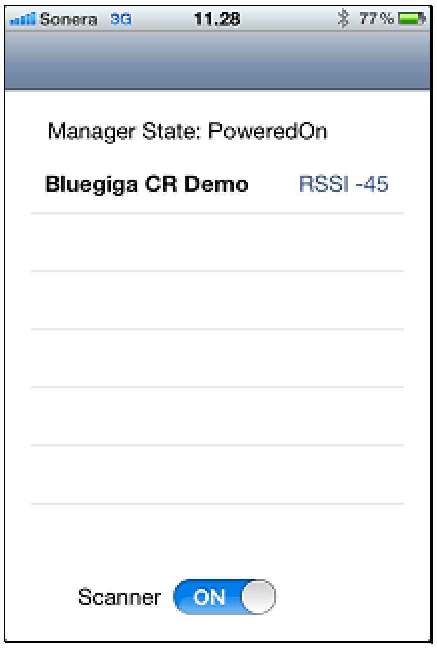
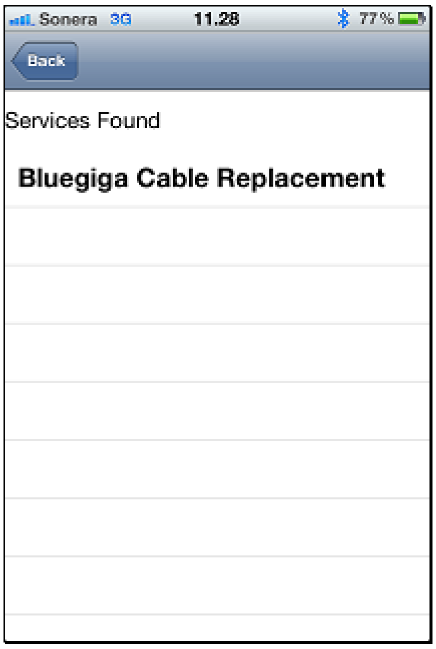


Figure 26: Scanning DevA

In order to connect a device:

1. Select the device you want to connect to and the App will show available services on this device
2. Select the **Bluegiga Cable Replacement** service

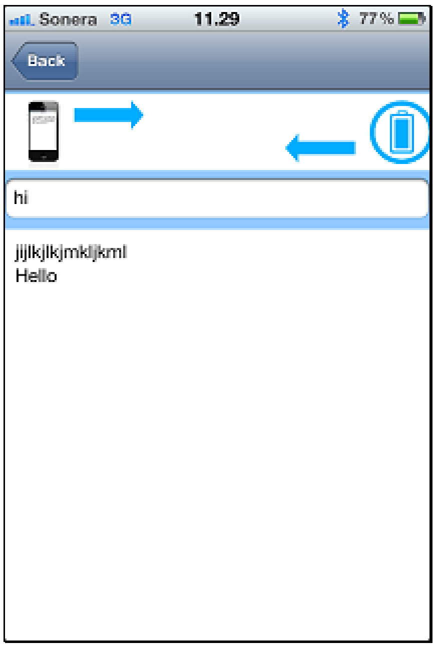


Services Found

Bluegiga Cable Replacement

Figure 27: Available services

3. A new window will open which simple allows you to exchange data with the Cable Replacement DevA.
4. Type data to the text field and it will be sent over a *Bluetooth* LE connection to the remote device.



hi

jijlkjlkjmkljkml
Hello

Figure 28: Data exchange

6 Appendix

6.1 External resources

- *Bluetooth* 4.0 software development kit is available at : http://techforum.bluegiga.com
- BLE112 and DKBLE112 hardware documentation is available at : http://techforum.bluegiga.com
- Project files of Cable Replacement Profile are in the SDK zip at: http://techforum.bluegiga.com
- *Bluetooth* SIG's developer portal: https://developer.*Bluetooth*.org/

7 Contact information

Sales:	sales@bluegiga.com
Technical support:	support@bluegiga.com http://techforum.bluegiga.com
Orders:	orders@bluegiga.com
WWW:	www.bluegiga.com www.bluegiga.hk

Head Office / Finland:

Phone: +358-9-4355 060
Fax: +358-9-4355 0660
Sinikalliontie 5A
02630 ESPOO
FINLAND

Postal address / Finland:

P.O. BOX 120
02631 ESPOO
FINLAND

Sales Office / USA:

Phone: +1 770 291 2181
Fax: +1 770 291 2183
Bluegiga Technologies, Inc.
3235 Satellite Boulevard, Building 400, Suite 300
Duluth, GA, 30096, USA

Sales Office / Hong-Kong:

Phone: +852 3182 7321
Fax: +852 3972 5777
Bluegiga Technologies, Inc.
19/F Silver Fortune Plaza, 1 Wellington Street,
Central Hong Kong