# QSG178: BT122 Quick-Start Guide

BT122 is a dual-mode Bluetooth BR/EDR - BLE module dedicated for applications that require both Bluetooth Low Energy and Classic connectivity. It can connect to legacy devices that only support Bluetooth SPP or Apple iAP2 profiles as well as to devices that support Bluetooth Low Energy. BT122 integrates a high-performance Bluetooth radio, a low-power ARM Cortex microcontroller, and a Silicon Laboratories BGScript stack software. BT122 can be used as a modem together with an external host MCU, but applications can also be embedded into the built-in ARM Cortex MCU with the Silicon Labs BGScript scripting language.

The document contains a few migration guides both for BT121 and iWRAP devices to their replacement – BT122. In the first chapter, the BT122 architecture is described and the BGAPI interface and BGScript scripting language are briefly covered. The second chapter describes the differences between BT121 and BT122 devices and describes the method of migration between them. The third chapter describes differences between BT121 and iWRAP devices and the method of migration between these two modules. The fourth chapter describes a tool that simplifies the migration between iWRAP and BT122 devices (BIMBER migration tool). The fifth chapter compares different firmware update methods between all prior covered modules.

**KEY POINTS**

- BT122 Architecture
- Differences between BT121 and BT122
- Differences between iWRAP and BT122
- iWRAP to BT122 migration tool
- BT122 firmware update guide

# 1. Version History

**Table 1.1. Version History with Comments**

| Version | Comment |
|---|---|
| 1.0 | Initial version |
| 1.1 | Updated UART/SPI interface information<br>Updated BGTool figures |
| 1.2 | Updates related to new development board hardware revision |

# 2. BT122 Architecture

The SDK implements a full Bluetooth BR/EDR and LE compatible Bluetooth Stack including support of L2CAP, RFCOMM, SMP, ATT protocols, Bluetooth SPP, HID, Apple iAP2, and GATT (over BR/EDR and BLE).

The Bluetooth Low Energy Software is also supported by a complete SDK for developing Bluetooth Low Energy applications using either an external host or BGAPI serial protocol over UART or fully standalone applications based on a simple scripting language called BGScript.

## 2.1 BT122 SDK Components and Tools

- **BGLIB** is a host library for external MCUs, and it implements a reference parser for the BGAPI serial protocol. BGLIB is delivered in C source code as part of the Bluetooth Dual Mode SDK, and it can be easily ported to various processor architectures.
- **BGBuild** compiler is a free-of-charge compiler that compiles the Bluetooth Smart Ready Stack, the BGScript application, and the Bluetooth GATT services into the firmware binary that can be installed to the Bluetooth Smart Ready modules.
- **BGTool** is a graphical user interface application and a developer tool, which allows the Bluetooth Dual Mode module to be controlled over the host interface using the BGAPI serial protocol. BGTool is a useful tool for testing the Bluetooth Dual Mode module and evaluating its functionality and APIs.
- **Device Firmware Upgrade (DFU) Tools** are also included as part of the SDK. They allow the firmware to be updated over the UART interface or J-Link.
- **BIMBER** is a simple migrator tool created because the iWRAP modules firmware is no longer supported. The migrator allows the user to convert a dump of persistent storage from iWRAP modules into BT122 project files (including BGScript).

## 2.2 BGScript

BGScript is a simple BASIC-style programming language that allows end-user applications to be embedded in the Silicon Labs Bluetooth modules. The benefit of using BGScript is that one can create fully standalone Bluetooth devices without the need of an external MCU, and this enables further size, cost, and power consumption reductions. Although being a simple and easy-to-learn programming language, BGScript does provide features and functions to create complex and powerful applications, and it provides the necessary APIs for managing Bluetooth connections, security, data transfer, and various hardware interfaces, such as UART, I$^2$C, GPIO, and ADC.

BGScript is a fully event-based programming language, and code execution is started when events such as system start-up, Bluetooth connection, I/O interrupt etc., occur.

BGScript applications are developed with Silicon Labs Bluetooth Modules SDKs, and the BGScript applications are executed in the BGScript Virtual Machine (VM) that is part of the BGStack software. The Bluetooth Dual Mode SDK comes with all the necessary tools for code editing, compilation, and the needed tools for installing the compiled firmware binary to the Silicon Labs BT122 modules. Multiple example demos and code snippets are also available from Silicon Labs for implementing applications like thermometers, heart-rate transmitters, medical sensors, iBeacons, etc.

### 2.2.1 BGScript Syntax Example

The example snipped code below shows the syntax for BGScript. This sample also shows how to create connectable devices in BR/EDR mode. Comments in the code clarify the content of the functions after the # sign.

```
Procedure PrepareToClassicConnection()
  #Set user-friendly server name
  call system_set_local_name(18, “BT122 Hello World!”)
  # Configure Bluetooth Security Manager
  # No MITM required, no input/output -> Just Works pairing mode
  call sm_configure(0,3)
  # Enable bonding mode
  call sm_set_bondable(1)
  # Set connectable and discoverable
  call bt_gap_set_mode(1, 1, 0)
  # Start RFCOMM server
  call bt_rfcomm_start_server(2, 0)
end
```

### 2.3 BGAPI

**BGAPI** is a binary serial protocol API to the Silicon Labs Bluetooth Dual Mode stack over UART interface. BGAPI is intended for users who want to use both Bluetooth BR/EDR and LE functionality and all the features in the Bluetooth Dual Mode stack from an external host, such as a low-power MCU.

BGAPI requires that a complete, valid packet be transmitted. All BGAPI commands are described in the BT122 API Reference Manual.

#### 2.3.1 BGAPI Packet Structure

Every BGAPI packet has a 4-byte header. A packet may have between 0-255 payload data bytes after header.

**Table 2.1. BGAPI Packet Structure**

| Byte | Name | Description |
|---|---|---|
| 0 | hilen | Message type |
| 1 | lolen | Minimum payload length |
| 2 | class ID | Command class ID |
| 3 | command ID | Command ID |
| 4-n | payload | Up to 255 bytes of payload |

Message type can have two values:

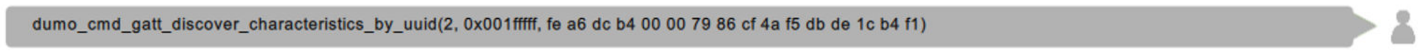
- 0x20 for commands and responses
- 0x80 for events

#### 2.3.2 Sleep Modes During BGAPI Use

The BT122 module optionally uses sleep modes to minimize power consumption. If sleep is allowed, the module will not receive UART data from an external microcontroller or other UART-connected host while asleep. Use the tag in *hardware.xml* to configure a pin, which can be asserted to wake up the module while sending data. Events and responses coming from the module will automatically wake up from sleep if necessary.

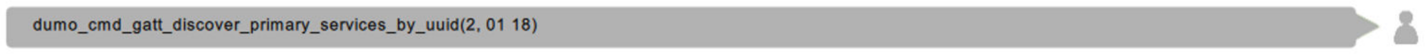
### 2.3.3 Sending Arrays of Data in Payload

All multi-byte, integer-based data payload fields (int16, uint16, int32, uint32, bd_addr) use **little-endian byte order**. The **uint8array** data type is made up of a **length** byte followed by that number of data bytes. The three figures below present examples of putting different multi-byte types into BGAPI by BGTool.



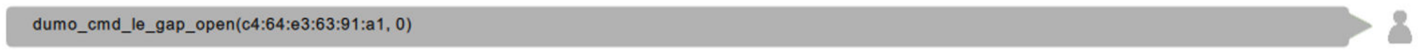
```
dumo_cmd_gatt_discover_characteristics_by_uuid(2, 0x001fffff, fe a6 dc b4 00 00 79 86 cf 4a f5 db de 1c b4 f1)
```

**Figure 2.1. Example of Multi-Byte Byte Types in BGTool (16-bit Service Handle(0x001fffff) and 128-bit Characteristic UUID(f1b41cde-dbf5-4acf-8679-0000b4dca6fe))**



```
dumo_cmd_gatt_discover_primary_services_by_uuid(2, 01 18)
```

**Figure 2.2. Example of Multi-Byte Byte Types in BGTool (16-bit Service UUID(1800))**



```
dumo_cmd_le_gap_open(c4:64:e3:63:91:a1, 0)
```

**Figure 2.3. Example of Multi-Byte Byte Types in BGTool (BT MAC Address in Little-Endian Byte Order)**

The code snippet below shows how to send multi-byte types in the BGLIB C library.

```
#include <bglib/cmd_def.h>
#include <bglib/uart.h>
#define ADDRESS_LENGTH 6
#define UUID16_LENGTH 2
/*Declaring address data (BT MAC address in little-endian order) and bd_addr structure used by API function*/
uint8 address[6] = {0xC4, 0x64, 0xE3, 0x63, 0x91, 0xA1};
bd_addr bt_address;
/*Declaring uuid data (SDP service 16-bit UUID - 0x1101) and uint8array structure used by API func-tion*/
uint8 uuid_data = {0x01, 0x11};
uint8array UUID;

int main(void) {
  /*Coping data into structer field*/
  memcpy(bt_address.addr, address, ADDRESS_LENGTH);
  memcpy(UUID.data, uuid_data, UUID16_LENGTH);
  UUID.len = UUID16_LENGTH;

  /*Sending data into BT122 device*/
  dumo_cmd_bt_rfcomm_open(&bt_address, 0, &UUID);
}
```

The next code snippet shows a way to send UUID128 of characteristic and uint32_t type service handle.

```
#include <bglib/cmd_def.h>
#include <bglib/uart.h>
#define UUID128_LENGTH 16

/*Declaring  uuid  128  data  (GATT  characteristic  128-bit  UUID  -  f1b41cde-dbf5-4acf-8679-0000b4dca6fe)  and
uint8array structure used by API function*/
uint8 uuid128_data = {0xFE, 0xA6, 0xDC, 0xB4, 0x00, 0x00, 0x79, 0x86, 0xCF, 0x4A, 0xF5, 0xDB, 0xDE,   0x1C,
0xB4, 0xF1};
uint8array UUID128;

int main(void) {
  memcpy(UUID128.data, uuid_data, UUID128_LENGTH);
  UUID128.len = UUID128_LENGTH;

  uint32 serivce_uuid = 0x001FFFFF;
  uint8 connection = 2;
  /*Sending data into BT122 device*/
  dumo_cmd_gatt_discover_characteristics_by_uuid(connection, serivce_uuid, &UUID128);
}
```

This last snippet shows how to put a multi-byte array in BGScript. This case presents UUID128 of characteristic, Bluetooth Mac Address, and uint32_t type service handles.

```
#Declaration of UUID128
dim uuid() = “\xfe\xa6\xdc\xb4\x00\x00\x79\x86\xcf\x4a\xf5\xdb\xde\x1c\xb4\xf1”
#Declaration of BT Mac Address
dim address() = "\xbb\x80\x91\xe9\x17\x00"
#Declaration of service handle
dim service = $001fffff
call gatt_discover_characteristics_by_uuid(0,service,16, uuid)()
```

## 2.4 BGAPI vs. BGScript

This section describes the differences between two operation modes of the BT122 module: Network Co-Processor (NCP) based on BGAPI protocol and Standalone with BGScript. In brief, the main differences are as follows:

- BGAPI is a custom serial protocol used to externally control the modules over host interface.
- BGScript is a simple BASIC-style programming language that allows end-user applications to be embedded in the Silicon Labs Bluetooth modules.

In more details, BGScript is a simple language which may allow implementing the applications to run on the Bluetooth module itself, thus allowing designers to provide a Bluetooth connectivity for their products at a lesser cost. However, it needs to be regarded as a better solution for low complexity behavior.

If the project, requires more complex operations during runtime of a device, there is NCP mode available which enables an external host MCU to control the Bluetooth module using BGAPI protocol. This approach allows a developer to move the application logic to the more advanced microcontroller unit, which would control the tasks of the module with appropriate commands via UART interface. Controlling the module in this way should also enable the use of a wider variety of debugging options, which are typically included in the most popular IDEs

For easier usage of the module in NCP mode, we provide BGLIB – ANSI C reference implementation of the BGAPI serial protocol.

**Table 2.2. Difference between BGAPI and BGScript**

| Question | BGAPI | BGScript |
|---|---|---|
| An external host needed? | YES | NO |
| Host interface | UART | No separate host needed |
| Bluetooth API | BGAPI or BGLIB APIs | BGScript API |
| Peripheral interface APIs and support | Host-dependent* | APIs for UART, I2C, GPIO, ADC, and PS store |
| Custom peripheral interface drivers | Can be developed to the host | Peripheral drivers are part of the BT122 Bluetooth Dual Mode stack |
| RAM available for application | Host-dependent | Application-dependent |
| Flash available for application | Host-dependent | Application-dependent |
| Execution speed | Host-dependent | Depends on the complexity of implementation of BGScript code |
| Bluetooth firmware / application updates | DFU over UART or J-Link | DFU over UART or J-Link |
| *<br><br>**Note:** The BT122 modules' peripheral interfaces are still available via BGAPI commands and can be used to extend the host MCUs I/Os. | | |

Advantages of both ways to control BT122 modules are presented below.

### 2.4.1 BGAPI Advantages

- Increased speed compared to BGScript.
- No interpreter resources are needed.
- More flexibility (functions, compiler directives, more RAM).

### 2.4.2 BGScript Advantages

- Decreased design complexity (module with BGScript does not require external host).
- More GPIOs available (host UART pins are free to use).
- Simplified usage.

# 3. Differences between BT121 and BT122

The BT122 module has been designed to replace the BT121 module. Therefore, these two modules have many features in common.

Both modules use the same Bluetooth stack, but in the case of BT122, the stack is improved and includes new functionalities. However, there are other hardware differences between the modules, most of which are outlined in the table below.

**Table 3.1. Differences between BT121 and BT122**

| Difference | BT121 | BT122 |
|---|---|---|
| MCU | ARM Cortex-M0 | ARM Cortex-M4 |
| MCU clock frequency | 48 MHz | 32MHz |
| MCU Flash size | 128 kB | 256kB |
| MCU RAM size | 16 kB | 32kB |
| Supported Bluetooth Core specification | 4.1 | supports 4.2 features, tested against 5.1 specification |
| BLE Secure Connection support | NO | YES |
| Filter Accept List support | NO | YES |
| GPIO pins number | 22 | 10 |
| 12-bit ADC channels | 4 | 3 |
| I$^2$C channels | 2 | 1 |
| SPI channels | 2 | 1 |

## 3.1 Bluetooth Stack

The most significant difference between the two modules is that the BT122 supports the 4.2 Bluetooth specification. As a result, BT122 module supports one of the Bluetooth 4.2 security features – BLE Secure Connection and implementation of Filter Accept List.

## 3.2 Hardware

In addition, Arm Cortex-M0 has been replaced with Arm Cortex-M4 in the BT122 module, making it more efficient. With twice as much Flash and RAM memory, BT122 can handle more complex applications with more advanced and much bigger BGScript code.

On the other hand, the BT121 had more I/Os than the BT122 (22 to 10). With a fewer number of GPIO pins, the BT122 module has only one I$^2$C channel as a result.

The BT122 has three GPIO ports (0 - A, 1 – C, 2 - F), and the BT121 has only two. Also, the BT122 has a slightly different system of interrupt numeration, due to the MCU specification. In the BT121, the number of interrupts is equal to the number of GPIO pins, which handle these interrupts. For instance, PA4 pin handles interrupt 4; also, PB4 handles the same interrupt number. If interrupts from PAX are selected, interrupts from PBX are disabled. In BT122, the interrupt assignment system is more sophisticated. In short, the MCU classifies interrupts in the key of modulo 4. So, for pins with number 0-3 for each port, only 4 interrupts (0-3) are available; for 4-7, only 4 interrupts (4-7), and analogously for the subsequent pins.

## 3.3 DFU Program Upgrade

Upgrading the module via UART DFU will differ significantly, especially when the device is running on BGScript with BGAPI disabled. The BGTool for the BT121 uses an ST vendor-specific UART bootloader protocol to upload the firmware, but BT122 does not support this protocol. The best method to reconcile BGScript with a DFU update is to implement the pin interrupt handler, which will enable the BGAPI mode of the device and reset it into bootloader mode. The other alternative, when BGAPI mode is not used, is to flash the device with SWD/JTAG interface in case the FW must be changed.

# 4. Differences between iWRAP and BT122

The BT122 module has also been designed as a replacement proposal for iWRAP modules. In that case, the difference between these two modules is significantly bigger. Most differences are presented in the table below.

**Table 4.1. Differences between iWRAP and BT122**

| Difference | iWRAP | BT122 |
|---|---|---|
| Bluetooth standard version support | 3.0 | supports 4.2 features, tested against 5.1 specification |
| BLE implementation | NO | YES |
| UART API | iWRAP API | BGAPI |
| BGScript support | NO | YES |
| External host need | YES | NO |
| Supported profiles | RFCOMM, SDP, SPP, HSP, HFP, DUN, OPP, FTP, A2DP, AVCRP, HID, PBAP, MAP, HDP, DI | RFCOMM, SPP, SDP, HID, DI |
| Data rate (over SPP) | 600 kbps | 1000 kbps |
| Output TX power | Up to 20dBm (for WT41) | Up to 11dBm |

## 4.1 Bluetooth Stack Support

The BT122 implements a much newer Bluetooth specification (BT v3.0 in iWRAP and BT v4.2 in BT122). Most importantly, BT122 supports Bluetooth Low Energy (BLE), which was introduced in the BT v4.0 Core specification. BLE significantly increases the range of possible applications and pays a key role in power optimization, which enables the user to develop products that work longer on a single battery charge.

## 4.2 Audio Support

BT122 does not support any audio features.

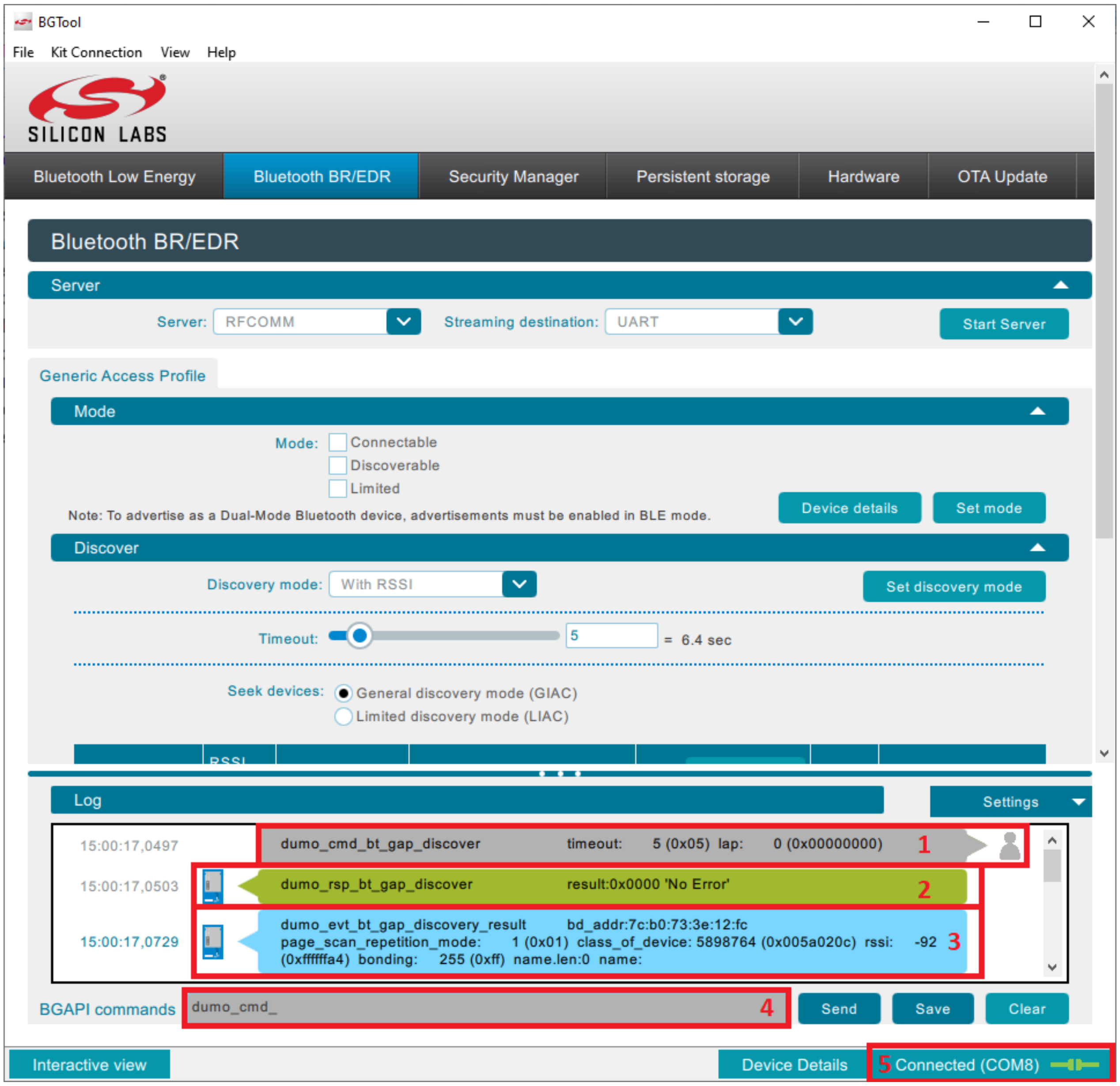
### 4.3 BGAPI

A quick introduction to BGAPI protocol was presented in Section 2.3 BGAPI. BGAPI is completely different from the iWRAP commands. The host application should be reworked.

First, in contrast to the WTXX modules, the BT122 does not require verifying a license key. After the flash erase and new firmware updates, the BT122 should work properly without any authentications. More information about firmware updates is presented in Section 6. Firmware Update.

The next big difference is in the command format. iWRAP commands are implemented in ASCII format. The BGAPI interpreter works differently. The BGAPI commands are declared in hex format (more information in Section 2.3.1 BGAPI Packet Structure).

There are two ways of using BGAPI from an external device. The BT122 SDK provides the BGTool program, which helps to ease the process of sending commands from a Windows PC.



Server: RFCOMM    Streaming destination: UART

Mode: ☐ Connectable ☐ Discoverable ☐ Limited

Note: To advertise as a Dual-Mode Bluetooth device, advertisements must be enabled in BLE mode.

Discovery mode: With RSSI

Timeout: 5 = 6.4 sec

Seek devices: [✓ General discovery mode (GIAC)] Limited discovery mode (LIAC)

```
15:00:17,0497  dumo_cmd_bt_gap_discover          timeout:    5 (0x05)  lap:    0 (0x00000000)
15:00:17,0503  dumo_rsp_bt_gap_discover          result:0x0000 'No Error'
15:00:17,0729  dumo_evt_bt_gap_discovery_result       bd_addr:7c:b0:73:3e:12:fc
               page_scan_repetition_mode:    1 (0x01)  class_of_device: 5898764 (0x005a020c)  rssi:    -92
               (0xffffffa4)  bonding:    255 (0xff)  name.len:0  name:
```

BGAPI commands: dumo_cmd_

Connected (COM8)

Figure 4.1. BGTool View

Command sender part of the BGTool contains the following elements:

1. Command sent to a device.
2. Response from a device.
3. Event received from a device.
4. Field to enter next command.

5. Currently connected serial port (after clicking this button, port can be changed).

When the user enters commands, the BGTool will prompt for possible command names and parameters so that the command will work properly. Additionally, the BGTool shows if all packets received from connected devices can be decoded as BGAPI. The application highlights each type of packet with another color: grey (command sent to device), green (positive response from device), red (negative response or error from device), and blue (events from device).

The BGTool is an excellent tool for quick testing and evaluation. But for application use, the BGLIB library is recommended. The BGLIB host library is a reference implementation of the BGAPI serial protocol parser, and it is provided in ANSI C source code in the Bluetooth Dual Mode SDK. It masks the complexity of the BGAPI serial protocol, and hex frames provide high-level C functions and call-back handlers to the application developer, which makes the application development easier and faster. The BGLIB library can be ported to various host systems, ranging from low-cost MCUs to devices running Linux, Windows, or OSX.

The snippets below present examples of BGLIB structures and functions used to build a Bluetooth application.

```
/* Function */
void dumo_cmd_bt_rfcomm_open( bd_addr *address,
                              uint8 streaming_destination,
                              uint8array uuid);
/* Response structure */
struct dumo_msg_bt_rfcomm_open_rsp_t
{
 uint16 result,
 uint8 endpoint
};

/* Response handler */
void dumo_rsp_bt_rfcomm_open(const struct dumo_msg_bt_rfcomm_open_rsp_t *msg);

/* Event structure */
struct dumo_msg_bt_rfcomm_opened_evt_t
{
 uint8 endpoint,
 bd_addr *address
};

/* Event handler*/
void dumo_evt_bt_rfcomm_opened(const struct dumo_msg_bt_rfcomm_opened_evt_t *msg);
```

### 4.4 BGScript and Hardware Configuration

In contrast to the iWRAP module, BT122 can easily work as a standalone device without an external host, thanks to the BGScript implementation. Section 2.2 BGScript describes this scripting language.

The BT122 SDK allows configuration of the hardware properties using an .xml file. After declaring peripheral use, most arguments are set as default values. The full list of possible configurations is presented in the BT122 Configuration Guide. Example code is presented below. Additionally, the BT122 SDK contains some useful examples and most of the implementations incorporate BGScript standalone applications. The XML code snippet below presents BT122 hardware configuration.

```
<hardware>
  <!-- Enable ADC internal channels: temperature sensor and supply voltage sensor -->
  <adc vdd="true" />
  <!-- Enable I2C -->
  <i2c pullup="true" />
  <!-- Sleep modes disabled -->
  <sleep enabled="false"/>
  <!-- UART enabled @115200bps -->
  <uart baud="115200" flowcontrol="true" bgapi="false"/>
  <!-- Configures PF2 and PF3 as inputs with falling interrupts enabled. -->
  <port index="2" input="0x000C" pullup="0x000C" interrupts_falling="0x000C" />
</hardware>
```

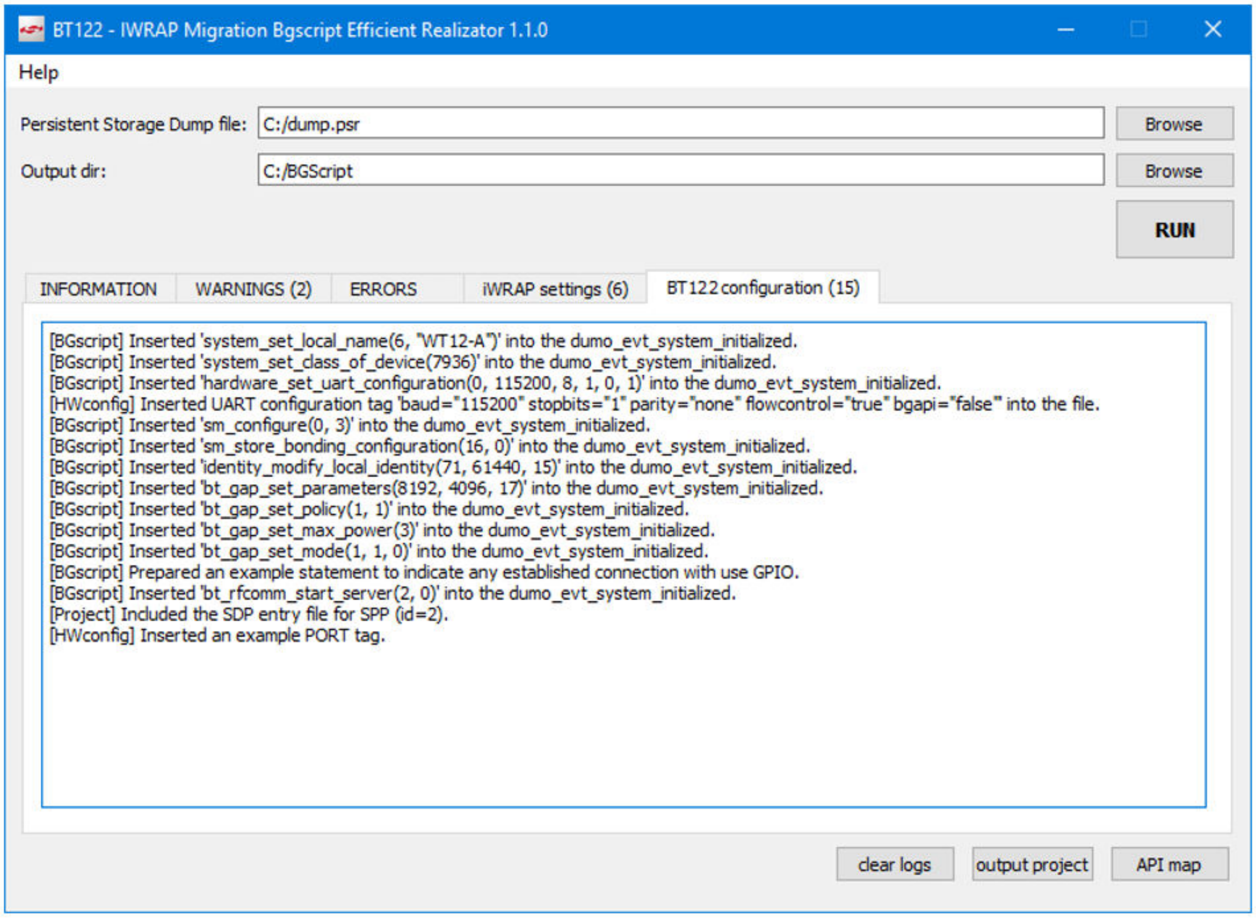
## 5. iWRAP to BT122 Migration Tool

Due to the end of WTXX module application support, driven by iWRAP firmware, Silicon Laboratories proposes the BT122 module as a hardware and firmware replacement. The new devices are dedicated to both new and ongoing projects in the maintenance phase. To help customers migrate from WTXX to the new modules, the “BT122 - IWRAP Migration BGScript Efficient Realizator” application is provided as a helpful developer tool. This program is the first step in transferring the existing iWRAP application to BT122 modules.

The “BT122 - IWRAP Migration BGScript Efficient Realizator” (BIMBER) application helps to run BT122 firmware projects from a sample model project, which has been applicable with WTXX modules (such as WT11-A, WT11-E, WT11I-A, WT11I-E, WT11U-A, WT11U-E, WT12-A, WT32-A, WT32-E, WT32I-A, W32I-E, WT41-A, WT41-E, WT41-N, WT41U-A, WT41U-E, WT41U-N) so far.

The basic concept of migration is retrieving WTXX module configuration from a dump of persistent storage (of this module) and preparing an example BT122 firmware project file as a replacement proposal. WTXX modules are controlled by iWRAP firmware, which is based on the ASCI command console. Some typed commands can change configuration stored in persistent storage. A similar configuration and functionality can be achieved on the BT122 module using API commands and other setup included in the firmware project files. API commands can be sent directly to the module (with API interface) or compiled in the standalone custom script (BGScript). Additionally, the results of the automatic migration process are ready to build the firmware project with BGScript script. The script source code is an example BT122 module application, which demonstrates the use of API commands and interactions with API events. Moreover, after adaptation and a detailed configuration, the module application example can be used as the final application.

More information can be found in a dedicated document: *BT122 IWRAP Migration Guide.*



Persistent Storage Dump file: C:/dump.psr

Output dir: C:/BGScript

```
[BGscript] Inserted 'system_set_local_name(6, "WT12-A")' into the dumo_evt_system_initialized.
[BGscript] Inserted 'system_set_class_of_device(7936)' into the dumo_evt_system_initialized.
[BGscript] Inserted 'hardware_set_uart_configuration(0, 115200, 8, 1, 0, 1)' into the dumo_evt_system_initialized.
[HWconfig] Inserted UART configuration tag 'baud="115200" stopbits="1" parity="none" flowcontrol="true" bgapi="false"' into the file.
[BGscript] Inserted 'sm_configure(0, 3)' into the dumo_evt_system_initialized.
[BGscript] Inserted 'sm_store_bonding_configuration(16, 0)' into the dumo_evt_system_initialized.
[BGscript] Inserted 'identity_modify_local_identity(71, 61440, 15)' into the dumo_evt_system_initialized.
[BGscript] Inserted 'bt_gap_set_parameters(8192, 4096, 17)' into the dumo_evt_system_initialized.
[BGscript] Inserted 'bt_gap_set_policy(1, 1)' into the dumo_evt_system_initialized.
[BGscript] Inserted 'bt_gap_set_max_power(3)' into the dumo_evt_system_initialized.
[BGscript] Inserted 'bt_gap_set_mode(1, 1, 0)' into the dumo_evt_system_initialized.
[BGscript] Prepared an example statement to indicate any established connection with use GPIO.
[BGscript] Inserted 'bt_rfcomm_start_server(2, 0)' into the dumo_evt_system_initialized.
[Project] Included the SDP entry file for SPP (id=2).
[HWconfig] Inserted an example PORT tag.
```

**Figure 5.1. BIMBER Application Main Window View**

# 6. Firmware Update

This section contains a comparison between firmware update methods in three devices: iWRAP, BT121, and BT122.

## 6.1 iWRAP

iWRAP firmware can be updated in three different ways using two different interfaces:

- Firmware update over SPI interface is the fastest and most reliable way of updating the firmware in your Bluetooth module. However, a downside is that you need physical access to the SPI pins of the module and a spatialized converter between the PC and the device because most PCs do not have any SPI port and operating systems do not directly support this protocol. But in some cases, the SPI interface is the only way to update the firmware. To update the module with a SPI interface, two tools can be used:
  - Silicon Labs delivers iWRAP Update Client. This application is a simple Windows software, which allows the user to update the iWRAP Bluetooth module firmware. This method of updating the firmware always restores the factory settings of module. iWRAP 5.0.0 and the newer firmware version require a license key to operate. iWRAP Update Client application can be used to install a license key into the Bluetooth module. The license key needs to match the Bluetooth address of the device.
  - Another tool to update firmware is by using BlueFlash tool from Cambridge Silicon Radio (CSR). It can be used to update the firmware of an iWRAP module in a similar fashion as with the iWRAP Update Client. However, a special separate firmware image file must be used with BlueFlash. The firmware image files delivered with the iWRAP Update Client must not be used with this application because they will corrupt the unique settings of the module.
- The third way to update firmware in iWRAP is by using the DFU protocol. DFU protocol allows you to update an iWRAP module over UART or USB interface. It allows more flexible firmware updates than SPI, especially when the iWRAP module is placed on an application board or connected to the host. Unlike the SPI, the DFU is an open protocol and can therefore be implemented on microcontrollers and processors. This might be useful in case a cabled connection from a PC to the iWRAP module is not possible. In such a case, the host processor can be used to perform the firmware update. To update firmware using DFU, SerialDFU application is available.

## 6.2 BT121

BT121 firmware can be updated in two ways:

The first method uses a BGAPI-based DFU, which allows the MCU in a host system to re-program the BT121 over the UART interface using BGAPI commands. The best way to understand how this method works is to read through the source code of the host example program found under the \host_example\dfu\ directory of the BT122 SDK. The firmware update process can be summarized in a few points, which are presented below:

1. Send `system_reset(1)` to reboot the module into DFU mode.
2. Wait for `dfu_boot event`.
3. Send `dfu_flash_set_address(...)` using address of 0x0.
4. Upload the whole .bin firmware image using `dfu_flash_upload(...)` – split image file into smaller parts.
5. Send `dfu_flash_upload_finish().`
6. Send `dfu_reset(0).`
7. Wait for received events.
   a. `system_boot` event is received. Firmware has been successfully updated and module is ready to use.
   b. `dfu_boot` event is received, which means there was an error during the update. Go back to point 3 and repeat the steps.

The second method of the update is by using the ST DFU protocol. For this method, support is built into the BT121 controller processor. This method can be carried on by using either the `bgupdate.exe` found inside the `\bin\ directory` of the SDK, or via the "Upload tool" section of the BGTool application for Windows (which uses the same code as the `bgupdate.exe`). For a DKBT board, this method is preferred, because the BGTool tools will recognize the Prolific USB-serial chip in the main board and then will use this chip to assert the BOOT0 pin and reset the module into DFU mode. When it keeps the BOOT0 pin high while resetting the module, the ST bootloader will not launch the BT121 software's own bootloader (which implements the BGAPI-based DFU functionality described above) and as a result will not launch the BT121 stack. In that case, the ST bootloader will remain waiting for the host system to start communicating using the application that supports the ST protocol to re-flash the module over the UART.

## 6.3 BT122

The main way to upload the firmware into the BT122 module is to use the BGTool application (available for Windows only). The BGTool application is delivered as a part of the official SDK package. For a better understanding of the process, it is worth getting acquainted with update interfaces offered by the module.

### 6.3.1 Interfaces

BT122 offers two different types of update interfaces: J-Link (SWD) and UART. Those are described below:

**J-link (SWD)**

The main method of uploading the software to the BT122 module is through the J-Link (SWD) interface. This process is effective and the most comprehensive. It allows complete module erasure and upload of the bootloader and firmware to the flash memory. It also allows one to repair a crashed bootloader (for example, when the user would flash, by mistake, a firmware-only variant of a binary file to the bootloader memory range of MCU flash). Proper connection of the J-Link (SWD) is shown in the figure below.

The development boards for BT122, BRD4315A and BRD4315B, have the J-Link (SWD) interface available and already connected as the on-board programmer for BT122.

Additionally, the mini-simplicity connector is also available on the development board.

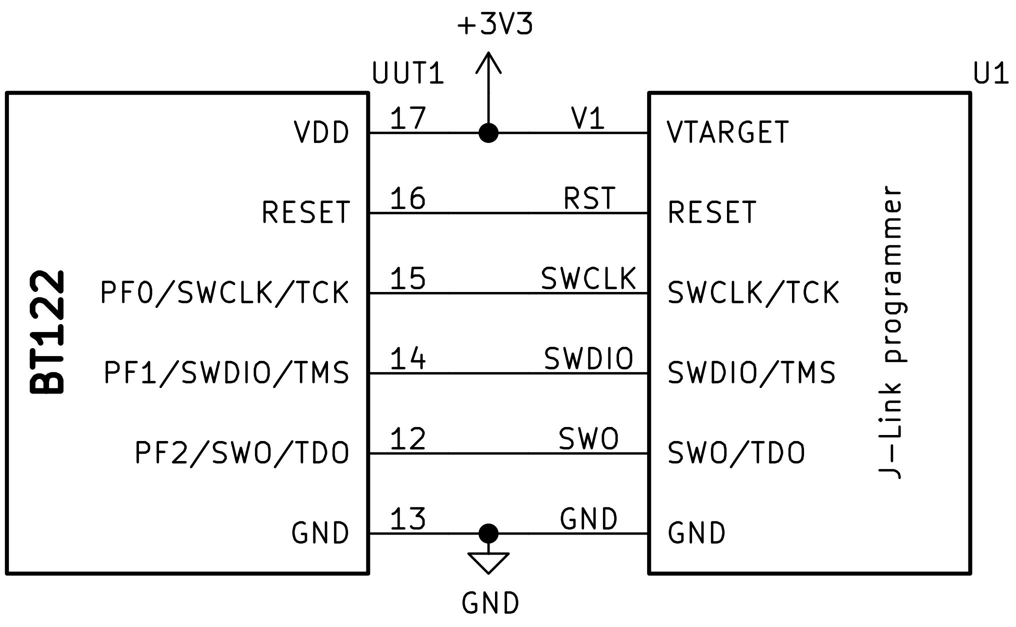


**Figure 6.1. BT122 Firmware Update via J-Link Debugger and SWD Interface**

**UART**

The module can also be flashed by an internal UART bootloader, which uses BGAPI-based DFU protocol. The BT122 firmware can be updated through the UART interface by holding the host MCU in reset state, which typically will free the UART lines to be used by the update interface. This process is similar to the one for BT121, which is described in Section 6.2 BT121. To enable the firmware update state, one must have an external UART interface connection as shown in the figure below. In contrast to the J-Link method, the user needs to make sure that the proper bootloader is flashed into the BT122 module before starting the update procedure.

The development boards for BT122, BRD4315A and BRD4315B, have the UART/USB bridge with an interface called VCOM, which is available on-board and already connected to the BT122 module.

For special applications that require the highest performance of the UART/USB connection, a user can choose to connect the external UART/USB bridge to use with BRD4315A instead of the built-in VCOM interface – for example, CP2102 UART/USB bridge. To do so, follow the steps below:

1. Desolder 0 ohm resistors on RX, TX, CTS, and RTS lines that connect the BT122 module with the on-board debugger - R202, R203, R207, R208.
2. Connect your external bridge lines to corresponding lines available on the left breakout pad.
3. Connect the power supply to your external bridge.

In the case of BRD4315B, CP2102 UART/USB bridge is available on the board and connected to the second USB Micro-B connector.

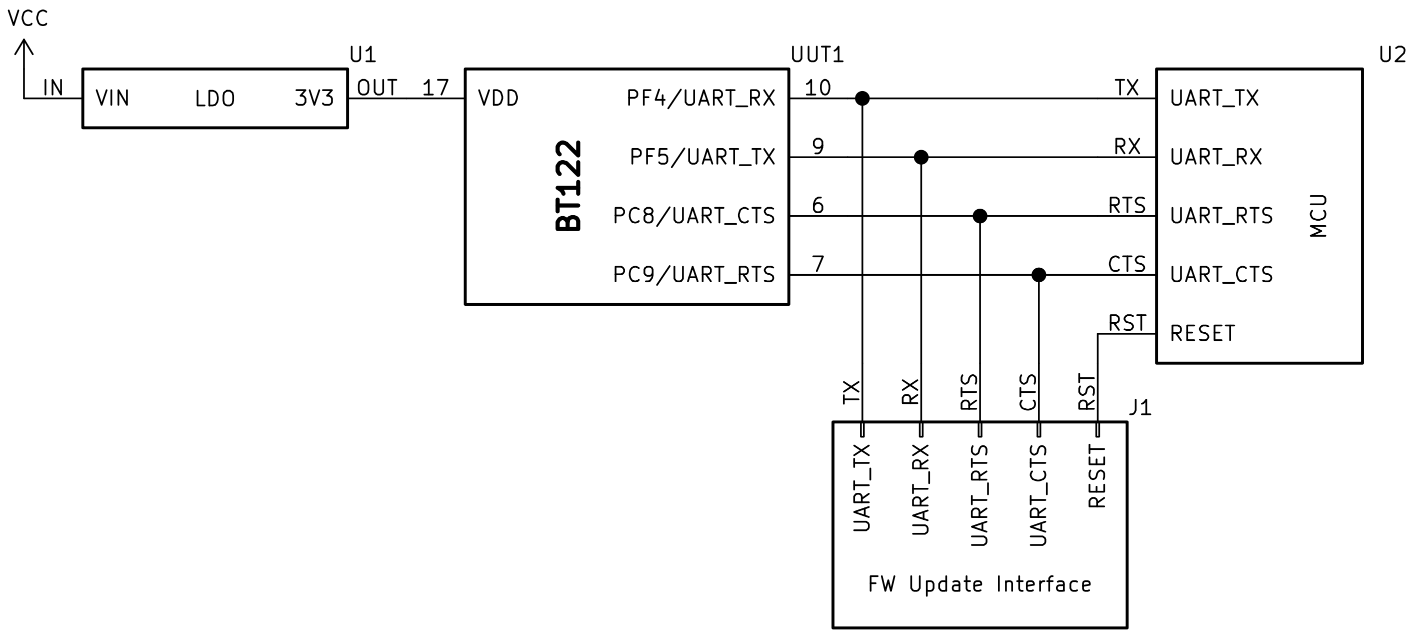


**Figure 6.2. BT122 Firmware Update via UART Connections**

### 6.3.2 BGTool

The BGTool lets users communicate with the module using BGAPI commands or a more user-friendly graphical interface, and to build projects using the BGBuild compiler, and then upload them to the module via the J-Link. The BGTool can be found in the SDK directory after installation (latest SDK package can be found on the module's website). To open the BGTool, go to the **bin** folder and run *bgtool.exe*. The BGTool uses the Simplicity Commander application and BGAPI-based DFU application to execute the update process. Both are included separately in the SDK package to be used by more advanced users outside the BGTool, according to the user's needs.

The default production settings in terms of VCOM configuration are the following: 115200 baud rate and RTS/CTS flow control enabled. If an application that has been prepared/selected to be uploaded uses different UART/USB bridge settings than factory ones and the user uses built-in VCOM, it is necessary to configure the on-board UART/USB bridge settings according to your application setting.

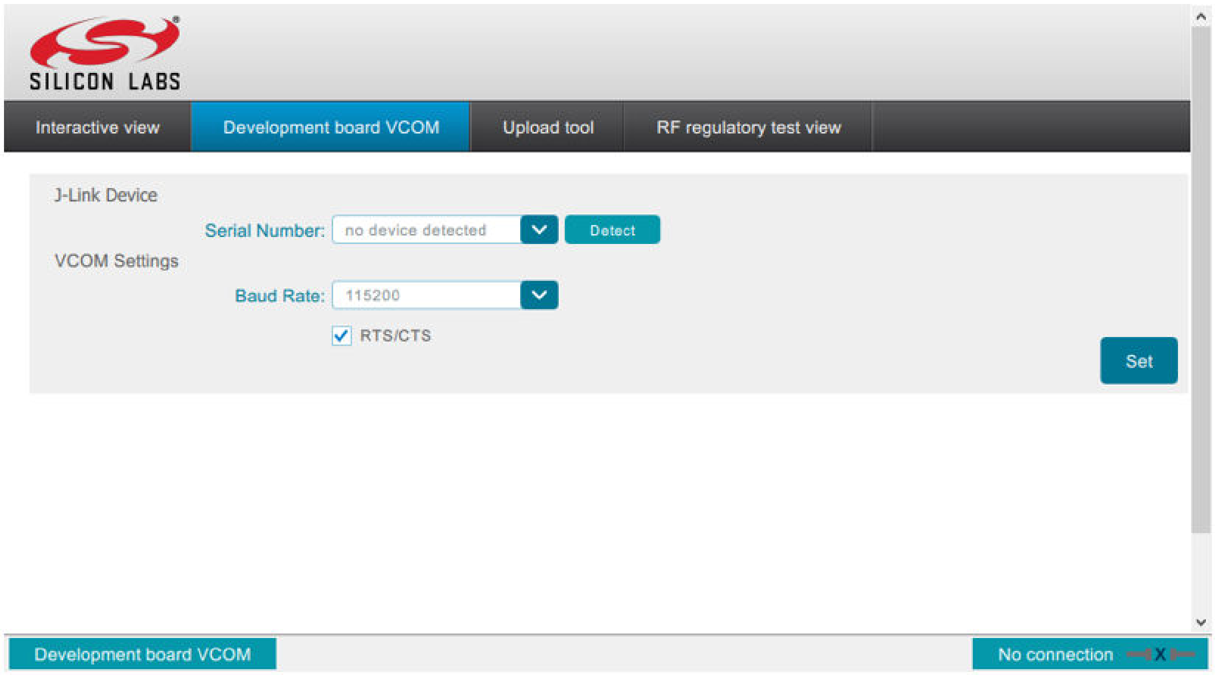
**VCOM Configuration**

The default production settings in terms of VCOM configuration are the following: 115200 baud rate and RTS/CTS flow control enabled. To make sure that connection to the board in the interactive view and usage of all flashing options are going to be possible when VCOM settings are other than factory, it is necessary to configure VCOM options. This can be done under the “Development board VCOM” tab (see the figure below).

**Note:** If you are using an external bridge, you can omit the part about VCOM configuration.

After correctly selecting the Serial Number, baud rate, and RTS/CTS option, press the Set button and if any system prompt arises (such as telnet connection request), accept them. Telnet connection is done locally, is safe, and is a part of the process.

**Note:** Once set, the VCOM settings are permanently stored in memory (until it is changed again).



Serial Number: no device detected

Baud Rate: 115200

✓ RTS/CTS

**Figure 6.3. BGTool in Development Board VCOM Mode**

The firmware does not support configuration over USB-CDC. Nonetheless, it is recommended that the user set the baud rate in the PC-side driver.

Additionally, you can change the VCOM bridge baud rate through CLI as an alternative to the previously mentioned BGTool GUI method:

1. Silink can be found under the SDK package directory …\BT122-x.x.x\bin\silink.
2. Start silink from the CLI with the following command: `“silink -sn <kitserial> -automap 49000”`.
3. Choose telnet localhost 49002.
4. Configure baudrate with `“serial vcom config speed <baudrate> handshake rtscts”`.

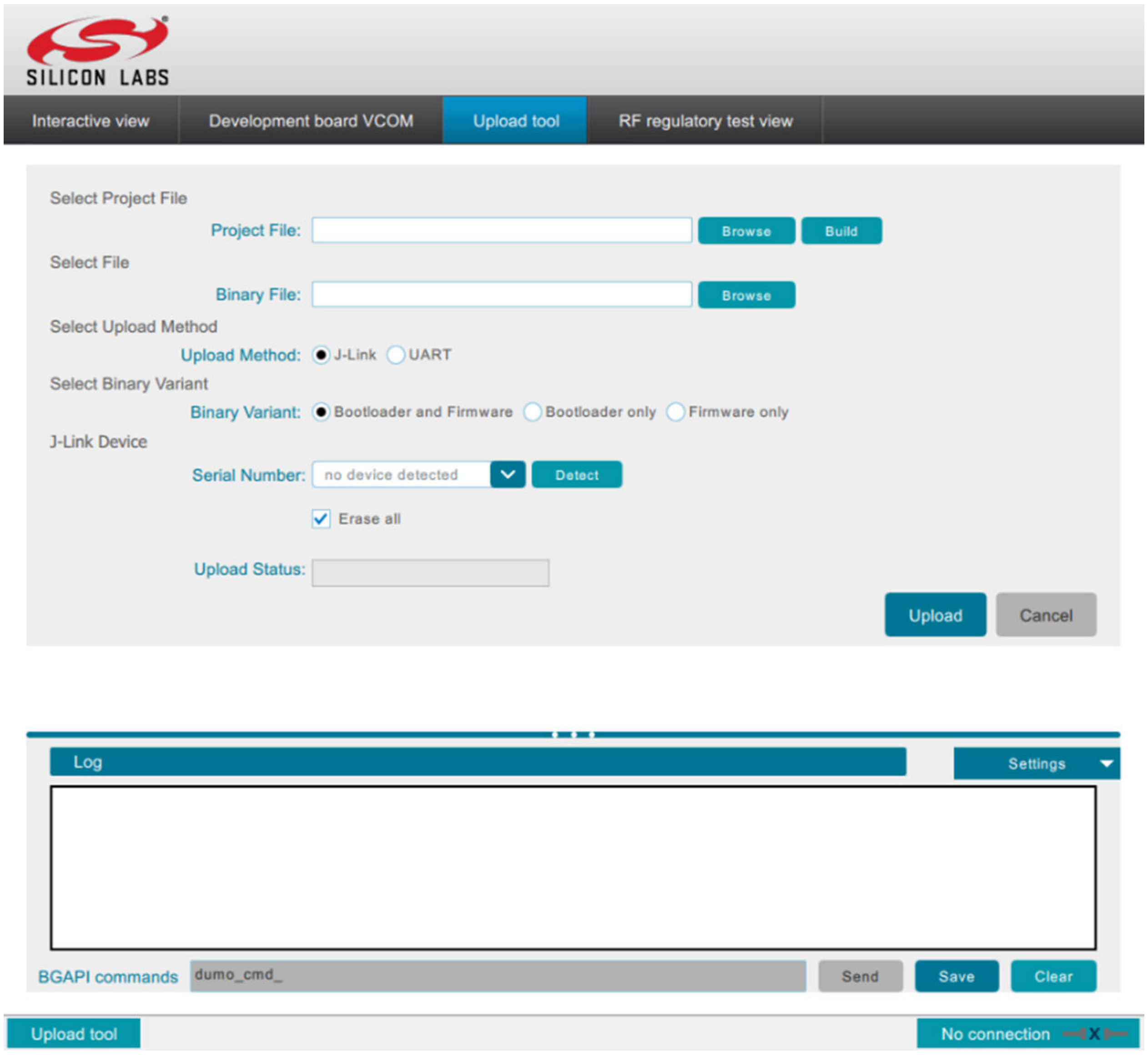
**Uploading FW**

The next step in the process is to go to the update itself. After switching the BGTool program into upload tool view, select the project.xml file of your application project. The project xml file contains links to all configuration files. More information about it can be found in the BT122 Project Configuration Guide. To build the project, call the BGBuild tool by pressing the Build button. BGBuild checks all configuration files, compiles them, and generates firmware binaries.

After this step, three .bin files should be created and placed in the same folder as the project.xml file:

- **Firmware only variant** – consists only of the FW part of the build; suitable if you want to keep the bootloader intact and update only the firmware part of the application.
- **Bootloader only variant** – consists only of the bootloader part of the build.
- **Combined firmware and bootloader variant** – classic approach for complete module firmware update (for example, after full erase)

After the compilation, a Binary File with firmware and bootloader combined will appear automatically in the Binary File field. Keep in mind that the BGTool only supports .bin files.



Select Project File
Project File:
Select File
Binary File:
Select Upload Method
Upload Method: [✓ J-Link] UART
Select Binary Variant
Binary Variant: [✓ Bootloader and Firmware] Bootloader only Firmware only
J-Link Device
Serial Number: no device detected
[✓ Erase all]
Upload Status:
BGAPI commands dumo_cmd_
No connection

**Figure 6.4. BGTool in Upload Tool Mode**

Next, it is necessary to choose an Upload Method – meaning the update interface:

- **J-Link** – selected binary variant will be uploaded through J-Link. This method is faster and can, by selecting the "Erase All" option, perform a "clean" firmware installation. Make sure that you select the correct J-Link device (its serial number in case there is more than one device connected).

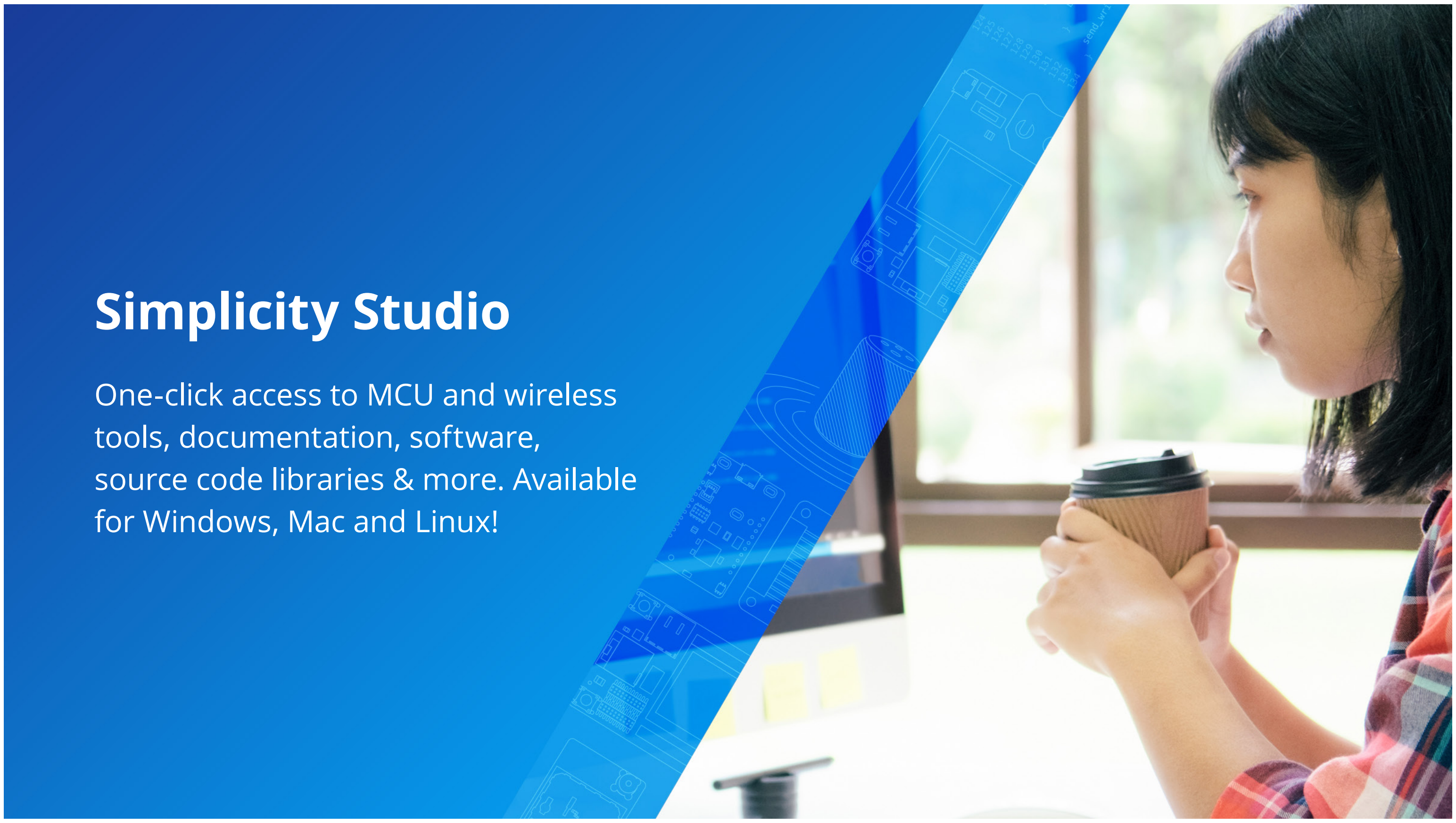
- **UART** – Used when the device is in DFU mode. The selected binary variant will be updated through UART, using selected Port, Baud rate, and flow control. Flow control is selected with the **RTS/CTS** checkbox. Normally, the device must be set in DFU mode by the BGAPI command or in the BGScript, but the BGTool will automatically send the **`dumo_cmd_dfu_reset(1)`** command, putting the device in DFU mode.

After selecting the interface option, specify the variant of the built binary that will be uploaded to the module. It is a very important selection, since this option selects not only the correct file, but also the address range that this file will be written into. It is important to choose a variant that corresponds to the binary file content that you have selected. It is worth mentioning that it is good to keep the names of the files generated by BGBuild, because thanks to them, the BGTool will suggest the correct options (Binary Variant and filename/path) automatically.

At the end of the process, select the Serial Number/Port and appropriate settings – such as erase all (clear whole FLASH) for J-Link and RTS/CTS, baud rate for UART. If all necessary fields are filled, click Upload. After the process is over, the "Done!" message should appear in the Upload Status field. If this does not appear, analyze the logs, and check the steps again.

To connect to the board in the Interactive view and use all flashing options in the Upload tool view, make sure that all UART/USB bridge options are the same for VCOM, uploaded application, and in the Interactive view connection options or DFU upload options.

**Note:** Each UART/USB connection settings change done with API commands must be also done for VCOM settings in the BGTool to keep the ability to connect to the module.


Simplicity Studio
One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

**IoT Portfolio**
www.silabs.com/IoT

**SW/HW**
www.silabs.com/simplicity

**Quality**
www.silabs.com/quality

**Support & Community**
www.silabs.com/community

**Silicon Laboratories Inc.**
**400 West Cesar Chavez**
**Austin, TX 78701**
**USA**

**www.silabs.com**