Telit wireless solutions

Bluetooth Library User Guide

For GE863-PRO3 with Linux APIs description

1VV0300790 Rev. 0 - 23/10/08

Making machines talk.

Disclaimer

The information contained in this document is the proprietary information of Telit Communications S.p.A. and its affiliates (“TELIT”).

The contents are confidential and any disclosure to persons other than the officers, employees, agents or subcontractors of the owner or licensee of this document, without the prior written consent of Telit, is strictly prohibited.

Telit makes every effort to ensure the quality of the information it makes available. Notwithstanding the foregoing, Telit does not make any warranty as to the information contained herein, and does not accept any liability for any injury, loss or damage of any kind incurred by use of or reliance upon the information.

Telit disclaims any and all responsibility for the application of the devices characterized in this document, and notes that the application of the device must comply with the safety standards of the applicable country, and where applicable, with the relevant wiring rules.

Telit reserves the right to make modifications, additions and deletions to this document due to typographical errors, inaccurate information, or improvements to programs and/or equipment at any time and without notice.

Such changes will, nevertheless be incorporated into new editions of this document.

Applicable Products

GE863-PRO3 with Linux OS	GE863PR3***-***

The suffix “***-***” depends on the module HW/SW configuration. Please contact your Telit representative for details

Contents

1 Introduction

The GE863-PRO^3 is an innovation to the quad-band, RoHS compliant GE863 product family which includes a powerful ARM9^{TM} processor core exclusively dedicated to customer applications. The concept of collocating a powerful processor core with the GSM/GPRS engine allows developers to host their application directly. The PRO^3 incorporates much of the necessary hardware for communicating microcontroller solutions, including the critical element of memory, significant simplification of the bill of material, vendor management, and logistics effort are achieved.

1.1 Scope

This user guide serves the following purpose:

- Describes GE863-PRO^3-Bluetooth hardware and software architecture
- Describes how software developers can use the functions of the Bluetooth software package to configure and manage a Bluetooth module.

1.2 Audience

This User Guide is intended for software developers who develop applications on the GE863-PRO^3 module that needs to configure and manage Bluetooth module.

1.3 Contact Information, Support

Our aim is to make this guide as helpful as possible. Keep us informed of your comments and suggestions for improvements.

For general contact, technical support, report documentation errors and to order manuals, contact Telit's Technical Support Center at:

TS-EMEA@telit.com or http://www.telit.com/en/products/technical-support-center/contact.php

Telit appreciates feedback from the users of our information.

1.4 Open Source Licenses

Bluetooth software package is made up of different Open Source Software licensed as follows.

1.4.1 BlueZ

BlueZ is an implementation of the Bluetooth™ wireless standards specifications for Linux. The code is licensed under the GNU General Public License (GPL) and is now included in the Linux 2.6 kernel series.
For further information about GNU License please have a look at http://www.gnu.org/copyleft/gpl.html.

1.5 Document Organization

This manual contains the following chapters:

- “Chapter 1, Introduction” provides a scope for this manual, target audience, technical contact information, and text conventions.
- “Chapter 2, GE863-PRO^3-Bluetooth architecture” describes the general hardware and software architecture for Bluetooth-GE863-PRO^3 system.
- “Chapter 3, Bluetooth module setup” describes how to downloading and installing the needed Bluetooth support modules and bluez and dbus packages.
- “Chapter 4, Commands summary” provides a list and some examples on the most commonly used shell commands for configuring Bluetooth module.
- “Chapter 5, BlueZ Utilities” provides a reference to the most used bluez commands.
- “Chapter 6, Library Setup” gives guidelines to setup your implementation project.
- “Chapter 7, Linux Bluetooth High Level APIs” describes the APIs that can be used by customer applications to configure and manage Bluetooth module from source code.
- "Chapter 8, Appendix" describes Bluetooth headset and sap profile architecture.
- “Chapter 9, List of acronyms and Abbreviation” provides definition for all the acronyms and abbreviations used in this guide.

How to Use

If you are new to this product, it is highly recommended to start by reading through TelitGE863PRO3Linux_Development and TelitGE863PRO3Linux_SW_UserGuide manuals and this document in their entirety in order to understand the concepts and specific features provided by the built in software of the GE863-PRO^3.

1.6 Text Conventions

This section lists the paragraph and font styles used for the various types of information presented in this user guide.

Format	Content
`Courier`	Linux shell commands at command prompt.

1.7 Terminology

In the following sections, the term “host” will refer to the computer where the development environment is running, while we’ll refer to the Pro^3 as the target.
The term “Bluetooth module” will refer to Bluetooth hardware connected to the PRO3. This hardware consists of radio, baseband and the link manager and will be found in Bluetooth chips, dongles and notebooks.
The term “local Bluetooth Adapter” will be used to refer to the Bluetooth module when it is connected to a remote Bluetooth device.

1.8 Related Documents

The following documents are related to this user guide:

[1] Telit_GE863-PRO^3_Hardware_User_Guide
[2] TelitGE863PRO3_EVK_UserGuide
[3] TelitGE863PRO3Linux_SW_UserGuide
[4] TelitGE863PRO3Linux_Development

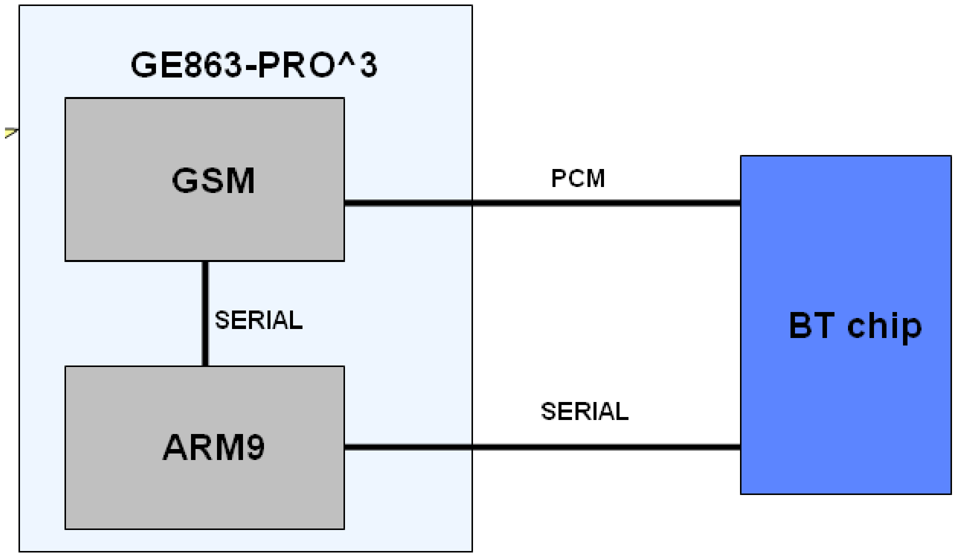
1.9 Document History

Revision	Date	Changes
ISSUE #0	23/10/08	First Release

2 GE863-PRO³-Bluetooth architecture

2.1 Hardware

The BT module is connected and communicates with GE863-PRO³ through an UART interface.
The PCM link connects BT chip to the DVI interface of the GSM module of GE863-PRO^3 to transport the PCM audio data for the HS profile.
For further hardware information please refer to [1] , .



2.2 Software

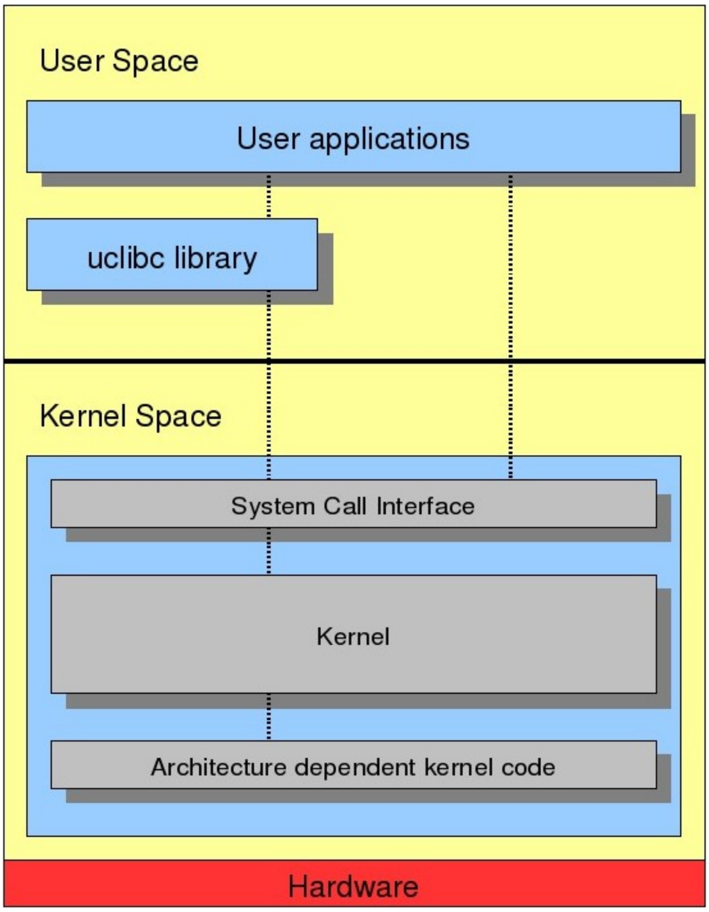
Below a high level description of Linux OS Architecture and the different software layers involved in the Bluetooth package to better understand how local Bluetooth Adapter can be configured and controlled by Telit GE863-PRO³.

2.2.1 Linux OS overview

The kernel is the central part of the GNU/Linux operating system: its main task is to manage system's resources in order to make the hardware and the software to communicate. A kernel usually deals with process management (including inter-process communication), memory management and device management.

The Linux kernel belongs to the family of Unix-like operating system kernel; created in 1991, it has been developed in the years by a huge number of contributors worldwide, becoming one of the most common and versatile kernel for embedded systems.

Below there is a picture representing, from a high level perspective, the architecture of a GNU/Linux operating system.



Two regions can be identified:

1) User space: where the user applications are executed.
2) Kernel space: where the kernel (with all its components such as device drivers) works.

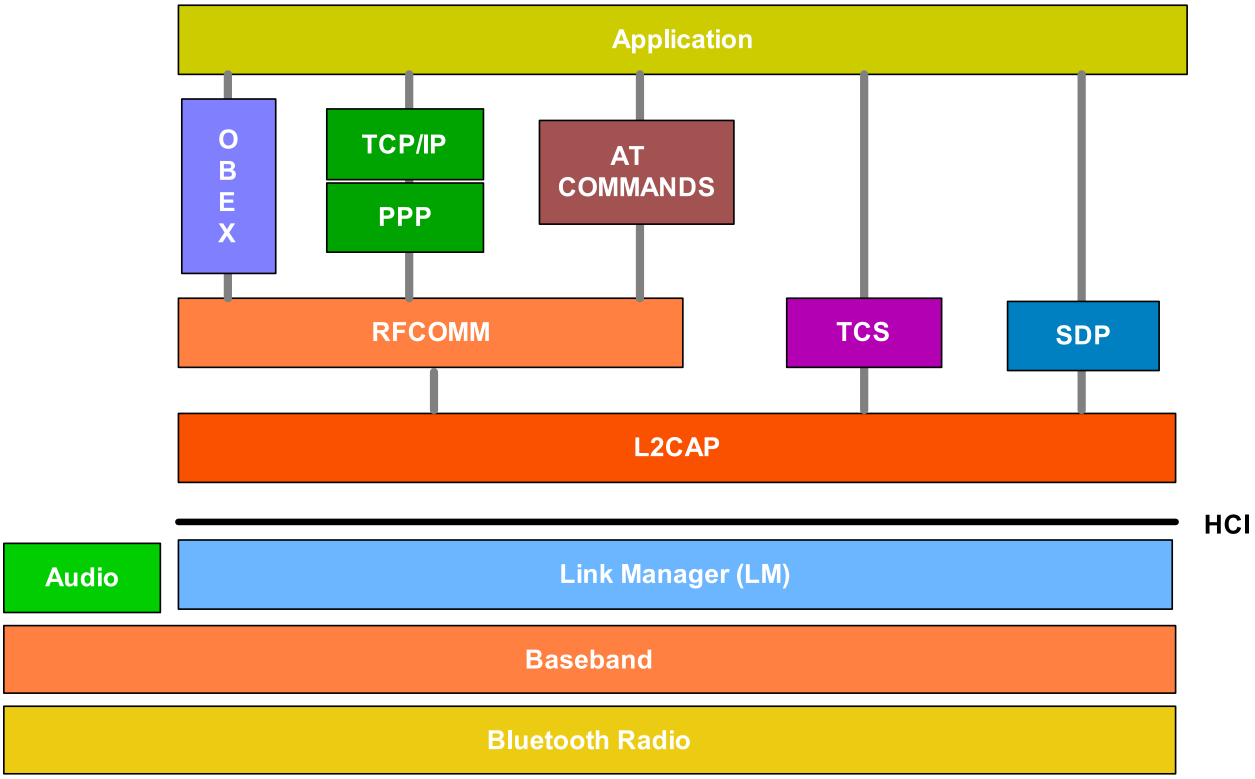
These two regions are separated and have different memory address spaces; there are several methods for user/kernel interaction:

- Using the System Call Interface that connects to the kernel and provides the mechanism to communicate between the user-space application and the kernel through the C library.
- Using kernel calls directly from application code leaping over the C library.
- Using the virtual filesystem /proc.

The ordinary C library in Linux system is the glibc. Uclibc is a C library mainly targeted for developing embedded Linux systems; despite being much smaller than the glibc it almost has all its features (including shared libraries and threading), making easy to port applications from glibc to uclibc.
The Linux kernel architecture-independent code stays on the top of platform specific code for the GE863-PRO^3 board: this code allows exploiting all the hardware features of the GE863-PRO^3.

2.2.2 Bluetooth Overview

Bluetooth is a wireless protocol utilizing short-range communications technology facilitating both voice and data transmissions over short distances from fixed and/or mobile devices.
Bluetooth provides a way to connect and exchange information between devices such as mobile phones, telephones, laptops, personal computers, printers, GPS receivers, digital cameras, and video game consoles over a secure, globally unlicensed Industrial, Scientific, and Medical (ISM) 2.4 GHz short-range radio frequency bandwidth.
The specification is developed, published and promoted by the Bluetooth Special Interest Group (SIG).



In order to use Bluetooth wireless technology, a device must be able to interpret certain Bluetooth profiles. The Profiles describe how the technology is used (i.e. how different parts of the specification can be used to fulfill a desired function for a Bluetooth device). Bluetooth profiles are general behaviors through which Bluetooth enabled devices communicate with other devices. Bluetooth technology defines a wide range of profiles that describe many different types of use cases.

2.2.2.1 Sim Access Profile (SAP)

The SIM Access Profile defines the protocols and procedures that shall be used to access a GSM SIM card via a Bluetooth link.
For example, this profile allows devices such as car phones with built in GSM transceivers to connect to a SIM card in a *Bluetooth* enabled phone. Therefore the car phone itself does not require a separate SIM card.
Furthermore the user can personalize his car embedded phone with data (like contacts, messages and so on) contained in his personal SIM card.

In order to ensure secure communication between Bluetooth devices, several security measures are mandatory.

2.2.2.2 Headset Profile (HSP)

The Headset Profile provides support for the popular Bluetooth Headsets to be used with mobile phones.
With this profile the headset can be wirelessly connected for the purposes of acting as the device's audio input and output mechanism, providing full duplex audio. The headset increases the user's mobility while maintaining call privacy.

2.2.3 Linux Bluetooth software framework

Linux Bluetooth package is made up of different components:

- BlueZ – official Linux Bluetooth protocol stack
- BT_lib - Bluetooth High Level APIs

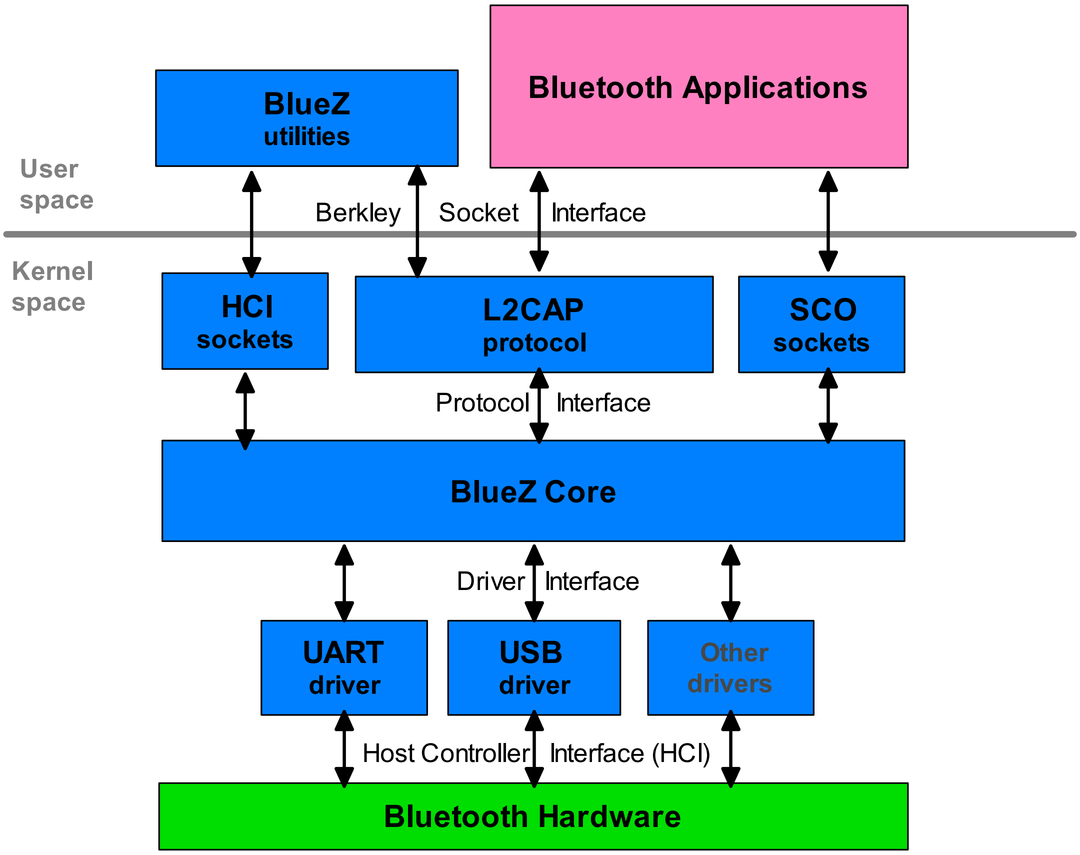
BlueZ provides kernel modules, libraries and utilities. It is composed by the following components:

- UART driver – Implements the UART transport layer
- BlueZ Core Layer – Provides a standard interface to the Bluetooth baseband controller and link manager services (host controller interface)
- L2CAP, SCO, RFCOMM - Bluetooth lower layers
- BlueZ Utilities – Provides some tools which simplify management of Bluetooth module

Bluetooth module is controlled via the Host Controller Interface (HCI) and for the communication between the host stack and the Bluetooth module a specific host transport driver is used.

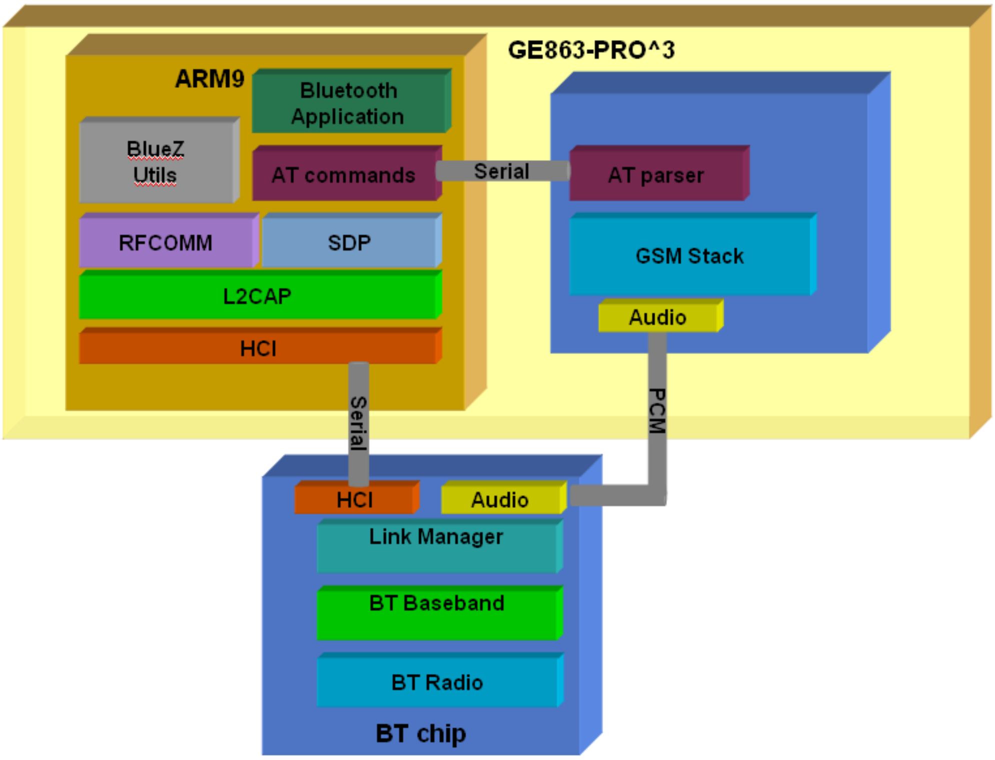
BlueZ implements HCI and Bluetooth lower layer (like L2CAP, SCO, RFCOMM) inside the kernel. Upper Bluetooth protocol layers are implemented as libraries (like Service Discovery Protocol – SDP) and made available as shell command or service exported through the system message bus.

The image below shows architecture of the software framework used to configure and control Bluetooth module.



🟦 **- BlueZ components.**

The following scheme represents the SW architecture of the system GE863-PRO^3 together with the BT module:



3 Bluetooth module setup

3.1.1 Bluetooth Package Downloading

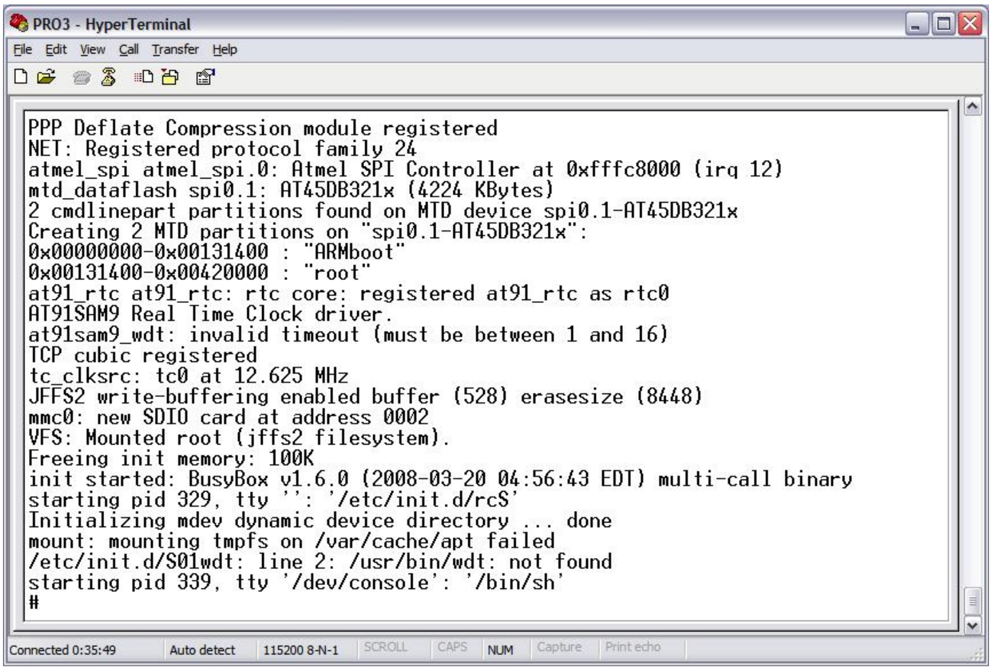
Before setting up Bluetooth module, the components of the Bluetooth Package must be downloaded onto GE863-PRO³ filesystem.

If you don’t have Bluetooth Package yet please contact our technical assistance on the following email: TS-EMEA@telit.com.

Connect the GE863-PRO³ to your host system via serial cable (use Debug port of the EVK, for further details refer to document [2]). Open a terminal program (such as Hyperterminal or Procomm) on your host system and use for the connection the following parameters:

Bits per second: 115200
Data bits: 8
Parity: None
Stop bits: 1
Flow Control: None

Turn GE863-PRO³ on. Once the system startup has finished, the terminal will display the shell prompt as shown below.



```
PPP Deflate Compression module registered
NET: Registered protocol family 24
atmel_spi atmel_spi.0: Atmel SPI Controller at 0xfffc8000 (irq 12)
mtd_dataflash spi0.1: AT45DB321x (4224 KBytes)
2 cmdlinepart partitions found on MTD device spi0.1-AT45DB321x
Creating 2 MTD partitions on "spi0.1-AT45DB321x":
0x00000000-0x00131400 : "ARMboot"
0x00131400-0x00420000 : "root"
at91_rtc at91_rtc: rtc core: registered at91_rtc as rtc0
AT91SAM9 Real Time Clock driver.
at91sam9_wdt: invalid timeout (must be between 1 and 16)
TCP cubic registered
tc_clksrc: tc0 at 12.625 MHz
JFFS2 write-buffering enabled buffer (528) erasesize (8448)
mmc0: new SDIO card at address 0002
VFS: Mounted root (jffs2 filesystem).
Freeing init memory: 100K
init started: BusyBox v1.6.0 (2008-03-20 04:56:43 EDT) multi-call binary
starting pid 329, tty '': '/etc/init.d/rcS'
Initializing mdev dynamic device directory ... done
mount: mounting tmpfs on /var/cache/apt failed
/etc/init.d/S01wdt: line 2: /usr/bin/wdt: not found
starting pid 339, tty '/dev/console': '/bin/sh'
#
```

Start Colinux and make sure the Ethernet on USB connection via USB port is correctly configured as shown in paragraph 5 of [4] .

Now start Eclipse and download the following files onto GE863-PRO³ filesystem as shown in paragraph 7.2 of [4] :

- UART Driver for Bluetooth module:
    - hci_uart.ko

- BlueZ Core Layer:
    - bluetooth.ko

- Bluetooth lower layers:
    - l2cap.ko
    - rfcomm.ko
    - sco.ko

- BlueZ Utilities:
    - bluez/

- Bluetooth High level API:
    - BT_lib/

In order to install the Bluetooth modules you have to execute the following steps:

Now create the kernel modules folders:

```
# mkdir /lib/modules/2.6.24-rc5-rt1/kernel/net
# mkdir /lib/modules/2.6.24-rc5-rt1/kernel/net/bluetooth
# mkdir /lib/modules/2.6.24-rc5-rt1/kernel/net/bluetooth/rfcomm
# mkdir /lib/modules/2.6.24-rc5-rt1/kernel/drivers/bluetooth
```

Now move the downloaded files from the download folder to the correct destination folder.
Supposing you are into the download folder, type:

- For UART Driver:

```
# mv hci_uart.ko /lib/modules/2.6.24-rc5-rt1/kernel/drivers/bluetooth
```

- For BlueZ Core Layer:

```
# mv bluetooth.ko /lib/modules/2.6.24-rc5-rt1/kernel/net/bluetooth
```

- For Bluetooth lower layers:

```
# mv l2cap.ko /lib/modules/2.6.24-rc5-rt1/kernel/net/bluetooth
# mv sco.ko /lib/modules/2.6.24-rc5-rt1/kernel/net/bluetooth
# mv rfcomm.ko /lib/modules/2.6.24-rc5-rt1/kernel/net/bluetooth/rfcomm
```

- For BlueZ Utilities :

```
# mv bluez/lib/libexpat.so.1 /lib
# mv bluez/lib/libdbus-1.so.3 /lib
# mv bluez/lib/libbluetooth.so.2 /lib
# mv bluez/ROOT_DIR_FILES/* /etc
# rm –r bluez/ROOT_DIR_FILES
# mv bluez /home
# mkdir /var/lib/dbus
# mkdir /var/run/dbus
# mv bluez /home
```

(Please, make sure all bluez utilities have the right execution privileges.)

- Update PATH environment variable:

```
# export PATH=$PATH:./:/home/bluez/bin:/home/bluez/sbin
```

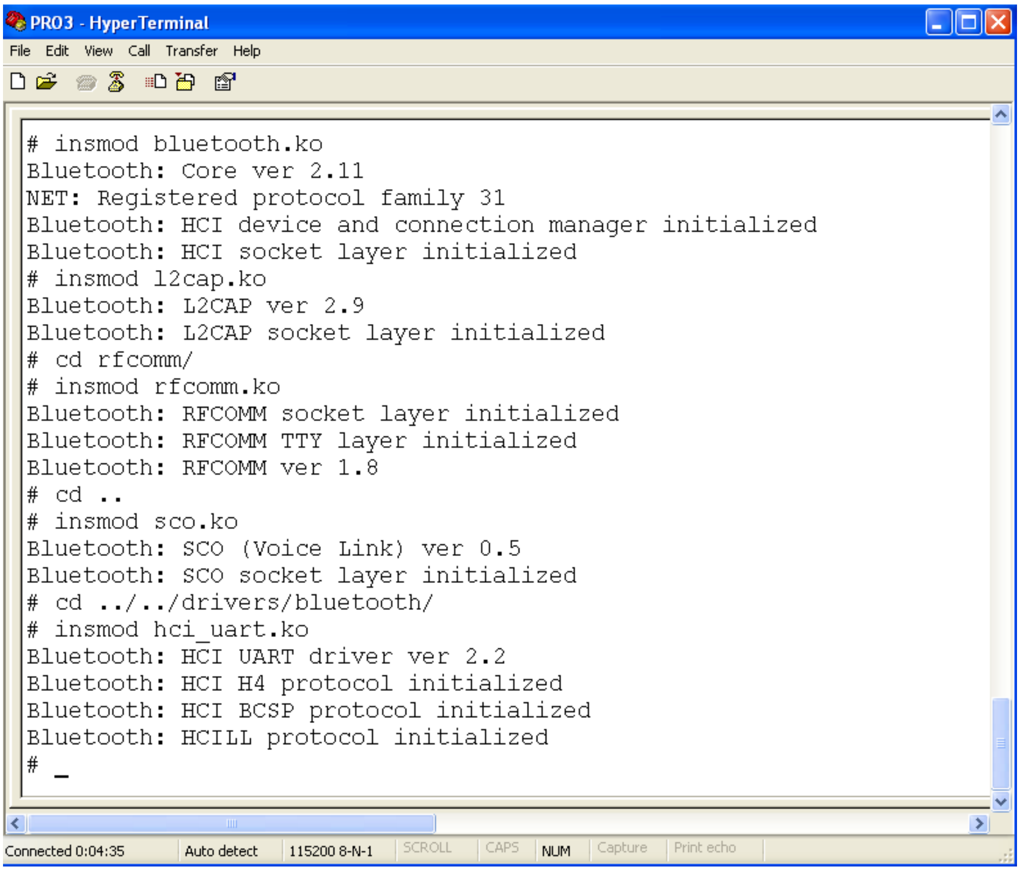
Finally add the messagebus user to the user group:

```
# adduser -SDH messagebus
```

3.1.2 Loading the kernel modules

Go to kernel/ folder and load the kernel modules as shown below:

```
# insmod /lib/modules/2.6.24-rc5-rt1/kernel/net/Bluetooth/bluetooth.ko
# insmod /lib/modules/2.6.24-rc5-rt1/kernel/net/Bluetooth/l2cap.ko
# insmod /lib/modules/2.6.24-rc5-rt1/kernel/net/Bluetooth/rfcomm/rfcomm.ko
# insmod /lib/modules/2.6.24-rc5-rt1/kernel/net/Bluetooth/sco.ko
# insmod /lib/modules/2.6.24-rc5-rt1/kernel//drivers/Bluetooth/hci_uart.ko
```


PRO3 - HyperTerminal
File Edit View Call Transfer Help
Connected 0:04:35 Auto detect 115200 8-N-1 SCROLL CAPS NUM Capture Print echo

```
# insmod bluetooth.ko
Bluetooth: Core ver 2.11
NET: Registered protocol family 31
Bluetooth: HCI device and connection manager initialized
Bluetooth: HCI socket layer initialized
# insmod l2cap.ko
Bluetooth: L2CAP ver 2.9
Bluetooth: L2CAP socket layer initialized
# cd rfcomm/
# insmod rfcomm.ko
Bluetooth: RFCOMM socket layer initialized
Bluetooth: RFCOMM TTY layer initialized
Bluetooth: RFCOMM ver 1.8
# cd ..
# insmod sco.ko
Bluetooth: SCO (Voice Link) ver 0.5
Bluetooth: SCO socket layer initialized
# cd ../../drivers/bluetooth/
# insmod hci_uart.ko
Bluetooth: HCI UART driver ver 2.2
Bluetooth: HCI H4 protocol initialized
Bluetooth: HCI BCSP protocol initialized
Bluetooth: HCILL protocol initialized
# _
```

Once the kernel modules has been successfully loaded, Bluetooth module can be attached to the system and upper layers of the Bluetooth protocol stack can be started.

3.1.3 Attach Bluetooth module via UART HCI

After loading kernel modules is possible to attach one (or more) Bluetooth module to the system. For example, if a CSR chip is used, type the following commands to start the Bluetooth module.

```
#hciattach -pt 10 ttyS1 csr 115200 flow
#hciconfig hci0 up
```

According with the sintax of the hciattach utility (refer to paragraph 5.1.1) changing the “type” option different Bluetooth modules can be used.
Example: in order to use an Ericsson based module the following command lines should be used:

```
#hciattach -pt 10 ttyS1 ericsson 115200 flow
#hciconfig hci0 up
```

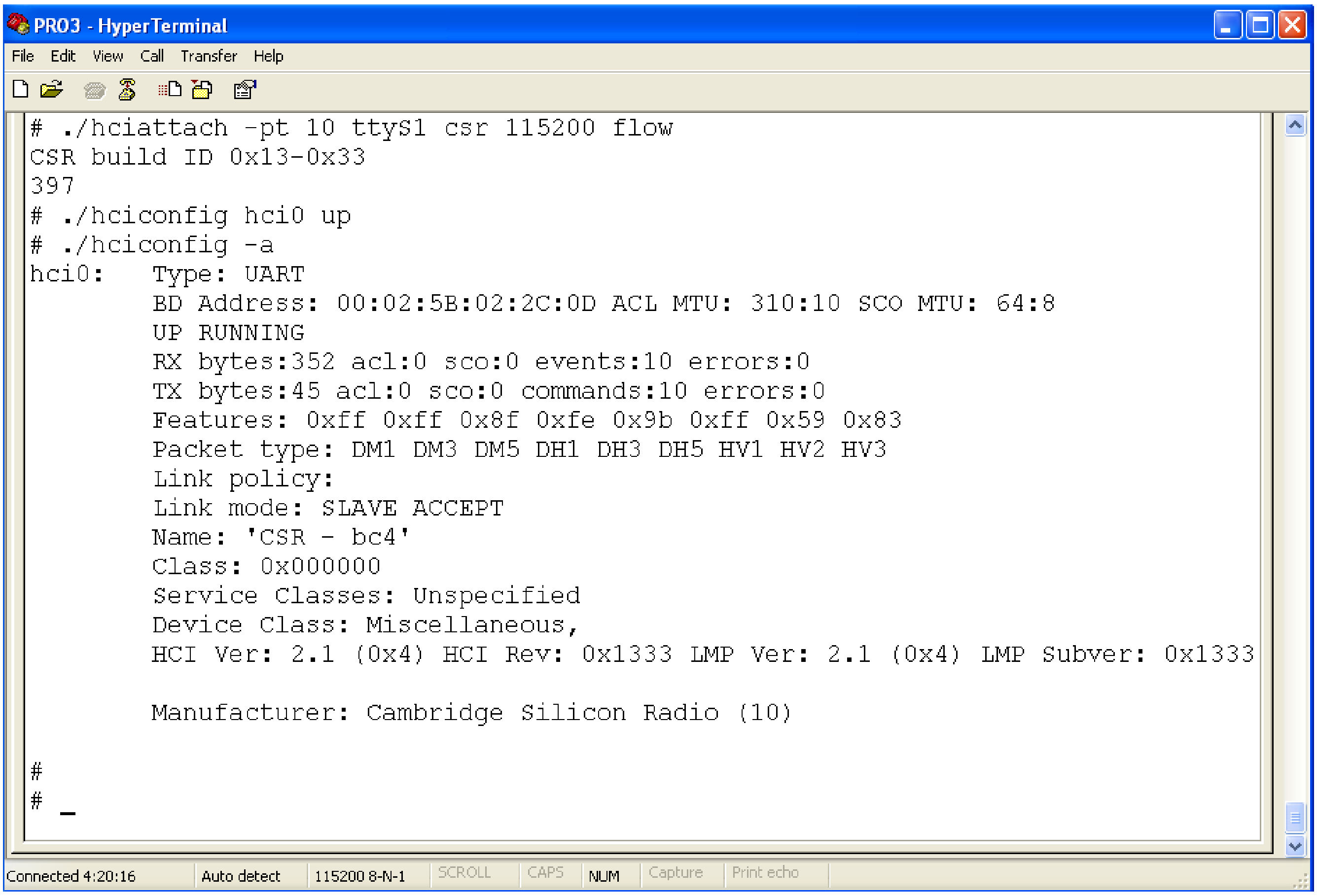
Else, for unknown Bluetooth module vendor that supports HCI UART interface, type:

```
#hciattach -pt 10 ttyS1 any 115200 flow
#hciconfig hci0 up
```

Pay attention to set the additional parameters of the "hciattach" (like speed, control flow etc…) in accord with the desired Bluetooth module.

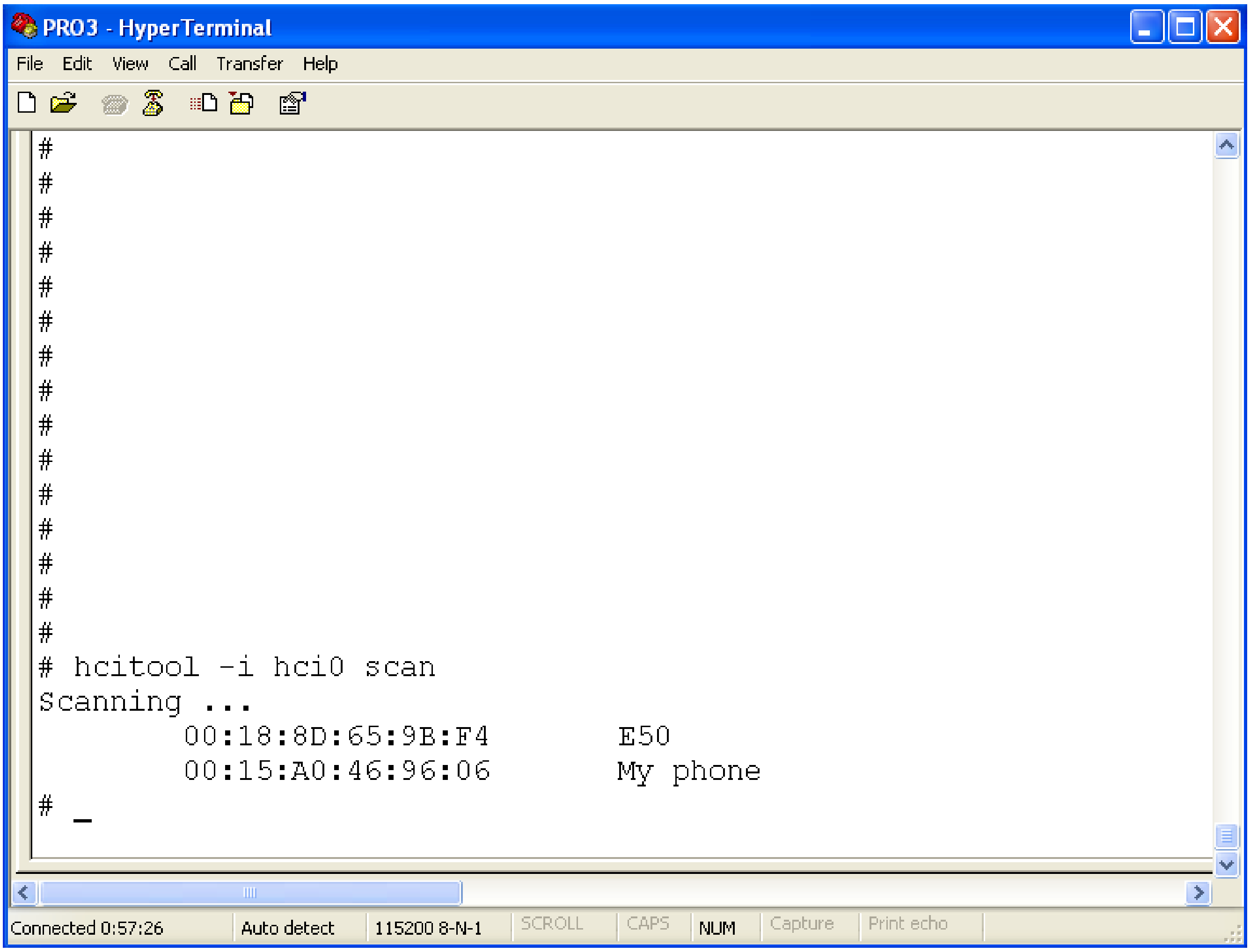
Once started the Bluetooth module, to check its status, type:

```
#hciconfig -a
```



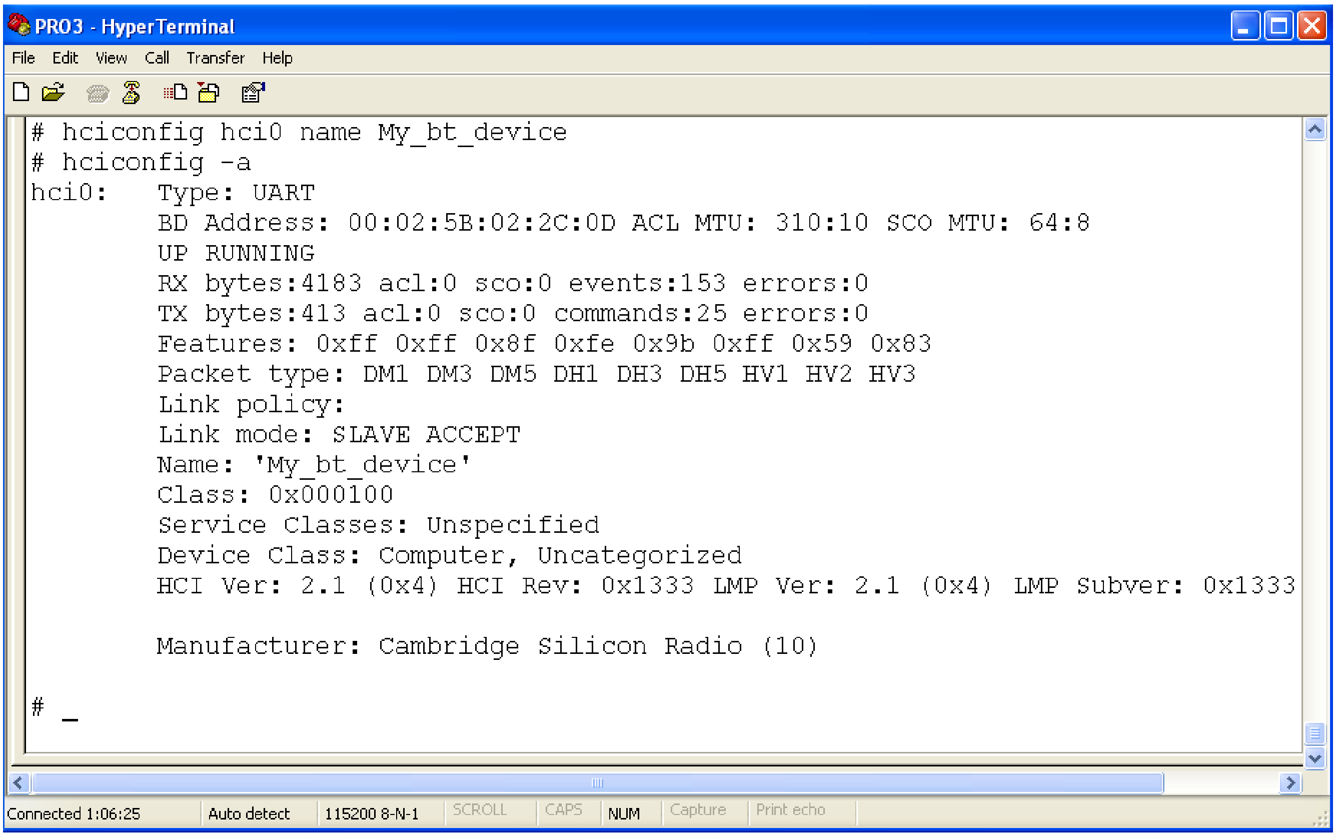
```
# ./hciattach -pt 10 ttyS1 csr 115200 flow
CSR build ID 0x13-0x33
397
# ./hciconfig hci0 up
# ./hciconfig -a
hci0:   Type: UART
        BD Address: 00:02:5B:02:2C:0D ACL MTU: 310:10 SCO MTU: 64:8
        UP RUNNING
        RX bytes:352 acl:0 sco:0 events:10 errors:0
        TX bytes:45 acl:0 sco:0 commands:10 errors:0
        Features: 0xff 0xff 0x8f 0xfe 0x9b 0xff 0x59 0x83
        Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
        Link policy:
        Link mode: SLAVE ACCEPT
        Name: 'CSR - bc4'
        Class: 0x000000
        Service Classes: Unspecified
        Device Class: Miscellaneous,
        HCI Ver: 2.1 (0x4) HCI Rev: 0x1333 LMP Ver: 2.1 (0x4) LMP Subver: 0x1333

        Manufacturer: Cambridge Silicon Radio (10)

#
# _
```

Once Bluetooth module is attached to the system, basic Bluetooth features are available using BlueZ Utilities from the shell.
For example it is possible to perform a scan for remote Bluetooth devices in range.



```
#
#
#
#
#
#
#
#
#
#
#
#
#
#
#
# hcitool -i hci0 scan
Scanning ...
        00:18:8D:65:9B:F4       E50
        00:15:A0:46:96:06       My phone
# _
```

It is also possible to set some parameter of the Bluetooth module.
The following example shows how to set the friendly-name of the local device.



```
# hciconfig hci0 name My_bt_device
# hciconfig -a
hci0:   Type: UART
        BD Address: 00:02:5B:02:2C:0D ACL MTU: 310:10 SCO MTU: 64:8
        UP RUNNING
        RX bytes:4183 acl:0 sco:0 events:153 errors:0
        TX bytes:413 acl:0 sco:0 commands:25 errors:0
        Features: 0xff 0xff 0x8f 0xfe 0x9b 0xff 0x59 0x83
        Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
        Link policy:
        Link mode: SLAVE ACCEPT
        Name: 'My_bt_device'
        Class: 0x000100
        Service Classes: Unspecified
        Device Class: Computer, Uncategorized
        HCI Ver: 2.1 (0x4) HCI Rev: 0x1333 LMP Ver: 2.1 (0x4) LMP Subver: 0x1333

        Manufacturer: Cambridge Silicon Radio (10)

# _
```

To make also available Bluetooth High Level APIs, the upper layer of the Bluetooth protocol stack should be started.

3.1.4 Attach Bluetooth module via USB interface

The “hci_uart.ko” kernel module is a driver for Bluetooth modules which implement the HCI via UART interface. In order to use a Bluetooth module, which implements the HCI via USB interface, the “hci_usb.ko” kernel module should be used instead of “hci_uart.ko”.
Moreover, the USB support must be properly installed and running on the PRO3 (“usb-core” and “ohci_hcd” kernel modules should be loaded).

Bluetooth USB devices get initialized automatically when they are plugged in; if it does not happen, they can be brought up manually with the hciconfig command :

```
#hciconfig hci0 up
```

3.1.4.1 Drivers for some specific Bluetooth modules

BlueZ supports a wide range of Bluetooth devices through the Host Control Interface (HCI). This interface provides a uniform method of accessing the Bluetooth module capabilities.
In order to be compatible with BlueZ, a bluetooth module must export the HCI .

The HCI can be exposed through different physical bus which are supported by Linux using a specific driver for each one. These drivers are included in the Linux kernel sources.
Once compiled with the Bluetooth support, these drivers can be found under the “lib/modules/kernel_version/kernel/drivers/bluetooth/” directory of the kernel binaries.

Example:

- hci_uart : Bluetooth HCI UART driver.
- hci_usb.ko : HCI USB driver.
- hci_vhci.ko : Bluetooth virtual HCI driver.
- bcm203x.ko : Broadcom Blutonium firmware driver.
- bfusb.ko : AVM BlueFRITZ! USB driver.
- bpa10x.ko : Digianswer Bluetooth USB driver.
- btsdio.ko : Generic Bluetooth SDIO driver.

Choose the driver in accord with the desired Bluetooth module.

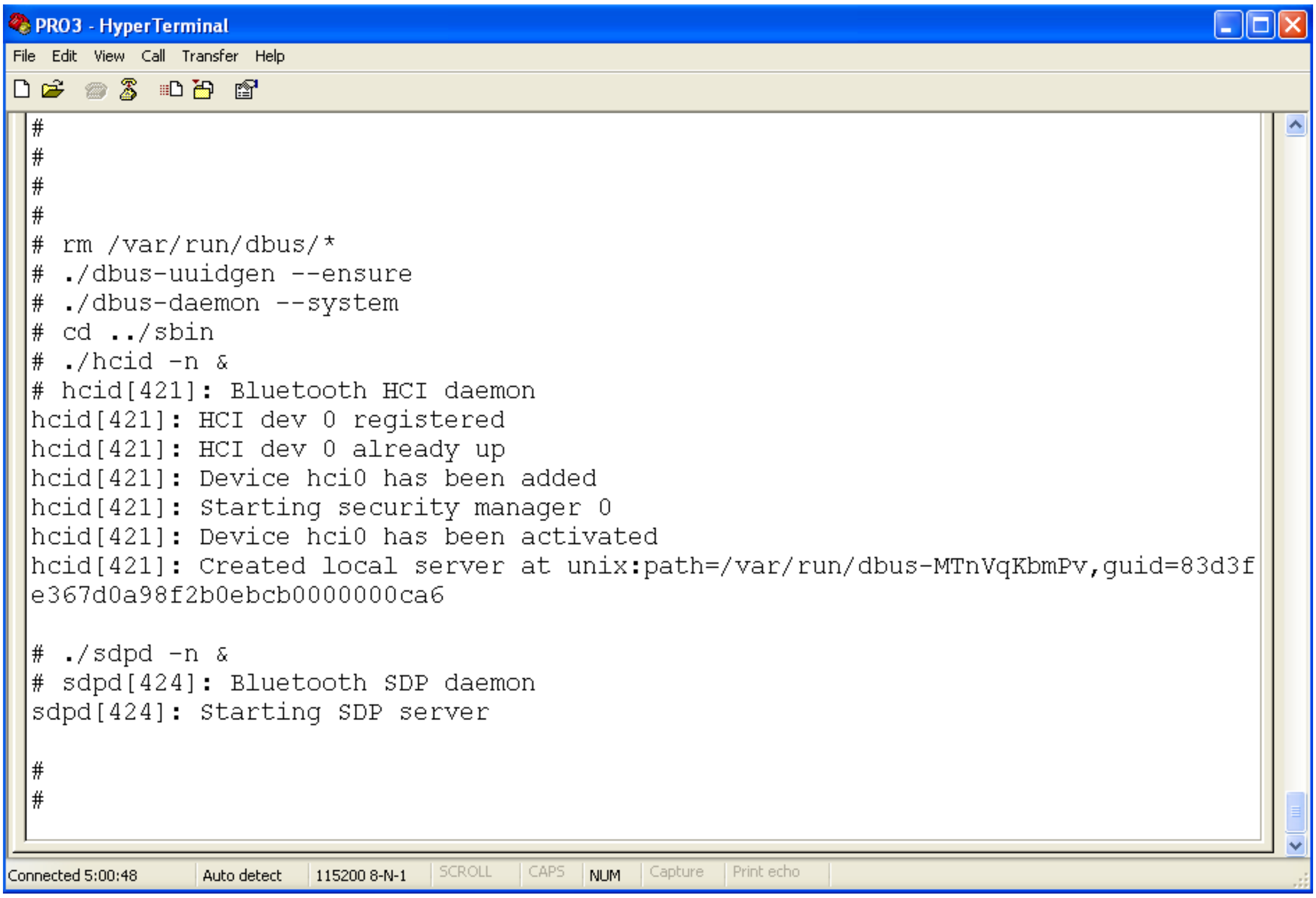
3.1.5 Starting Bluetooth Upper Layers

Before starting Bluetooth Upper Layers, the D-Bus application must be in running. If not yet started type the following commands:

```
# rm /var/run/dbus/*
#dbus-uuidgen --ensure
#dbus-daemon --system
```

Once the D-Bus has been actived it is possible to start the Bluetooth daemons:

```
#hcid -n &
#sdpd -n &
```



```
#
#
#
#
# rm /var/run/dbus/*
# ./dbus-uuidgen --ensure
# ./dbus-daemon --system
# cd ../sbin
# ./hcid -n &
# hcid[421]: Bluetooth HCI daemon
hcid[421]: HCI dev 0 registered
hcid[421]: HCI dev 0 already up
hcid[421]: Device hci0 has been added
hcid[421]: Starting security manager 0
hcid[421]: Device hci0 has been activated
hcid[421]: Created local server at unix:path=/var/run/dbus-MTnVqKbmPv,guid=83d3f
e367d0a98f2b0ebcb0000000ca6

# ./sdpd -n &
# sdpd[424]: Bluetooth SDP daemon
sdpd[424]: Starting SDP server

#
#
```

Now Bluetooth High Level APIs are available. Using them, the BLUETOOTH MODULE can be controlled from source.

3.1.6 Auto-Setup at system startup

The shell script “BT_Bluetooth_Start.sh” (provided by Telit) will perform all the necessary steps in order to initialize the Bluetooth module and run BlueZ correctly.
This script refers to a configuration with only a CSR Bluetooth module connected through a UART interface. If a different Bluetooth device should be used, change the line:

```
hciattach -pt 10 /dev/ttyS1 csr 115200 flow
```

in accord with the desired Bluetooth module (as explained in the paragraphs 3.1.3 , 3.1.4 and 3.1.4.1).

If the upper layers of BlueZ are not necessaries, the following instructions can be used instead of the “BT_Bluetooth_Start.sh” script :

```
hciattach -pt 10 /dev/ttyS1 csr 115200 flow
hciconfig hci0 up
```

Once completed Bluetooth operations, the shell script “BT_Bluetooth_Stop.sh” (provided by Telit) can be used in order to stop the BlueZ framework in the right way.

Refer to paragraph 7.1.3 for more details about these scripts.

4 Commands summary

There are mainly two possible ways to configure and control the Bluetooth module: shell commands (BlueZ Utilities) and source code using Bluetooth High Level APIs.
BlueZ Utilities (hciconfig, hcitool, sdptool, rfcomm) provide simple Linux shell commands that can be used to set some specific parameters of the Bluetooth module and perform basic Bluetooth functions.
Customer applications using Bluetooth High Level APIs, can perform almost the same operations plus some advanced and more complex tasks.

The table below shows examples of the most commonly used shell commands and Bluetooth High Level APIs.

Functionality		Shell Commands (BlueZ Utilities)	High Level APIs
	Enable page and inquiry scan (discoverable)	hciconfig hci0 piscan	
	Disable page and inquiry scan (not discoverable)	hciconfig hci0 noscan	
	Set local name to My_name	hciconfig hci0 name My_name	BT_Set_Local_Name()
	Get local name	hciconfig hci0 name	BT_Get_Local_Name()
Bluez Bluetooth Daemons Manag.	*Start bluez daemons*	dbus-daemon hcid sdpd hciattach	BT_Bluetooth_Start()
	Stop bluez daemons		BT_Bluetooth_Stop()
Bluetooth Connection Manag	*Inquire remote devices (with name resolution)*	hcitool -i hci0 inq	BT_Scan()
	Display active baseband connections	hcitool -i hci0 con	
	Retrieve Remote device name		BT_Get_Remote_Name()
	Look up if a device has bonding		BT_Has_Bonding()
	List devices bonded		BT_List_Bondings()
	Create the bonding		BT_Pair_Device()
	Remove the bonding		BT_Unpair_Device()
Bluetooth Service Manag	*Browse all available services on the remote device specified*	sdptool browse 00:0C:78:32:00:64	BT_Browse_Services()
	Search for a specific service	sdptool search SP	
	Connect a remote device to a specific service		BT_Connect_Services()
	Disconnect a remote device from a spec. serv		BT_Disconnect_Services()
Bluetooth SAP Manag.	*Create a Sap connection*		BT_Start_SAP()
	Remove a Sap connection		BT_Stop_SAP()
Bluetooth Headset Manag.	*Connect a headset device*		BT_Connect_Headset()
	Disconnect a headset device		BT_Disconnect_Headset()
	Set audio speaker volume		BT_Set_Volume_Gain()

	Get audio speaker volume		BT_Get_Volume_Gain()

All the shell commands seen above can be used from source code performing the "system" system call (i.e. system("hciconfig hci0 piscan");).

5 BlueZ Utilities

BlueZ Utilities (BU) is a set of tools that allow to configure and manage Bluetooth module by linux command shell.
BlueZ Utilities package includes the following executables:

- **hciattach:** attaches serial devices via UART HCI to BlueZ stack;
- **hciconfig:** configures Bluetooth module;
- **hcitool:** configures Bluetooth connections;
- **sdptool:** controls and interrogates SDP servers;
- **rfcomm:** manages the RFCOMM configuration of the local Bluetooth adapter.

BlueZ Utilities are part of BlueZ package, please have a look to paragraph 1.4.1 for information about BlueZ License.

The following subparagraphs describe BU commands as shown in man pages.

5.1.1 hciattach

Hciattach is used to attach a serial UART to the Bluetooth stack as HCI transport interface.

Synopsis

hciattach [-n] [-p] [-t timeout] tty type|id speed flow bdaddr

Options

-n Don't detach from controlling terminal.

-p Print the PID when detaching.

-t timeout

Specify an initialization timeout. (Default is 5 seconds.)

tty This specifies the serial device to attach. A leading /dev can be omitted. Examples: /dev/ttyS1 ttyS2

type|id

The type or id of the Bluetooth device that is to be attached, like vendor or other device specific identifier. Currently sup- ported types are

type description

any Unspecified HCI_UART interface, no vendor specific
options

ericsson
Ericsson based modules

digi Digianswer based cards

xircom Xircom PCMCIA cards: Credit Card Adapter and Real Port
Adapter

csr CSR Casira serial adapter or BrainBoxes serial dongle
(BL642)

bboxes BrainBoxes PCMCIA card (BL620)

swave Silicon Wave kits

bcsp Serial adapters using CSR chips with BCSP serial protocol

Supported IDs are (manufacturer id, product id)

0x0105, 0x080a
Xircom PCMCIA cards: Credit Card Adapter and Real Port
Adapter

0x0160, 0x0002
BrainBoxes PCMCIA card (BL620)

speed The speed specifies the UART speed to use. Baudrates higher than 115.200bps requires vendor specific initializations that are not implemented for all types of devices. In general the following speeds are supported:
9600, 19200, 38400, 57600, 115200, 230400, 460800, 921600
Supported vendor devices are automatically initialized to their respective best settings.

flow If the keyword flow is appended to the list of options then hardware flow control is forced on the serial link (CRTSCTS). All above mentioned device types have flow set by default. To force no flow control use noflow instead.

bdaddr The bdaddr specifies the Bluetooth Address to use. Some devices (like the STLC2500) do not store the Bluetooth address in hard-ware memory. Instead it must be uploaded during the initialization process. If this argument is specified, then the address will be used to initialize the device. Otherwise, a default address will be used.

5.1.2 hciconfig

hciconfig is used to configure Bluetooth devices. hciX is the name of a Bluetooth device installed in the system. If hciX is not given, hci-onfig prints name and basic information about all the Bluetooth devices installed in the system. If hciX is given but no command is given, it prints basic information on device hciX only. Basic information is interface type, BD address, ACL MTU, SCO MTU, flags (up, init, running, raw, page scan enabled, inquiry scan enabled, inquiry, authentication enabled, encryption enabled).

Synopsis

```
hciconfig -h
hciconfig [-a]
hciconfig [-a] [command [command parameters]]
```

Options

-h, --help
Gives a list of possible commands.

-a, --all
Other than the basic info, print features, packet type, link policy, link mode, name, class, version.

Parameters

up Open and initialize HCI device.

down Close HCI device.

reset Reset HCI device.

rstat Reset statistic counters.

auth Enable authentication (sets device to security mode 3).

noauth Disable authentication.

encrypt
Enable encryption (sets device to security mode 3).

noencrypt
Disable encryption.

secmgr Enable security manager (current kernel support is limited).

nosecmgr
Disable security manager.

piscan Enable page and inquiry scan.

noscan Disable page and inquiry scan.

iscan Enable inquiry scan, disable page scan.

pscan Enable page scan, disable inquiry scan.

ptype [type]
Without specifying type, displays the current packet types. Otherwise, all the packet types specified by type are set. type is a comma-separated list of packet types, where the possible packet types are DM1, DM3, DM5, DH1, DH3, DH5, HV1, HV2, HV3.

name [name]
Without specifying, prints local name. Otherwise, sets local name to name.

class [class]
Without specifying, prints class of device. Otherwise, sets class of device to class. class is a 24-bit hex number describing the class of device,
voice [voice]
Without specifying, prints voice setting. Otherwise, sets voice setting to voice. voice is a 16-bit hex number describing the voice setting.

iac [iac]
Without specifying iac, prints the current IAC setting. Otherwise, sets the IAC to iac.

inqtpl [level]
Without specifying level, prints out the current inquiry transmit power level. Otherwise, sets inquiry transmit power level to level.

inqmode [mode]
Without specifying mode, prints out the current inquiry mode. Otherwise, sets inquiry mode to mode.

inqdata [data]
Without specifying name, prints out the current inquiry data. Otherwise, sets inquiry data to data.

inqtype [type]
Without specifying type, prints out the current inquiry scan type. Otherwise, sets inquiry scan type to type.

inqparams [win:int]
Without specifying win:int, prints inquiry scan window and interval. Otherwise, sets inquiry scan window to win slots and inquiry scan interval to int slots.

pageparms [win:int]

Without specifying win:int, prints page scan window and interval. Otherwise, sets page scan window to win slots and page scan interval to int slots.

pageto [to]

Without specifying to, prints page timeout. Otherwise, sets page timeout to.I to slots.

afhmode [mode]

Without specifying mode, prints out the current AFH mode. Otherwise, sets AFH mode to mode.

sspmode [mode]

Without specifying mode, prints out the current Simple Pairing mode. Otherwise, sets Simple Pairing mode to mode.

aclmtu mtu:pkt

Sets ACL MTU to to mtu bytes and ACL buffer size to pkt packets.

scomtu mtu:pkt

Sets SCO MTU to mtu bytes and SCO buffer size to pkt packets.

putkey <bdaddr>

This command stores the link key for bdaddr on the device.

delkey <bdaddr>

This command deletes the stored link key for bdaddr from the device.

oobdata

Display local OOB data.

commands

Display supported commands.

features

Display device features.

version

Display version information.

revision

Display revision information.

lm [mode]

Without specifying mode , prints link mode. The modes MASTER, SLAVE mean, respectively, to ask to become master or to remain slave when a connection request comes in.
The mode ACCEPT means that the baseband connections will be accepted even if there are no listening AF_BLUETOOTH sockets.

The mode NONE sets link policy to the default behavior of remaining slave and not accepting baseband connections when there are no listening AF_BLUETOOTH sockets.

5.1.3 hcitool

hcitool is used to configure Bluetooth connections and send some special command to Bluetooth devices. If no command is given, or if the option -h is used, hcitool prints some usage information and exits.

Synopsis

hcitool [-h]
hcitool [-i <hciX>] [command [command parameters]]

Options

-h Gives a list of possible commands

-i <hciX>
The command is applied to device hciX , which must be the name of an installed Bluetooth device. If not specified, the command will be sent to the first available Bluetooth device.

Parameters

dev Display local devices

inq Inquire remote devices. For each discovered device, Bluetooth device address, clock offset and class are printed.

scan Inquire remote devices. For each discovered device, device name are printed.

name <bdaddr>
Print device name of remote device with Bluetooth address bdaddr.

info <bdaddr>
Print device name, version and supported features of remote device with Bluetooth address bdaddr.

spinq Start periodic inquiry process. No inquiry results are printed.

epinq Exit periodic inquiry process.

cmd <ogf> <ocf> [parameters]

Submit an arbitrary HCI command to local device. ogf, ocf and parameters are hexadecimal bytes.

con Display active baseband connections

cc [--role=m|s] [--pkt-type=<ptype>] <bdaddr> Create baseband connection to remote device with Bluetooth address bdaddr. Option --pkt-type specifies a list of allowed packet types. <ptype> is a comma-separated list of packet types, where the possible packet types are DM1, DM3, DM5, DH1, DH3, DH5, HV1, HV2, HV3. Default is to allow all packet types. Option --role can have value m (do not allow role switch, stay master) or s (allow role switch, become slave if the peer asks to become master). Default is m.

dc <bdaddr>
Delete baseband connection from remote device with Bluetooth address bdaddr.

sr <bdaddr> <role>
Switch role for the baseband connection from the remote device to master or slave.

cpt <bdaddr> <packet types>
Change packet types for baseband connection to device with Bluetooth address bdaddr. packet types is a comma-separated list of packet types, where the possible packet types are DM1, DM3, DM5, DH1, DH3, DH5, HV1, HV2, HV3.

rssi <bdaddr>
Display received signal strength information for the connection to the device with Bluetooth address bdaddr.

lq <bdaddr>
Display link quality for the connection to the device with Bluetooth address bdaddr.

tpl <bdaddr> [type]
Display power level transmission for the connection to the device with Bluetooth address bdaddr. The type can be 0 for the current power level in transmission (which is default) or 1 for the maximum power level in transmission.

afh <bdaddr>
Display AFH channel map for the connection to the device with Bluetooth address bdaddr.

lst <bdaddr> [value]
With no value, displays link supervision timeout for the connection to the device with Bluetooth address bdaddr. If value is given, sets the link supervision timeout for that connection to value slots, or to infinite if value is 0.

auth <bdaddr>
Request authentication of the device with Bluetooth address bdaddr.

enc <bdaddr> [encrypt enable]

Enable or disable the encryption for the device with Bluetooth address bdaddr.

key <bdaddr>
Change the connection link key for the device with Bluetooth address bdaddr.

clkoff <bdaddr>
Read the clock offset for the device with Bluetooth address bdaddr.

clock [bdaddr] [which clock]
Read the clock for the device with Bluetooth address bdaddr. The clock can be 0 for the local clock or 1 for the piconet clock (which is default).

5.1.4 sdptool

sdptool provides the interface for performing SDP queries on Bluetooth devices, and administering a local sdpd.

Synopsis
sdptool [options] {command} [command parameters ...]

Parameters
The following commands are available. In all cases bdaddr specifies the device to search or browse. If local is used for bdaddr, then the local sdpd is searched.

Services are identified and manipulated with a 4-byte record_handle (NOT the service name). To find a service's record_handle, look for the "Service RecHandle" line in the search or browse results

search [--bdaddr bdaddr] [--tree] [--raw] [--xml] service_name
Search for services.
Known service names are DID, SP, DUN, LAN, FAX, OPUSH, FTP, HS, HF, HFAG, SAP, NAP, GN, PANU, HCRP, HID, CIP, A2SRC, A2SNK, AVRCT, AVRTG, UDIUE, UDITE and SYNCML.

browse [--tree] [--raw] [--xml] [bdaddr]
Browse all available services on the device specified by a Bluetooth address as a parameter.

records [--tree] [--raw] [--xml] bdaddr
Retrieve all possible service records.

add [--handle=N --channel=N]
Add a service to the local sdpd.
You can specify a handle for this record using the --handle option.
You can specify a channel to add the service using the --channel option.

del record_handle
Remove a service from the local sdpd.

get [--tree] [--raw] [--xml] [--bdaddr bdaddr] record_handle
Retrieve a service from the local sdpd.

setattr record_handle attrib_id attrib_value
Set or add an attribute to an SDP record.

setseq record_handle attrib_id attrib_values
Set or add an attribute sequence to an SDP record.

5.1.5 rfcomm

rfcomm is used to set up, maintain, and inspect the RFCOMM configuration of the Bluetooth subsystem in the Linux kernel. If no command is given, or if the option -a is used, rfcomm prints information about the configured RFCOMM devices.

Synopsis
rfcomm [options] < command > < dev >

Options
-h Gives a list of possible commands.

-a Prints information about all configured RFCOMM devices.

-r Switch TTY into raw mode (doesn't work with "bind").

-f <file>
Specify alternate config file.

-i <hciX> | <bdaddr>
The command is applied to device -A Enable authentication. -E Enable encryption. -S Secure connection. -M Become the master of a piconet. hciX , which must be the name or the address of an installed Bluetooth device. If not specified, the command will be use the first available Bluetooth device.

-A Enable authentification

-E Enable encryption

-S Secure connection

-M Become the master of a piconet

-L <seconds>
Set linger timeout

Commands

show <dev>
Display the information about the specified device.

connect <dev> [bdaddr] [channel]
Connect the RFCOMM device to the remote Bluetooth device on the specified channel. If no channel is specified, it will use the channel number 1. If also the Bluetooth address is left out, it tries to read the data from the config file. This command can be terminated with the key sequence CTRL-C.

listen <dev> [channel] [cmd]
Listen on a specified RFCOMM channel for incoming connections. If no channel is specified, it will use the channel number 1, but a channel must be specified before cmd. If cmd is given, it will be executed as soon as a client connects. When the child process terminates or the client disconnect, the command will terminate. Occurences of {} in cmd will be replaced by the name of the device used by the connection. This command can be terminated with the key sequence CTRL-C.

watch <dev> [channel] [cmd]
Watch is identical to listen except that when the child process terminates or the client disconnect, the command will restart listening with the same parameters.

bind <dev> [bdaddr] [channel]
This binds the RFCOMM device to a remote Bluetooth device. The command does not establish a connection to the remote device, it only creates the binding. The connection will be established right after an application tries to open the RFCOMM device. If no channel number is specified, it uses the channel number 1. If the Bluetooth address is also left out, it tries to read the data from the config file.
If all is specified for the RFCOMM device, then all devices that have bind yes set in the config will be bound.

release <dev>
This command releases a defined RFCOMM binding.
If all is specified for the RFCOMM device, then all bindings will be removed. This command didn't care about the settings in the config file.

6 Library setup

It is possible to add the BT library on your development environment simply inserting the header file and the library, within the folder within the */opt/crosstools/telit/include/* and */opt/crosstools/telit/lib/* directories respectively:

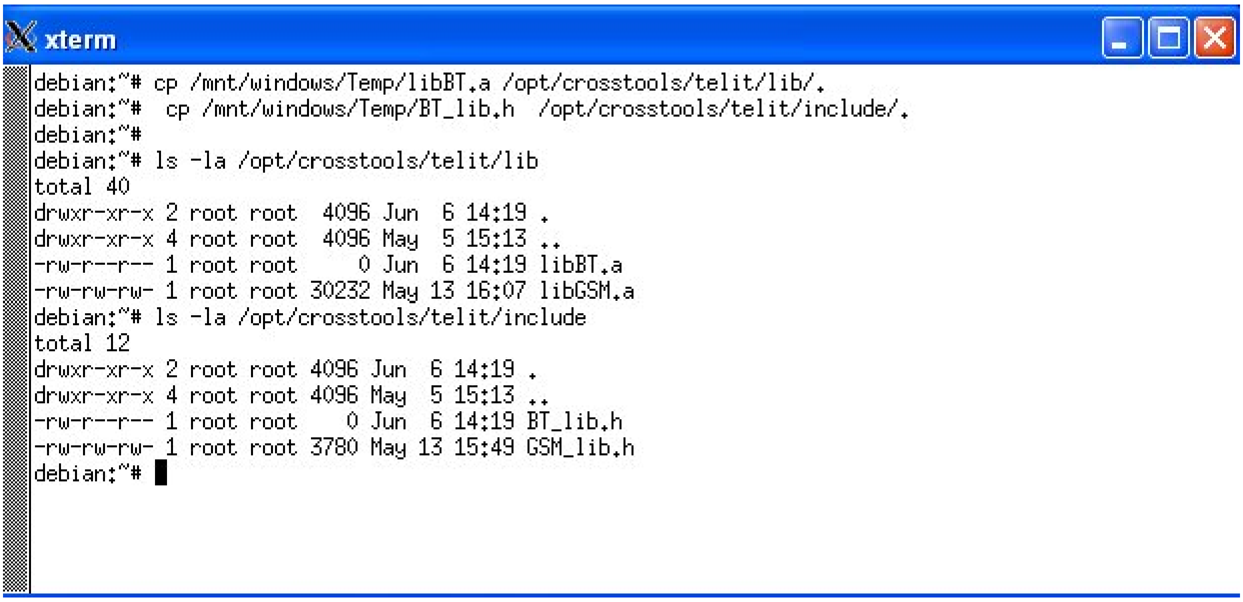
- Start the Linux console (Windows Start Menu → All Programs → Telit Development Platform → Console).
- Copy the library typing
  cp /mnt/windows/<PATH>/libBT.a /opt/crosstools/telit/lib
- Copy the header file typing
  cp /mnt/windows/<PATH>/BT_lib.h /opt/crosstools/telit/include

where **<PATH>** is the windows folder where you have stored the new version of the library files. For example, if you store them within C:\Temp you have to digit

cp /mnt/windows/Temp/libBT.a /opt/crosstools/telit/lib

and

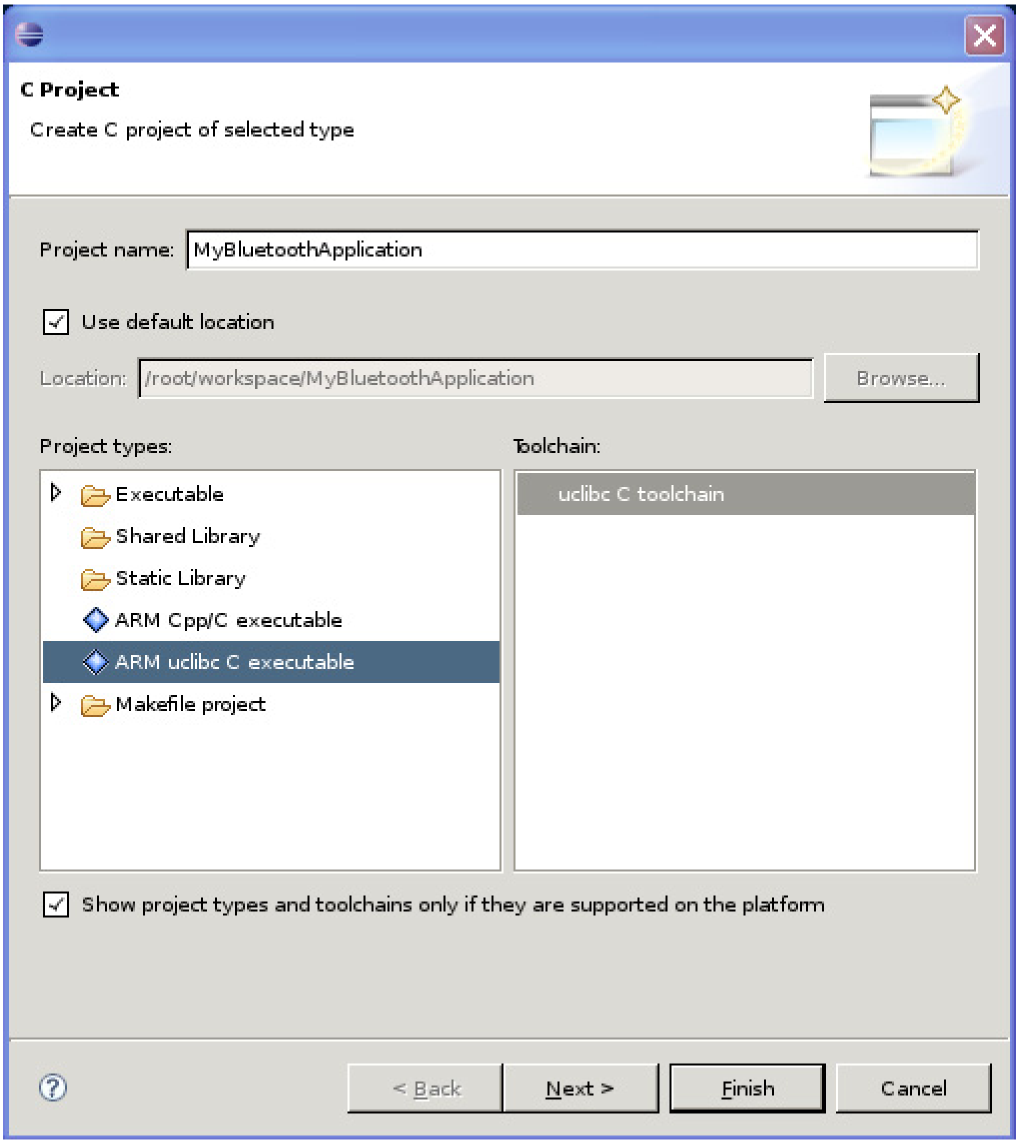
cp /mnt/windows/Temp/BT_lib.h /opt/crosstools/telit/include



```
debian:~# cp /mnt/windows/Temp/libBT.a /opt/crosstools/telit/lib/.
debian:~#  cp /mnt/windows/Temp/BT_lib.h  /opt/crosstools/telit/include/.
debian:~#
debian:~# ls -la /opt/crosstools/telit/lib
total 40
drwxr-xr-x 2 root root  4096 Jun  6 14:19 .
drwxr-xr-x 4 root root  4096 May  5 15:13 ..
-rw-r--r-- 1 root root     0 Jun  6 14:19 libBT.a
-rw-rw-rw- 1 root root 30232 May 13 16:07 libGSM.a
debian:~# ls -la /opt/crosstools/telit/include
total 12
drwxr-xr-x 2 root root 4096 Jun  6 14:19 .
drwxr-xr-x 4 root root 4096 May  5 15:13 ..
-rw-r--r-- 1 root root    0 Jun  6 14:19 BT_lib.h
-rw-rw-rw- 1 root root 3780 May 13 15:49 GSM_lib.h
debian:~#
```

6.1 How to Build a Client Application Project

Open your "Telit Customized Eclipse" starting from "Telit Development Platform".
Create a New Project "ARM uclibc C executable".
See figure below.



C Project

Create C project of selected type

Project name: MyBluetoothApplication

☑ Use default location

Location: /root/workspace/MyBluetoothApplication

Project types:

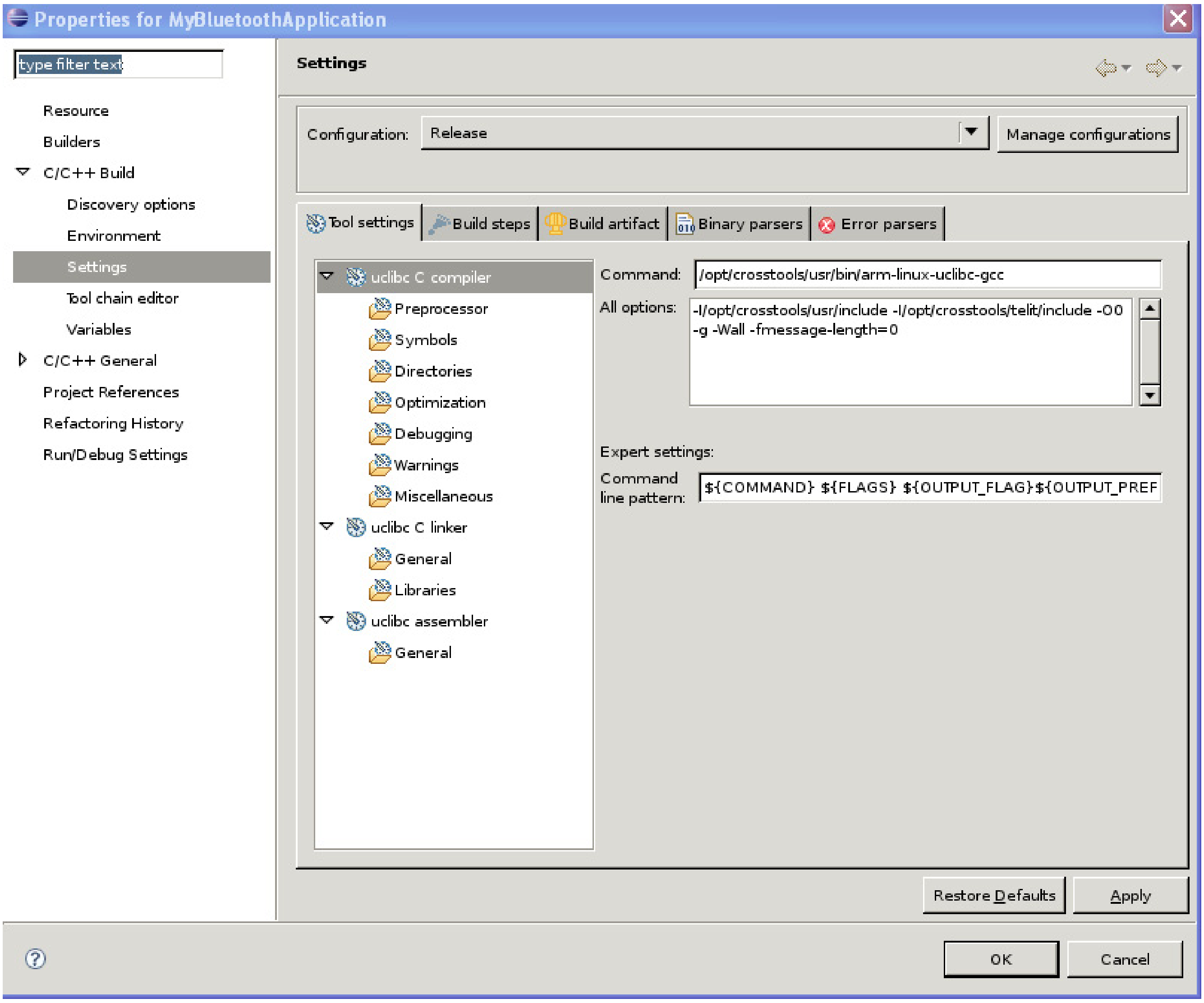
- Executable
- Shared Library
- Static Library
- ARM Cpp/C executable
- ARM uclibc C executable
- Makefile project

Toolchain:

- uclibc C toolchain

☑ Show project types and toolchains only if they are supported on the platform

Open new project Properties window end select C/C++ Build -> Setting.



Configuration: Release

Command: /opt/crosstools/usr/bin/arm-linux-uclibc-gcc

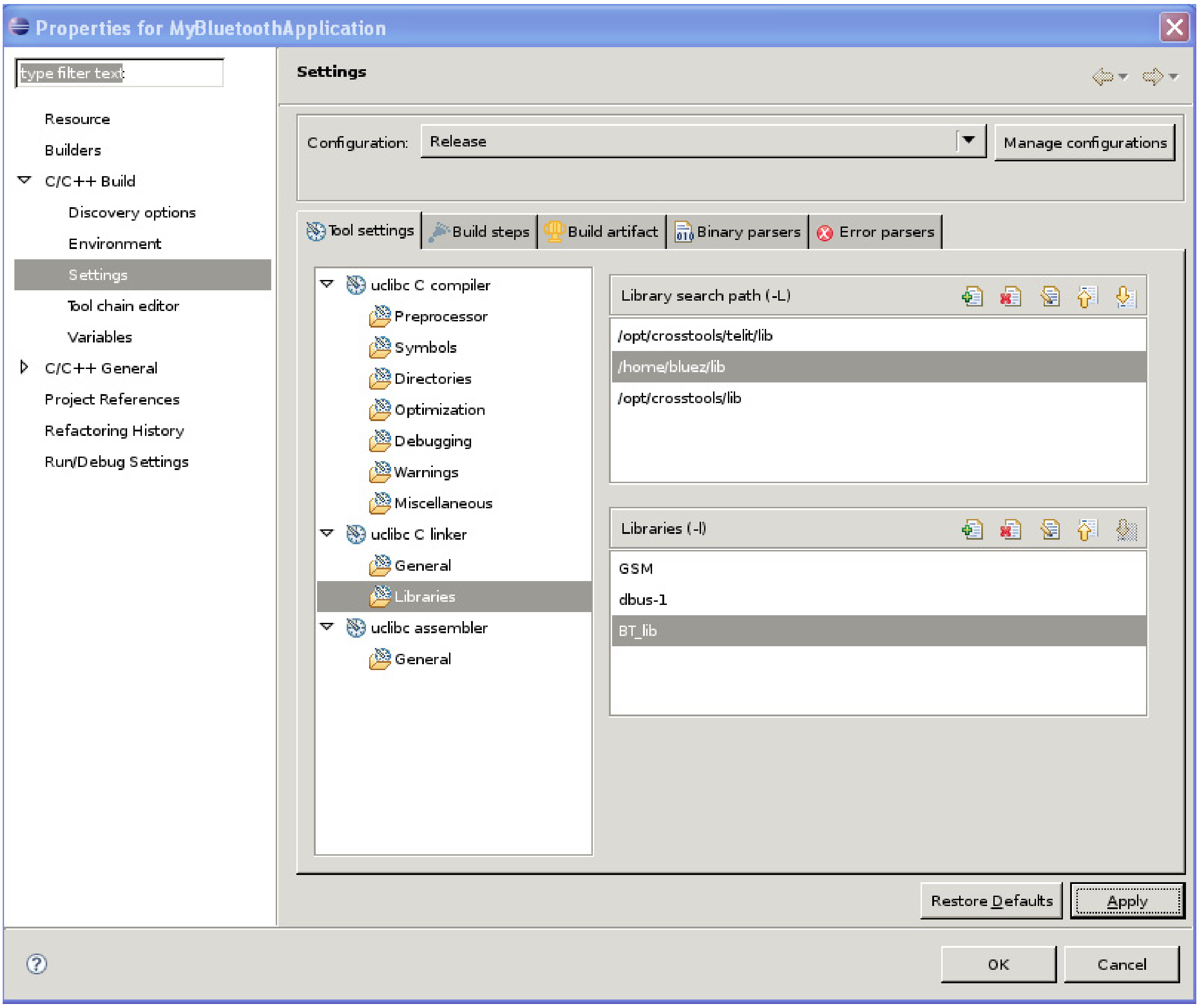
All options: -I/opt/crosstools/usr/include -I/opt/crosstools/telit/include -O0 -g -Wall -fmessage-length=0

Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLAG}${OUTPUT_PREF

Add in the uclib C linker -> Libraries the following libraries:

- libdbus-1.a library search path: /home/bluez/lib
- libBT_lib.a library search path: /opt/crosstools/telit/lib
- libpthread.so



Configuration: Release

Library search path (-L)
/opt/crosstools/telit/lib
/home/bluez/lib
/opt/crosstools/lib

Libraries (-l)
GSM
dbus-1
BT_lib

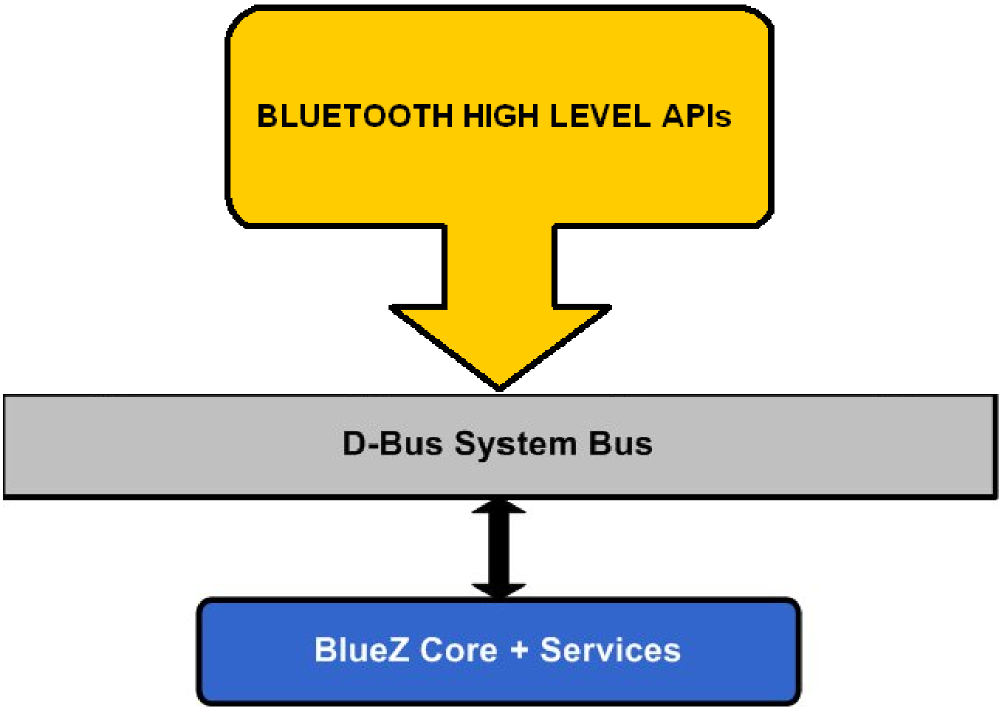
Click on "OK". Now you are ready to develop you Bluetooth Application.

7 Linux Bluetooth High Level APIs

With Bluetooth High Level APIs it is possible to control Bluetooth module, manage generic Bluetooth connection and perform some advanced tasks. These APIs are based on BlueZ D-Bus services, which are exported through the system message bus.

D-Bus is a message bus system which provides a simple way for applications to talk to one another.

The following picture explains relations between Bluetooth High Level APIs, D-Bus and BlueZ:



7.1 Description

Bluetooth high level API Package consist of the following files:

- BT_lib.h: generic API header file;
- libBT_lib.a: Telit Bluetooth static Library;
- BT_lib.conf: Telit Bluetooth Library configuration file;

7.1.1 Data Types

7.1.1.1 BT_Boolean_t

This type is an enum containing BT_True and BT_False values.

```
typedef enum {
        BT_False,
        BT_True
} BT_Boolean_t;
```

7.1.1.2 BT_Return_Code_t

This type is an enum containing codes for all errors that may occur during BT operations. Each function described in the next paragraph returns an error code.

```
typedef enum {
/* 0 */     BT_EXEC_OK,
/* 1 */     BT_ERROR,
/* 2 */     BT_HS_ERROR,
/* 3 */     BT_HS_NOT_CREATED,
/* 4 */     BT_HS_SIGNAL_NOT_RECEIVED,
/* 5 */     BT_HS_CHECK_CALL_ERROR,
/* 6 */     BT_HS_ANSWER_CALL_ERROR,
/* 7 */     BT_HS_GSM_AT_CMD_ERROR,
/* 8 */     BT_TIME_EXPIRED_ERROR,
/* 9 */     BT_HS_NOT_CONNECTED,
/* 10 */        BT_HS_NOT_PAIRED_ERROR,
        /* 11 */        BT_AUDIO_SERV_NOT_RUNNING,

        // Shared Errors (by BlueZ)
        /* 12 */        BT_DEVICE_UNREACHABLE_ERROR,
        /* 13 */        BT_ALREADY_CONNECTED_ERROR,
        /* 14 */        BT_CONNECTION_ATTEMPT_FAILED_ERROR,
        /* 15 */        BT_NOT_CONNECTED_ERROR,
        /* 16 */        BT_IN_PROGRESS_ERROR,
        /* 17 */        BT_INVALID_ARGUMENTS_ERROR,
        /* 18 */        BT_OUT_OF_MEMORY_ERROR,
        /* 19 */        BT_NOT_AVAILABLE_ERROR,
        /* 20 */        BT_NOT_SUPPORTED_ERROR,
        /* 21 */        BT_ALREADY_EXISTS_ERROR,
        /* 22 */        BT_DOES_NOT_EXISTS_ERROR,
        /* 23 */        BT_CANCELED_ERROR,
        /* 24 */        BT_FAILED_ERROR,

        // Hcid specific Errors (by hcid only)
        /* 25 */        BT_NOT_READY_ERROR,
        /* 26 */        BT_UNKNOWN_METHOD_ERROR,
        /* 27 */        BT_NOT_AUTHORIZED_ERROR,
        /* 28 */        BT_REJECTED_ERROR,
        /* 29 */        BT_NO_SUCH_ADAPTER_ERROR,
        /* 30 */        BT_NO_SUCH_SERVICE_ERROR,
        /* 31 */        BT_REQUEST_DEFERRED_ERROR,
        /* 32 */        BT_NOT_IN_PROGRESS_ERROR,
        /* 33 */        BT_AUTHENTICATION_CANCELED_ERROR,
        /* 34 */        BT_AUTHENTICATION_FAILED_ERROR,
        /* 35 */        BT_AUTHENTICATION_TIMEOUT_ERROR,
        /* 36 */        BT_AUTHENTICATION_REJECTED_ERROR,
        /* 37 */        BT_REPEATED_ATTEMPTS_ERROR,
        /* 38 */        BT_UNKNOWN_ERROR,
        /* 39 */        DBUS_BUS_GET_ERROR,
        /* 40 */        DBUS_MESSAGE_NEW_METHOD_CALL_ERROR_EMPTY_MSG,
        /* 41 */        DBUS_MESSAGE_APPEND_ARGS_ERROR,
        /* 42 */        DBUS_CONNECTION_SEND_ERROR,
        /* 43 */        DBUS_BUS_SEND_WITH_REPLY_AND_BLOCK_ERROR,
        /* 44 */        DBUS_BUS_ADD_MATCH_ERROR,
        /* 45 */        DBUS_MESSAGE_GET_ARGS_ERROR,
        /* 46 */        DBUS_UT_CONN_FREE_ERROR,
        /* 47 */        DBUS_UNKNOWN_ERROR,
        /* 48 */        BT_OUT_OF_RANGE,
        /* 49 */        BT_OPEN_FILE_ERROR,
        /* 50 */        BT_DBUS_CONNECTION_ERROR,
        /* 51 */        BT_SAP_BRIDGE_START_ERROR,
        /* 52 */        BT_SAP_BRIDGE_RUNNING_ERROR,
```

```
        /* 53 */         BT_SAP_BRIDGE_STOP_ERROR,
         /* 54 */         BT_SAP_GSM_STOP_ERROR,

    /* 55 */          BT_VALUE_NOT_FOUND_ERROR,
    /* 56 */         BT_DAEMONS_RUNNING_ERROR,
    /* 57 */          BT_DAEMONS_START_ERROR,
    /* 58 */         BT_SENDING_RING_ERROR,
    /* 59 */          BT_STOP_RING_ERROR

} BT_Return_Code_t;
```

7.1.1.3 BT_Addr_t

This type contains the Bluetooth address of a remote Bluetooth device. It should be in the following string form “XX:XX:XX:XX:XX:XX”.

```
#define BT_Addr_t char*
```

7.1.1.4 BT_Process_Id

This type contains the "pid" of a process.

```
#define BT_Process_Id  int
```

7.1.1.5 BT_Dev_Name_t

This type contains the friendly name of a remote Bluetooth device.

```
#define BT_Dev_Name_t char*
```

7.1.1.6 BT_Passkey_t

This type contains the passkey (Bluetooth PIN) associate with a remote Bluetooth device.

```
#define BT_Passkey_t char*
```

7.1.1.7 BT_Device_t

This type is a struct containg basic information about a remote Bluetooth device.

```
typedef struct BT_Device_t
{
        BT_Addr_t addr;
        BT_Dev_Name_t name;
        BT_Passkey_t passkey;
} BT_Device_t;
```

7.1.1.8 BT_Service_t

This type contains the name of a specific Bluetooth service.

```
#define BT_Service_t char*
```

7.1.1.9 BTList

This type define a generic list.

```
typedef struct _BTList
 {
        void *data;
        struct _BTList *next;
} BTList;
```

7.1.1.10 BT_Services_List_t

This data type will contain list of BT_Service_t.

```
#define BT_Services_List_t BTList
```

7.1.1.11 BT_Devices_List_t

This data type will contain list of BT_Device_t.

```
#define BT_Devices_List_t BTList
```

7.1.2 Configuration Files

7.1.2.1 bt_lib.conf

It has to be moved in the /etc/bluetooth directory. It contains the BT_lib configuration values. It is classified in three main groups: 'General', 'Headset' and 'Sap'.
'#' character indicates a comment line.

[General]
At the moment it doesn't contains any value.

[Headset]
It has to be chosen the port to send At command to the pro3 gsm module. Use "/dev/cmux3" if you are using the cmux (to use headset with sap profile or just sap profile you have to) or "/dev/ttyS3" if you don't need cmux to run your bluetooth application.

[Sap]
In order to use Telit SAP Client feature, the cmux must be activated. The field *“AtcommandPort”* specifies the cmuxt virtual port used internally by BT_lib APIs to send AT Commands to GSM engine. The field *“SAPmessagesPort”* specifies the cmux virtual port used internally by BT_lib APIs to exchange Remote SIM data with the GSM engine.

7.1.3 Linux Shell Script

7.1.3.1 BT_Bluetooth_Start.sh

Linux shell script "BT_Bluetooth_Start.sh" starts the Bluez end DBus daemons needed in order to call the BT_lib.api. The daemons started are "dbus-daemon", "hcid", "sdpd", "hciattach" and "auth-agent". It's up to the customer running or not running "cmux".

7.1.3.2 BT_Bluetooth_Stop.sh

Linux shell script "BT_Bluetooth_Stop.sh" stops DBus and Bluez daemons needed in order to call the BT_lib api. The processes stopped are "dbus-daemon", "hcid", "sdpd", "hciattach" and "auth-agent".

7.1.4 Functions

7.1.4.1 Generic Bluetooth procedures

char * BT_PrintLibVersion(void)

7.1.4.1.1 BT_PrintLibVersion()

This function print on the default standard output the current version of the BT_lib and returns a string which contains the info about version (Example 33.01.00).

Prototype

```
char*        BT_PrintLibVersion   (void)
```

Parameters

None

Return values

A string which contains information about the actual version of BT_lib.

7.1.4.1.2 BT_Scan()

This function starts the device discovery procedure. This includes an inquiry and an optional remote device name resolving.

Prototype

```
BT_Return_Code_t        BT_Scan   (BT_device_t  **info_device_scan,
                                     BT_Boolean_t name_resolving_enable_flag)
```

Parameters

<info_device_scan> It's a pointer to an array of BT_device_t. It will contain information about remote devices in range.

< name_resolving_enable_flag > It's a boolean_t that enables/disables retrieve of discoverable Bluetooth devices in range.

Return values

BT_EXEC_OK	Command correctly executed
not BT_EXEC_OK	An unpredictable error occurred

Example

```
BT_Devices_List_t* devicesList = NULL;

BT_Return_Code_t res = BT_EXEC_OK;

res = BT_Scan(&devicesList, BT_True);

BT_Devices_List_t *l;

for (l = devicesList; l != NULL; l = l->next)
{
      printf("...%s: ADDRESS: %s - NAME: %s\n",
                                  ((BT_Device_t*)(l->data))->addr,
                                  ((BT_Device_t*)(l->data))->name);
}

……..
……..

if (devicesList)
     list_free(deviceslist);
if (l)
    list_free(l);
```

7.1.4.1.3 BT_Pair_Device()

This function creates a bonding with a remote Bluetooth device using a specific passkey.
The passkey should be passed to this method as input parameter. If a link key for this adapter already exists, this method returns a *"BT_EXEC_OK"* instead of trying to create a new pairing.
If no connection to the remote device exists, a low-level ACL connection must be created.

Prototype

```
BT_Return_Code_t     BT_Pair_Device (BT_Addr_t  *remote_dev_addr ,
                                     BT_Passkey_t *remote_PASSKEY )
```

Parameters

<remote_dev_addr> It's a BT_Addr_t that contains the Bluetooth address of the remote Bluetooth device.

<remote_PASSKEY> It's a BT_Passkey_t that contains the passkey (Bluetooth PIN) associated to the remote Bluetooth device.

Return values

BT_EXEC_OK Command correctly executed. Pairing with the remote device created.

not BT_EXEC_OK An unpredictable error occurred

Example

```
char btAddress[BT_ADDRESS_SIZE];
char btRemPassKey[BT_PASSKEY_SIZE];

BT_Return_Code_t res = BT_EXEC_OK;

sprint(btAddress,"00:00:00:11:22:33");
sprint(btRemPassKey,"0000");

res = BT_Pair_Device(btAddress, btRemPassKey);
```

7.1.4.1.4 BT_Unpair_Device()

This function removes pairing with local device. For security reasons this includes removing the actual link key and also disconnecting any open connections for the remote device.

Prototype

```
BT_Return_Code_t     BT_Unpair_Device     (BT_Addr_t  remote_dev_addr  )
```

Parameters

<remote_dev_addr> It's a BT_Addr_t that contains the Bluetooth address of the remote Bluetooth device.

NOTE :
Please pay attention when removing the link key related to an active Bluetooth link. If it happens, the Bluetooth link will be lost, the related service will be stopped and a BT_FAILED_ERROR may be returned from the BT_unpair_Device() API. In order to avoid this behaviour, the service (like SAP or HSP) related to the Remote Bluetooth Device to unpair, should be stopped before performing the unpair procedure.

Return values

BT_EXEC_OK	Command correctly executed
!= (BT_EXEC_OK)	An unpredictable error occurred

Example

```
char btAddress[BT_ADDRESS_SIZE];

BT_Return_Code_t res = BT_EXEC_OK;
sprintf(btAddress,"00:00:00:11:22:33");

res = BT_Unpair_Device(btAddress);
```

7.1.4.1.5 BT_Has_Bonding()

This function returns BT_True if the remote Bluetooth device is bonded and BT_False if no link key is available.

Prototype

```
BT_Boolean_t    BT_Has_Bonding    (BT_Addr_t remote_dev_addr, BT_Return_Code_t*
                                   return_error )
```

Parameters

<remote_dev_addr> It's a BT_Addr_t that contains the Bluetooth address of the remote Bluetooth device.

<return_error> It’s a pointer to the location where to put the error code returned by BlueZ. Pass the NULL pointer if not interested on the reason of failure (Example BT_INVALID_ARGUMENTS_ERROR).

Return values

BT_True	A bonding with remote Bluetooth device already exists.
BT_False	No link key available for this remote device or an error occurs.

Example

```
char btAddress[BT_ADDRESS_SIZE];

BT_Return_Code_t res = BT_EXEC_OK;

BT_Boolean_t hasBond = BT_False;

sprintf(btAddress,"00:00:00:11:22:33");

hasBond = BT_Has_Bonding(btAddress,&res);

printf("Has %s Bonding?    %d\n", btAddress, hasBond);
```

7.1.4.1.6 BT_List_Bondings()

This function gets a list of the Bluetooth Address of the paired devices with local device.

Prototype

```
BT_Return_Code_t   BT_List_Bondings  (BT_Addr_t  **remote_addresses, int* num_devices)
```

Parameters

<remote_addresses>	It’s a pointer to an array of BT_Addr_t. It will contain the Bluetooth address of the remote Bluetooth devices paired with local Bluetooth adapter.
<num>	Number of remote Bluetooth devices paired with local Bluetooth adapter.

Return values

BT_EXEC_OK	Command correctly executed
!= (BT_EXEC_OK)	An unpredictable error occurred

Example

```
BT_Addr_t  *remote_addresses;
int remoteDevicesNum;
int i;

BT_Return_Code_t res = BT_EXEC_OK;

res = BT_List_Bondings(&remote_addresses,&remoteDevicesNum);

for (i=0;i<remoteDevicesNum;i++)
          printf("Address: %s\n", remote_addresses[i]);

……..
……..

for (i=0;i<remoteDevicesNum;i++){
   if remote_addresses[i]
      free(remote_addresses[i]);
}
```

7.1.4.1.7 BT_Set_Local_Name()

This function sets the local adapter name (friendly name).

Prototype

```
BT_Return_Code_t  BT_Set_Local_Name  (BT_Dev_Name_t   local_name)
```

Parameters

<local_name>	It’s a BT_Dev_Name_t that contains the friendly name to set.

Return values

BT_EXEC_OK	Command correctly executed
!= (BT_EXEC_OK)	An unpredictable error occurred

Example

```
BT_Return_Code_t res = BT_EXEC_OK;
char localName[50];

sprintf(localName,"BT-LocalDevice");

res = BT_Set_Local_Name(localName);
```

7.1.4.1.8 BT_Get_Local_Name()

This function retrieves the local adapter name (friendly name).

Prototype

```
BT_Return_Code_t  BT_Get_Local_Name  (BT_Dev_Name_t   *local_name)
```

Parameters

<local_name>	It’s a BT_Dev_Name_t pointer that will contain the friendly name retrieved.

Return values

BT_EXEC_OK	Command correctly executed
!= (BT_EXEC_OK)	An unpredictable error occurred

Example

```
BT_Dev_Name_t deviceName;
BT_Return_Code_t res = BT_EXEC_OK;

res = BT_Get_Local_Name(&deviceName);

printf("Local  Device Name: %s\n", deviceName);
```

```
 ……..
……..
if (deviceName)
     free(deviceName);
```

7.1.4.1.9 BT_Get_Remote_Name()

This function retrieves the name (friendly name) of the specified remote Bluetooth device. This method retrieves always a cached name and an error code is returned if the name is not in the cache. In order to update the cache, a BT_Scan() with name resolution or a BT_Browse_Services() should be performed.

Prototype

```
BT_Return_Code_t    BT_Get_Remote_Name    (BT_Addr_t  remote_dev_addr ,
                                                 BT_Dev_Name_t   *remote_name)
```

Parameters

<remote_dev_addr> It's a BT_Addr_t that contains the Bluetooth address of the remote Bluetooth device.

<remote_name> It's It's a BT_Dev_Name_t that will contain the friendly name of the remote Bluetooth device retrieved.

Return values

BT_EXEC_OK	Command correctly executed
!= (BT_EXEC_OK)	An unpredictable error occurred

Example

```
BT_Return_Code_t res = BT_EXEC_OK;


char btAddress[BT_ADDRESS_SIZE];

BT_Dev_Name_t remote_name;
```

```
sprintf(btAddress,"00:00:00:11:22:33");

res = BT_Get_Remote_Name(btAddress, &remote_name);

printf("BT_ADDRESS: %s - NAME: %s.\n", btAddress, remote_name);

……..
……..

if  (remote_name)
    free(remove_name);
```

7.1.4.1.10 BT_Browse_Services ()

This method will request the SDP database of a remote device and retrieve information about services available.
Pay attention that you can perform a services Browsing of a remote device with security level "3" (like headset) only if the local device has already executed a pair with that remote device.

Prototype

```
BT_Return_Code_t        BT_Browse_Services   (BT_Addr_t remote_bt_addr,
                                                 BT_Services_List_t** services_list);
```

Parameters

<remote_address> It's a BT_Addr_t that contains the Bluetooth address of the remote Bluetooth device

<services_list> It's a pointer to a pointer to BT_Services_List_t that contains info about browsed services.

Return values

BT_EXEC_OK	Command correctly executed
!= (BT_EXEC_OK)	An unpredictable error occurred

Example

```
BT_Return_Code_t res = BT_EXEC_OK;
BT_Services_List_t* services_list= NULL;

res = BT_Browse_Services("00:18:88:66:9B:00",&services_list);

BT_Services_List_t *l1;

for (l1 = services_list; l1 != NULL; l1 = l1->next)
{
      printf("...%s: SERVICE: %s\n", ((BT_Service_t)l1->data));
}

……..
……..

if (services_list=)
       list_free(services_list=);
if (l1)
      list_free(l1);
```

7.1.4.2 SAP Bluetooth Procedures

7.1.4.2.1 *BT_Start_SAP()*

This function creates a connection toward a remote SAP Server and starts Telit SAP Client inside GE863. In order to start the Telit SAP Client inside the GSM engine, the cmux must be activated before call this method. Else an error code will be returned.

Prototype

```
BT_Return_Code_t    BT_Start_SAP    (BT_Addr_t remote_dev,  BT_Passkey_t
                                                           remote_PASSKEY);
```

Parameters

<remote_dev > It’s a BT_Addr_t that contains the Bluetooth address of the remote

SAP Server

<remote_PASSKEY> It's a BT_Passkey_t that contains the passkey (Bluetooth PIN) associated to the SAP service.

NOTE :

In order to improve security, the SAP server may require a passkey longer than the actual passkey used during a previous pairing procedure. In this situation the remote_PASSKEY parameter is required, in order to avoid connection failure.
Some SAP server will not ask again for a stronger passkey and reject the connection; in this situation an "unpair" procedure is required before a connection procedure toward SAP service.
If the PRO3 has not been paired with the SAP Server yet, this parameter is required, in order to perform pairing procedure before connection to the service. If a pair with a strong passkey is already present between PRO3 and SAP server, the NULL value can be passed instead a valid remote passkey.

Return values

BT_EXEC_OK	Command correctly executed. Telit SAP Client is running.
!= (BT_EXEC_OK)	An unpredictable error occurred

Example

```
char btAddress[BT_ADDRESS_SIZE];
char btRemPassKey[BT_PASSKEY_SIZE];

BT_Return_Code_t res = BT_EXEC_OK;

sprintf(btAddress,"00:00:00:11:22:33");
sprintf(btRemPassKey,"1234567891234567");

res = BT_Start_SAP(btAddress, btRemPassKey);
```

7.1.4.2.2 BT_Stop_SAP()

This function starts the "Disconnect Initiated by the Client" procedure. If it goes successfully the RFcomm data channel, between the Client and the Server, shall be immediately disconnected and Telit SAP Client inside GE863-PRO^{3} will be stopped.

Prototype

```
        BT_Return_Code_t    BT_Stop_SAP    (BT_Addr_t remote_dev);
```

Parameters

<remote_dev> It's a BT_Addr_t that contains the Bluetooth address of the remote SAP Server

Return values

BT_EXEC_OK	Command correctly executed. Telit SAP Client is stopped.
!= (BT_EXEC_OK)	An unpredictable error occurred

Example

```
char btAddress[BT_ADDRESS_SIZE];

BT_Return_Code_t res = BT_EXEC_OK;

sprintf(btAddress,"00:00:00:11:22:33");


res  = BT_Stop_SAP(remote_dev);
```

7.1.4.3 Headset Bluetooth Procedures

7.1.4.3.1 BT_Headset_Start()

This function has to be called before any of the following headset utilities.
So, if you want to link a headset to your local Bluetooth device, you have to call BT_Headset_Start() and afterwards the function BT_Headset_Stop(). It executes a forked function in order to catch any Headset press button (AT+CKPD). The remote Headset must have been already paired to the local device. The int pid output parameter have to be used to call BT_Headset_Stop() function .

Prototype

```
    BT_Return_Code_t      BT_Headset_Start (BT_Addr_t remote_bt_addr,
                                            BT_Passkey_t   remote_PASSKEY,int* pid)
```

Parameters

<remote_bt_addr> It's a BT_Addr_t that contains the Bluetooth address of the

remote Bluetooth device.

<remote_PASSKEY >   It’s a BT_Passkey_t that contains the pin code to use in pairing process.

<pid>   Forked process ID needed to kill the process at the end.

Return values

BT_EXEC_OK	Command correctly executed
!= (BT_EXEC_OK)	An unpredictable error occurred

Example

```
char btAddress[BT_ADDRESS_SIZE];
char btRemPassKey[BT_PASSKEY_SIZE];
int pid;

BT_Return_Code_t res = BT_EXEC_OK;

sprint(btAddress,"00:00:00:11:22:33");
sprint(btRemPassKey,"0000");

res = BT_Headset_Start(btAddress, btRemPassKey,&pid);

……..
……..
……..

res = BT_EXEC_OK;

res = BT_Headset_Stop(pid);
```

7.1.4.3.2 BT_Headset_Stop()

This function has to be called when finished using Headset utilities. It kill the process forked with the BT_Headset_Start() function.

Prototype

```
     BT_Return_Code_t      BT_Headset_Stop (int pid)
```

Parameters

<pid>	Forked process ID needed to kill the process. It's the value returned by the BT_Headset_Start() function.

Return values

BT_EXEC_OK	Command correctly executed
!= (BT_EXEC_OK)	An unpredictable error occurred

Example

```
// see BT_Headset_Start() Example

………..

……......

res = BT_EXEC_OK;

res = BT_Headset_Stop(pid);
```

7.1.4.3.3 BT_Connect_Headset ()

This function connects the local device with a headset device. All the preliminary steps (like SDP query) are internal. This function connects the local device to the HSP service on the remote device. The remote Headset must have been already paired to the local device. If the headset is already connected, it doesn't do anything.

Prototype

```
BT_Return_Code_t  BT_Connect_Headset (BT_Addr_t remote_address , BT_Passkey_t
                                                                    pin_code)
```

Parameters

<remote_address> It's a BT_Addr_t that contains the Bluetooth address of the remote Bluetooth device.

<pin_code> It's a BT_Passkey_t that contains the pin code to use in pairing process.

Return values

BT_EXEC_OK Command correctly executed, connection with headset device correctly established.

!= (BT_EXEC_OK) An unpredictable error occurred

Example

```
char btAddress[BT_ADDRESS_SIZE];
char btRemPassKey[BT_PASSKEY_SIZE];

BT_Return_Code_t res = BT_EXEC_OK;

sprint(btAddress,"00:00:00:11:22:33");
sprint(btRemPassKey,"0000");

res = BT_Connect_Headset(btAddress, btRemPassKey);
```

7.1.4.3.4 BT_Disconnect_Headset ()

This function disconnects from the HSP service on the remote device and removes all information related to the headset device. If the headset is already disconnected, it doesn't do anything.

Prototype

```
BT_Return_Code_t   BT_Disconnect_Headset (BT_Addr_t remote_address,
                                          BT_Passkey_t pin_code)
```

Parameters

<remote_address> It's a BT_Addr_t that contains the Bluetooth address of the remote Bluetooth device.

<pin_code> It's a BT_Passkey_t that contains the pin code to use in pairing process.

Return values

BT_EXEC_OK Command correctly executed, headset device correctly disconnected and removed

!= (BT_EXEC_OK) An unpredictable error occurred

Example

```
char btAddress[BT_ADDRESS_SIZE];
char btRemPassKey[BT_PASSKEY_SIZE];

BT_Return_Code_t res = BT_EXEC_OK;

sprint(btAddress,"00:00:00:11:22:33");
sprint(btRemPassKey,"0000");

res = BT_Disconnect_Headset(btAddress, btRemPassKey);
```

7.1.4.3.5 BT_Set_Speaker_Volume_Gain ()

This function set speaker volume gain for the remote BT_Addr_t specified in the parameter. It is provided only for device that support audio (like headset). The headset must be connected to the AG.

Prototype

```
BT_Return_Code_t  BT_Set_Speaker_Volume_Gain (BT_Addr_t remote_address,
                                              BT_Passkey_t  pin_code,
                                              unsigned short volume_gain)
```

Parameters

<remote_address> It's a BT_Addr_t that contains the Bluetooth address of the remote Bluetooth device.

<pin_code> It's a BT_Passkey_t that contains the pin code to use in pairing process.

<volume_gain> It's a number indicating speaker gain to set.

Return values

BT_EXEC_OK Command correctly executed; volume gain correctly set for the remote device

!= (BT_EXEC_OK) An unpredictable error occurred

Example

```
int volumeGain = 0;
char btAddress[BT_ADDRESS_SIZE];
char btRemPassKey[BT_PASSKEY_SIZE];

BT_Return_Code_t res = BT_EXEC_OK;

printf("\n\n\nInsert New Speaker Volume Value (0..15)\n: ");

scanf("%d", &volumeGain);

res = BT_Set_Speaker_Volume_Gain(btAddress, btRemPassKey,(unsigned short) volumeGain);
```

7.1.4.3.6 BT_Get_Speaker_Volume_Gain ()

This function gets speaker volume gain for the remote BT_Addr_t specified in the parameter.
The headset must be connected to the AG.

Prototype

```
BT_Return_Code_t  BT_Get_Speaker_Volume_Gain (BT_Addr_t  remote_address,
                                               BT_Passkey_t  pin_code,
                                              unsigned short *volume_gain)
```

Parameters

<remote_address> It’s a BT_Addr_t that contains the Bluetooth address of the remote Bluetooth device.

<pin_code> It’s a BT_Passkey_t that contains the pin code to use in pairing process.

<volume_gain> It’s a pointer to an unsigned short indicating speaker gain of the remote device

Return values

BT_EXEC_OK	Command correctly executed
!= (BT_EXEC_OK)	An unpredictable error occurred

Example

```
unsigned short speakVolume;

char btAddress[BT_ADDRESS_SIZE];
char btRemPassKey[BT_PASSKEY_SIZE];

BT_Return_Code_t res = BT_EXEC_OK;

res = BT_Get_Speaker_Volume_Gain(btAddress, btRemPassKey,&speakVolume);


printf("Speaker Volume Gain: %d\n", speakVolume);
```

8 List of acronyms and Abbreviation

Acronym	Explanation
HSP	Hands Free Profile
GSM	Global System for Mobile communications
IP	Internet Protocol
SAP	Sim Access Profile
PDU	Protocol Data Unit
PIN	Personal Identification Number
PPP	Point to Point Protocol
PUK	Personal Unblocking Key
SIM	Subscriber Identity Module
SMS	Short Message Service
TCP	Transmission Control Protocol