



Java Bluetooth stack Technical Documentation

Version 0.5

Doc. No.:

Java Bluetooth stack	Version: 0.5
Developer's Guide	Date: 2004-01-10

Revision History

Date	Version	Description	Author
2004-01-10	0.1	Initial Draft	Marko Đurić
2004-01-11	0.4	Added Installation guides and chapter about using CVS from Eclipse Platform	Tomislav Sečen
2004-01-11	0.5	Minor updates	Marko Đurić

Java Bluetooth stack	Version: 0.5
Developer's Guide	Date: 2004-01-10

Table of Contents

1.	Introduction	6
1.1	Purpose of this document	6
1.2	Intended Audience	6
1.3	Scope	6
1.4	Definitions and acronyms	6
1.4.1	Definitions	6
1.4.2	Acronyms and abbreviations	6
2.	Acknowledgements	7
3.	Motivation	7
4.	Bluetooth technology overview	7
4.1	Harald stack	7
4.2	General Description	7
4.3	Network topology	8
4.4	Bluetooth SIG	9
4.5	Where can Bluetooth be useful?	9
4.6	Layers	9
4.6.1	Baseband	10
4.6.2	Link Control (LC)	11
4.6.3	Link Manager (LM)	11
4.6.4	Host Controller Interface (HCI)	11
4.6.5	Transport Interface	12
5.	Installation guide – software	12
5.1	Installing Eclipse Platform and Java commAPI and configuring environment variables for commAPI	12
5.1.1	Eclipse Platform installation	13
5.1.2	Sun Wireless ToolKit installation	13
5.1.3	Installing commAPI	13
5.1.4	Importing project into workspace	14
5.1.5	Running samples and JUnit tests	14
6.	Installation guide – hardware	14
6.1	Xircom CBT	15
6.2	Generic Bluetooth serial module install instructions	16
	Appendix A: Creating Eclipse Runtime configurations	18
	Appendix B: Checking and configuring project references	19
	Appendix C: Eclipse JUnit support	20
	Writing JUnit tests:	20
	Running JUnit tests	23
	Creating a new JUnit Run Configuration	24
	Creating a JUnit test suite	25
	Appendix D: Eclipse & CVS	27
	Creating a CVS repository location	27
	Sharing a new project using CVS	28
	Checking out a project from a CVS repository	30

Java Bluetooth stack	Version: 0.5
Developer's Guide	Date: 2004-01-10

Updating	31
Committing	35
Synchronizing with a CVS repository	36
Literature and resources	37
Books	37
Links on the Internet	37
Index	37

Java Bluetooth stack	Version: 0.5
Developer's Guide	Date: 2004-01-10

1. Introduction

1.1 Purpose of this document

Technical Documentation provides technical information about Java Bluetooth Stack. It gives some basic introduction to the Bluetooth technology, and the technical guidance for using the Java Bluetooth Stack.

1.2 Intended Audience

This document is intended for Java Bluetooth developers or anyone interested in using Java Bluetooth stack.

1.3 Scope

Bluetooth technology for wireless communication is described by the layers of the Bluetooth stack and by the basic principles of the Bluetooth communications.

Java Bluetooth Stack is introduced to the users/developers package-by-package, so it can be easily used later in developing Bluetooth java applications.

1.4 Definitions and acronyms

1.4.1 Definitions

Keyword	Definitions
Piconet	More Bluetooth devices organized in a group
Scatternet	Formation of more piconets connected together in one network

1.4.2 Acronyms and abbreviations

Acronym or abbreviation	Definitions

Java Bluetooth stack	Version: 0.5
Developer's Guide	Date: 2004-01-10

2. Acknowledgements

This project wouldn't have the form it has today if it weren't for two great Bluetooth stacks: [Harald's Bluetooth stack](#), created by John Eker, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, and [javablueetooth](#), created by Christian Lorentz. This project is based on javablueetooth, and it uses some code from Harald. We would like to thank both of them for creating these great stacks, and for teaching us a dozen of neat in-depth Java tricks.

Another great help in coding of RFCOMM protocol layer was the official Linux Bluetooth protocol stack, [BlueZ](#), so we would like to thank them also.

3. Motivation

Motivation for this project comes mainly because of lack of free Bluetooth stacks that has implemented some higher-level layers and protocols (such as RFCOMM or OBEX).

4. Bluetooth technology overview

4.1 Harald stack

A little bit of history?

Harald Bluetooth (originally Blåtand) was a Viking king who ruled all of Denmark and Norway in the 10th century (960 – 986).

Harald Blåtand was a very active king. He won the whole of Denmark and Norway and large enterprises were commenced during his time as king; much of Harald's history was learnt from two runic stones erected in the town of Jelling in Denmark.

Did he actually have blue teeth, you must wonder? No, Blåtand actually means dark complexion – he had very dark hair, which was unusual for Vikings. Not only did Harald not fit the classic image physically, he was a rather unusual Viking.

That is, if your understanding was that the life of a Viking was all battles and pillage. The good King Harald brought Christianity to Scandinavia and also united Denmark and Norway.

When he was first crowned in 960, he erected a runic stone – monument to his parents – with the following text written (in symbols known as runes) on it: "Harald king executes these sepulchral monuments after Gorm, his father and Thyra, his mother. The Harald who won the whole of Denmark and Norway and turned the Danes to Christianity."

A millennium later, in 1994 in Lund, Sweden, a new wireless technology for mobile communication was named after the great king, and his name became famous worldwide.

4.2 General Description

Bluetooth is wireless communication protocol that was originally invented 1994. by Swedish phone equipment maker Ericsson. Its purpose is wireless communication between mobile devices on short distances (up to 100m).



Fig. 4.1.1: Runic stone in memory of Harald's parents

Java Bluetooth stack	Version: 0.5
Developer's Guide	Date: 2004-01-10

Bluetooth operates in 2.4GHz radio band that is reserved for Industrial – Scientific – Medical (ISM) purposes. Because it communicates by radio signals, there is no need for optical visibility between devices.

There are 79 available RF channels with base frequency $f = (2402+k)\text{MHz}$, $k=0,1,\dots,78$. Channel spacing is 1MHz, and in order to comply Bluetooth frequency range of 2400MHz – 2483.5MHz, there is 2MHz Lower Guard Band, and 3,5MHz Upper Guard Band. For added security, channels use frequency hopping (frequency is changed in pseudo-random pattern 1600 times per second).

Some countries have national limitations in the frequency range, so the Bluetooth frequency range is reduced in those countries. Because of that, products implementing the reduced frequency band **will not work** with products implementing the full band.

There may be some questions why use Bluetooth, when there is existing Infrared or 802.11b wireless communication standards.

When comparing with Infrared communication, the answer is pretty simple – Infrared must have optical visibility between the devices, and that's the main disadvantage. For that reason, it's better to deal with comparison of Bluetooth and 802.11b wireless protocols.

The main difference between Bluetooth and 802.11b is the different goals of those two communication protocols. While 802.11b is intended to be used for connecting relatively large devices that uses lots of power at high speeds (11Mbps), Bluetooth is the complete opposite of that – it is used for small (mobile) devices that runs on batteries (or other low-power supply), and because of that, for lower speed communications on short distances (10m (=30ft) at max of 1Mbps).

4.3 Network topology

Intention of Bluetooth devices is ad-hoc connectivity of small devices like mobile phones or PDA-s and for that reason the range is limited to 10m (max. of 100m in Class I devices – larger devices with better power supply (more capacity)). It is mostly used for small amount of data transfer between devices (asynchronous mode) or for speech (synchronous mode).

Bluetooth connection philosophy is very simple – mobile devices that wants to communicate via Bluetooth connection must be able to create small networks with minimum of user interaction (principle of ad-hoc networking).

Ad-hoc networking by definition means that communicating devices can spontaneously form a community of networks that persists only as long as it's needed. Other RF networking (802.11b for example) needs user interaction for creating and administrating the network.

Two or more Bluetooth-enabled devices sharing the same channel are organized in groups. This groups are called *piconets* and they can contain maximum of eight devices (one master and up to seven active slaves). A master unit is the device that initiates the communication.

A device in one piconet can communicate to other device in another piconet, and with that connection it forms a *scatternet*. Device that is slave in one piconet can be master in another piconet.

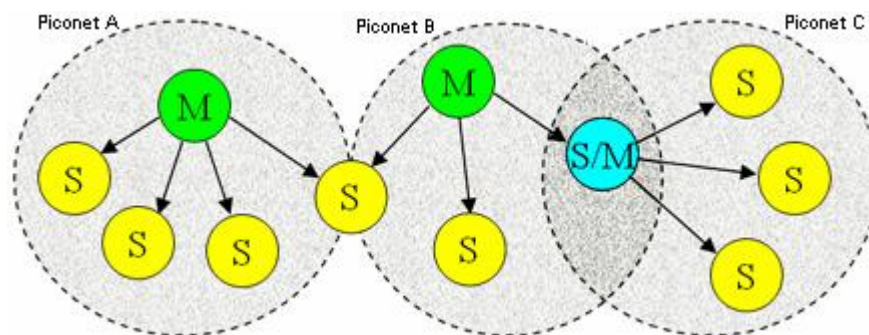


Fig. 4.3.2: More Piconets forms a Scatternet

The normal duration of transmission is one time slot (625µs), and a packet can last up to five time slots in length. For achieving full-duplex communication, Bluetooth uses TDM (time-division multiplexing) scheme, in which a master device always uses an even-numbered slot, and slave uses a odd-numbered slot.

4.4 Bluetooth SIG

In 1997, some major companies (such as IBM, Intel, Nokia and Toshiba) joined with Ericsson into Bluetooth Special Interest Group (SIG) consortium for further development of Bluetooth protocol. After the release of the first Bluetooth specification (version 1.0), 3COM, Agere, Microsoft and Motorola also joined the SIG consortium. Bluetooth SIG now has more than 2000 members.

Bluetooth SIG is responsible for updating the Bluetooth specifications and for promoting and improving the Bluetooth standard.

4.5 Where can Bluetooth be useful?

Bluetooth is useful for connecting low-power mobile devices on short ranges. Here are some of the main advantages of Bluetooth:

- communication with peripherals (i.e. computer -> printer communication, etc.)
- very low power needed for Bluetooth communications (1mW on Class 3 devices)
- it is specially designed for smaller data transfers (synchronization, for example)
- Bluetooth is inexpensive (relatively low cost of incorporating Bluetooth technology to existing PDA-s, cell-phones, etc.)
- it uses regulated, but unlicensed radio frequency band (ISM band)
- communicating devices don't need an unobstructed line of sight between them
- Bluetooth uses frequency hopping so there is very low possibility that communications will be intercepted
- Bluetooth can handle both voice and data communications

Because of this advantages, there is a lot of possibilities for using Bluetooth communication. It can be used anywhere where is need for transfer of small amount of data on shorter ranges (synchronization, for example) and it can be also used as a cable replacement technology.

Bluetooth is not useful for large file transfers or long-range communication, but that's not the intention of Bluetooth at all. For that purposes there is 802.11b protocol (or other versions of that protocol (802.11a, 802.11g)).

4.6 Layers

Bluetooth stack has a layered structure (like referent ISO/OSI network model). It consists of protocols that are specific to Bluetooth (L2CAP, SDP, etc.) and other adopted protocols (i.e. OBEX).

There are four main groups in Bluetooth stack:

- Bluetooth core protocols – Baseband, Link Manager Protocol, L2CAP and SDP
- Cable replacement protocol – RFCOMM
- Telephony control protocol – TCS Binary
- Adopted protocols – PPP, UDP/TCP/IP, OBEX, WAP

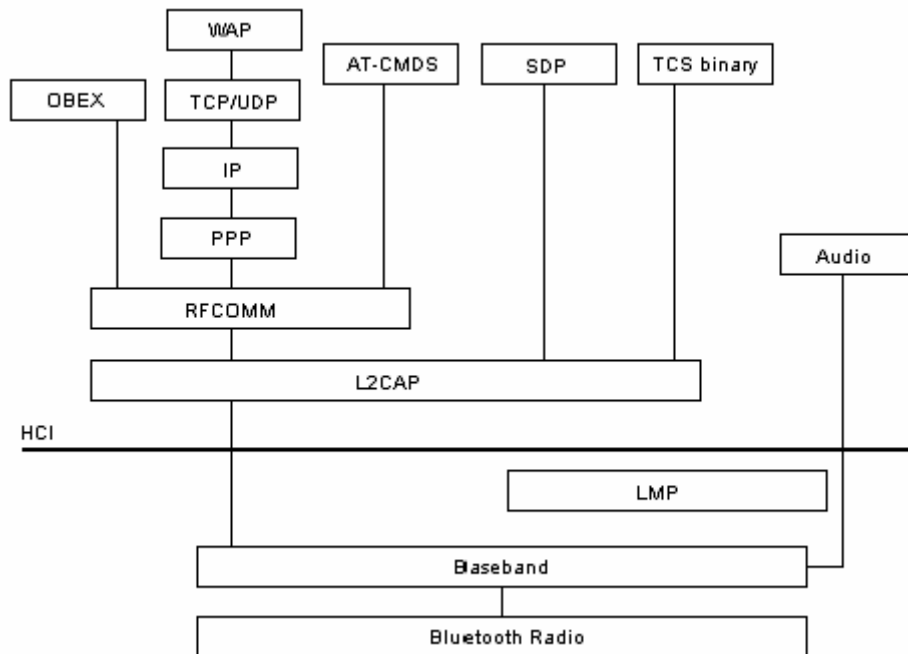


Fig. 4.6.1: Layered structure of Bluetooth stack

4.6.1 Baseband

Bluetooth operates in the unlicensed ISM band at 2.4GHz. It uses binary FM modulation to minimize transceiver complexity, and frequency hop for preventing interference with other devices that are communicating on the same frequency. Communication is full-duplex, and that is achieved by using TDD (time-division duplex) scheme, where time is divided in slots. Every slot has length of 625 μ s (there is 1600 frequency hops in second). Packets are usually sent in single slot, but it can be extended up to 5 slots.

Baseband layer is the Bluetooth layer that is positioned right above the Bluetooth radio. It enables physical radio-frequency (RF) link between Bluetooth enabled devices that wants to communicate. Its function is to manage Bluetooth channels (frequency hopping, time-slots, etc.) and packet transmissions.

Packets defined by Baseband Specification are sent in Little Endian format. That means the Least Significant Bit (LSB) is first sent over the air. General packet format is shown on Fig. 4.6.2.

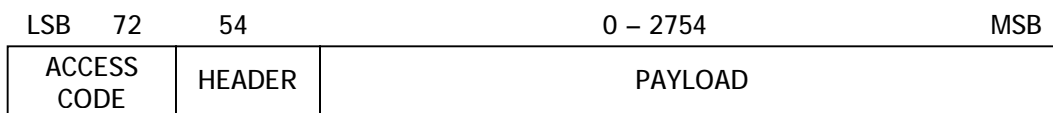


Fig. 4.6.2: Standard packet format

The access code and header size have fixed length (72 bits and 54 bits respectively). The payload can range from 0 to maximum of 2754 bits. Packets may be very short and contain only access code, they can also contain access code and header, and there are full-sized packets that contain access code, header and payload.

For detailed specification of standard packet format, please refer to the Specification of the

Java Bluetooth stack	Version: 0.5
Developer's Guide	Date: 2004-01-10

Bluetooth System (Books [1]).

4.6.2 Link Control (LC)

Link Control channel is mapped onto the packet header (see Fig. 4.6.2.). This channel carries low level link control information like ARQ (Automatic Repeat reQuest), flow control, and payload characterization. The LC channel is carried in every packet except in the ID packet that has no packet header.

4.6.3 Link Manager (LM)

Link Manager Protocol (LMP) messages are used for link set-up, security and control. They are transferred in the payload instead of L2CAP and are distinguished by the L_CH code 11 (that indicates LM channel). LM channel typically uses protected DM (Data Medium rate) packets.

The Link Manager control channel carries control information (LMP messages) exchanged between the link managers of the master and slave(s). Those messages have higher priority than user data, so if the LM needs to send a message, it will not be delayed by the L2CAP traffic. Delay may only occur if there is many retransmissions of individual baseband packets.

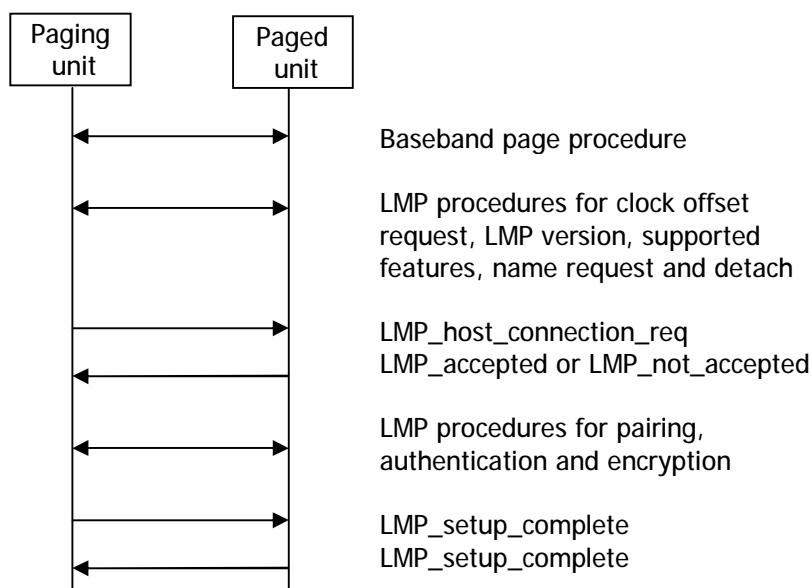


Fig. 4.6.3: Connection establishment

Figure 4.6.3. shows connection establishment between two devices – paging device and paged device. When the paging device wishes to create a connection involving layers above LM, it sends LMP_host_connection_req. Paged device upon receiving connection request can accept or reject that connection. In case it wants to accept the connection, it sends response LMP_accepted to the paging device. After LMP_accepted response, LMP security procedures (pairing, authentication and encryption) can be invoked. After both devices have sent LMP_setup_complete the first packet on a logical channel different from LMP can then be transmitted.

In some cases there is need for slave to request master/slave switch. That request is sent by LMP_slot_offset or LMP_switch_req after the LMP_host_connection_req is received. When the master/slave switch has been successfully completed, the old slave will reply with LMP_accepted, but with the transaction ID set to 0.

4.6.4 Host Controller Interface (HCI)

HCI provides a uniform interface method of accessing the Bluetooth hardware capabilities. HCI commands are implemented in HCI firmware by accessing baseband commands, link manager commands, hardware status registers, control registers, and event registers. In some cases there is

Java Bluetooth stack	Version: 0.5
Developer's Guide	Date: 2004-01-10

also Host Controller Transport Layer as an intermediate layer for communication between HCI firmware and host HCI driver.

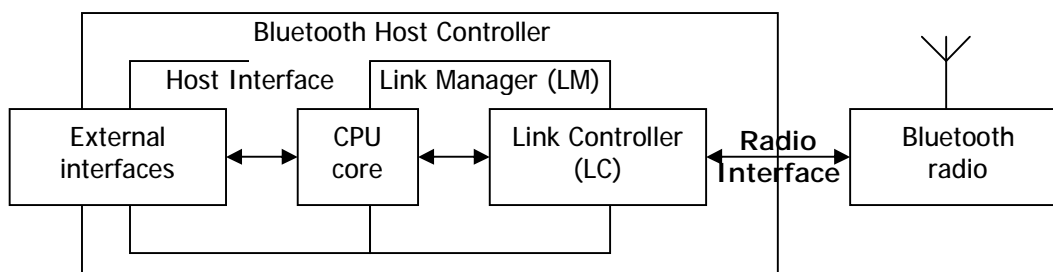


Fig. 4.6.4: Bluetooth Hardware architecture overview

In order to avoid filling up the Host Controller data buffers with ACL (Asynchronous Connection-Less) data destined for a remote device that is not responding, HCI uses flow control in the direction from the Host to the Host Controller.

4.6.5 Transport Interface

Transport Interface layer is a transport layer between the Host Controller driver and the Host Controller. Some of the examples of transport interfaces are PCMCIA, UART, USB, etc.

This layer is used because the Host Controller driver should not care whether it is running over USB or PC-Card. This allows the Host Controller Interface or the Host Controller to be upgraded without affecting the transport layer.

5. Installation guide – software

Before you can start using Java Bluetooth Stack, there are few things you should know. First, since this project is a library to be integrated into Java applications that use Bluetooth, it's distributed (for now) in the form of Eclipse project. The [Eclipse Platform](#) is an open extensible IDE for anything, but the most evolved part of the platform are Java Development Tools (JDT). Eclipse platform along with JDT offers currently the most usable general-purpose Java IDE.

With this in mind, common procedures (importing projects, creating run configurations, configuring libraries) are described later on in this document, in order to help the novice users through their first steps in the Eclipse.

Next very important thing about Java Bluetooth Stack is hardware. JBS is a hardware dependent project, and very little can be done without it. Currently JBS supports only H4 (UART) HCI Transport Layer (for some supported devices see Hardware installation guide) through the Java commAPI, so, in order to get the test configuration going, Java commAPI needs to be installed correctly (pay special attention to this, as this can be the source of many problems afterwards).

The third and the final thing that needs to be installed is some kind of environment supporting MIDP profile. Since JSR-82 specification is targeted at MIDP devices, JBS needs some J2ME classes to function completely (this goes mostly for javax.microedition.rms package, which is used only to store information about trusted devices). So you can either install Sun's Wireless Toolkit (and put midpapi.zip in project references) or use ME4SE (MicroEdition4StandardEdition) library (and put me4se.jar and png.jar in project references). ME4SE is a library that enables a user to run a midlet (applet written for J2ME) on a J2SE platform (see [link](#) for details).

5.1 Installing Eclipse Platform and Java commAPI and configuring environment variables for commAPI

Java Bluetooth stack	Version: 0.5
Developer's Guide	Date: 2004-01-10

5.1.1 Eclipse Platform installation

For Eclipse install instructions, and any issues regarding Eclipse platform, please refer to the Eclipse project FAQ (<http://www.eclipse.org/eclipse/faq/eclipse-faq.html>).

5.1.2 Sun Wireless ToolKit installation

Installing Sun WTK (Wireless ToolKit) is pretty straightforward, just download and run the executable installation file. Follow the on-screen instructions, and note the directory in which the WTK will be installed, you'll need it later on when configuring project references (midpapi.zip which needs to be in project references, is usually in the WTK_install_dir\lib\ subdirectory).

5.1.3 Installing commAPI

- Unzip the file javacomm20-win32.zip. This will produce a hierarchy with a top-level directory commAPI

The examples in this document assume that you have unzipped the javacomm20-win32.zip file in your C: partition and your JDK installation is in C:\jdk1.4.2_02.

If you have installed JDK in an other location or unzipped javacomm20-win32.zip in an other location modify the example commands appropriately.

- Copy win32com.dll to your <JDK>\bin and to your <JDK>\jre\bin directory
C:\>copy c:\commapi\win32com.dll to c:\jdk1.4.2_02\bin
C:\>copy c:\commapi\win32com.dll to c:\jdk1.4.2_02\jre\bin
- Copy comm.jar to your <JDK>\lib and to your <JDK>\jre\lib directory
C:\>copy c:\commapi\comm.jar c:\jdk1.4.2_02\lib
C:\>copy c:\commapi\comm.jar c:\jdk1.4.2_02\jre\lib
- Copy javax.comm.properties to your <JDK>\lib and to your <JDK>\jre\lib directory
C:\>copy c:\commapi\javax.comm.properties c:\jdk1.4.2_02\lib
C:\>copy c:\commapi\javax.comm.properties c:\jdk1.4.2_02\jre\lib

The javax.comm.properties file must be installed. If it is not, no ports will be found by the system.

Set your path variables like this (this is probably the most important part):

[Beware, this jars are the ones installed by J2SDK 1.4.2_02, if you're configuring any other J2SDK version with this commapi please check jar names and change them accordingly.]

```
JAVA_HOME=<JDK>\
CLASSPATH=%JAVA_HOME%\lib\comm.jar;%JAVA_HOME%\jre\lib\comm.jar;%JAVA_HOME%\jre\lib\rt.jar;%JAVA_HOME%\jre\lib\jsse.jar;%JAVA_HOME%\jre\lib\charsets.jar;%JAVA_HOME%\jre\lib\jce.jar;%JAVA_HOME%\jre\lib\charsets.jar;%JAVA_HOME%\jre\lib\plugin.jar;%JAVA_HOME%\jre\lib\sunrsasign.jar;%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\lib\htmlconverter.jar;%JAVA_HOME%\lib\tools.jar
```

PATH=%JAVA_HOME%\bin;%PATH% (please note that "%JAVA_HOME%\bin;" must be at the very beginning of the path variable)

Restart your machine, and to test your newly installed/configured Java commAPI run Blackbox example from commAPI installation directory:

- Go to the samples\BlackBox subdirectory of the directory you unpacked commapi:

```
C:\>cd commapi\samples\BlackBox
```

Java Bluetooth stack	Version: 0.5
Developer's Guide	Date: 2004-01-10

- Start the BlackBox sample giving the path to the comm.jar as the classpath parameter:

```
C:\commapi\samples\BlackBox>java -cp .;c:\j2sdk1.4.2_02\lib\comm.jar BlackBox
```

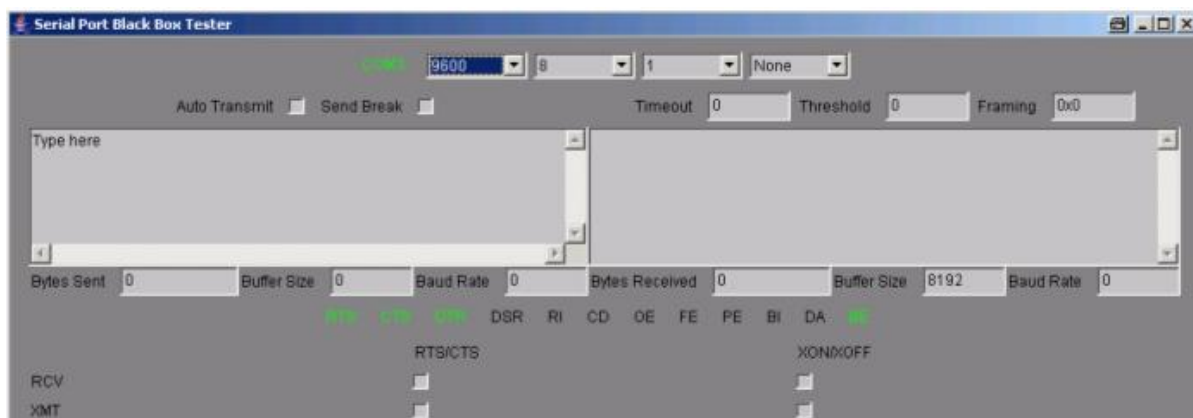


Fig. 5.1.1: Blackbox example main screen

5.1.4 Importing project into workspace

After you have got commapi to work, download the project zip and unpack it into directory of your choice (just make sure you remember which one is it). Start Eclipse Platform and follow the instructions for importing existing projects into workspace and import the JBS into your workspace.

If you imported jbs project correctly, package explorer view should look like on Fig. 7.1.2.

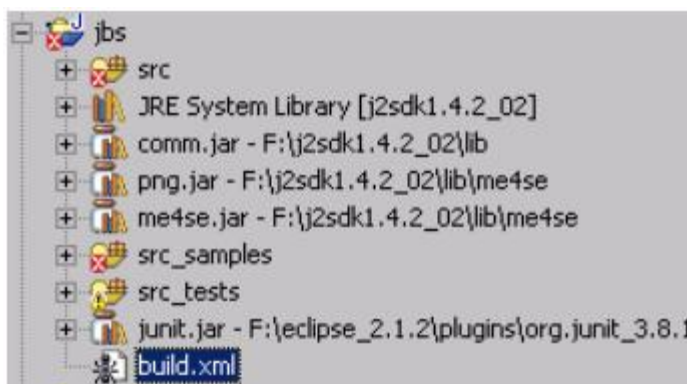


Fig. 5.1.2: Package explorer view after importing JBS project

When you're done with importing, follow the instructions for repairing project references (instructions in Appendix C) – they have to contain comm.jar, junit.jar (found in the eclipse_install_directory\plugins\org.junit_X.X.X, X.X.X denotes version), and midpapi.zip (or combination of me4se.jar and png.jar instead of midpapi.zip).

5.1.5 Running samples and JUnit tests

Once the project is imported and all required libraries in place, you can start exploring samples (but make sure you have installed some Bluetooth hardware with H4 (UART) Transport Layer – refer to the hardware install instructions for more detail).

For creating new Run Configuration please refer to the Appendix B, and for creating new JUnit Run Configuration refer to Appendix D.

6. Installation guide – hardware

Currently, Java Bluetooth Stack supports only Bluetooth devices that use H4 (UART) Host transport layer to communicate with Hosts (typically PCs). Support for those devices is accomplished

Java Bluetooth stack	Version: 0.5
Developer's Guide	Date: 2004-01-10

through Java Communication API (for instructions on installing the commAPI, please refer to the software installation guide (chapter 7 of this documentation)).

In order to use the stack, you will need to have at least two Bluetooth devices with UART transport layer enabled. These devices are typically Bluetooth serial modules such as ROK101008 from Ericsson Application Toolkit (other companies that manufacture serial modules are Brainboxes, SMART Modular Technologies, etc.).

Some PC-Cards (PCMCIA) might work, as long as they are recognized by the OS as COM ports. This heavily depends on the OS you're using, i.e. Xircom CBT PC-Card is installed by Windows 2000 as virtual COM port, while Windows XP configure it differently, rendering it unusable with JBS.



Fig. 6.1: Xircom CBT PC-Card installed as a virtual COM port

Important notice: following instructions apply only to Bluetooth hardware noted here, which has been tested exhaustively. They might work for other similar hardware, but we cannot take ANY responsibility for any damage caused by improper Bluetooth hardware installation.

6.1 Xircom CBT

General notes

You should start by downloading the driver from the [Xircom](#) site. After you download the driver, run it, follow the instructions and install it into a temporary directory (but make sure you remember which one).

- a. Xircom CreditCard Bluetooth Adapter install instructions for notebook computers (computers with build-in PC-Card slot)

Insert the card into the PC-Card slot. Windows should detect and ask for drivers. Point them to the location where you have previously unpacked (installed) downloaded drivers, and wait until all the profiles (virtual COM ports) are installed.

[BUG] Even if the "Restart computer now" message box appear, please wait until you're absolutely certain that the driver installation has ended (i.e. there's no disk activity for a period of 3 minutes), and then and only then restart your computer. This is a known issue and is documented in the driver installation notes.

- b. Xircom CreditCard Bluetooth Adapter install instructions for regular PCs

If you need to install any PC-Card to a standard PC, you need to have a PC-Card adapter. These instructions refer to the Ricoh PC-Card adapter, but the procedure should work with any similar PC-Card Adapter.

First, install the drivers from the installation diskette. Shutdown your computer and install the PC-Card adapter into a free PCI slot. Turn your computer on, Windows should recognize newly installed adapter and finish the installation.

Restart your computer, and when Windows finish booting insert Xircom CBT into Ricoh PC-Card adapter. Windows will prompt you for drivers. Follow the instructions given above (installation for notebook computers).




Fig. 6.2: Xircom CBT PC-Card

Java Bluetooth stack	Version: 0.5
Developer's Guide	Date: 2004-01-10

c. Post install notes

The last thing to do in order to use Xircom CBT with JBS, is to open Device Manager and, under Ports (COM & LPT) tree node you should see something like "Xircom CreditCard Bluetooth Adapter (COMX)" (see picture above). You should note the COM port number, as you will use it in initializing the BCC (Bluetooth Control Center) in every JBS application.

Before you run any of the samples included with the stack using COM port number assigned to Xircom CBT, you must disable the Xircom Bluetooth environment application (in the tray menu right click the  icon and uncheck "Radio On", or choose "Exit" from the pop-up menu), otherwise you will get Port in use Exception.

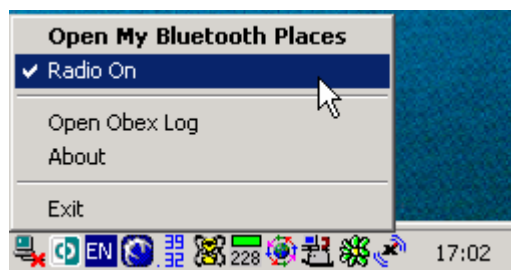


Fig. 6.3: BT tray icon

6.2 Generic Bluetooth serial module install instructions

There are two main things you should provide to a serial module – power and connection to serial port. As they consume a significant amount of current (as much as 150 mA during inquiry), they need a separate power source (serial port can only provide about 25 mW of power). If serial module supports USB besides UART, you can connect it to the PC with the USB cable provided with the module. If it doesn't, you must consult the module documentation to find out what voltage level the module requires. If that's 5V DC or 12 V DC, you can use standard PC AT power supply.

[Caution: messing around with 220 V AC can be very dangerous if you don't know exactly what are you doing!]

a. Ericsson ROK 101008 install instructions

In the package with the module you should have received crossover serial cable (female DB9 <-> female DB9 connectors), serial cable adapter (for board header connector) and a USB cable. Connect the serial cable to a free COM port, on the other end connect the serial cable adapter, and connect the adapter to the module board. Connect the module with the USB cable to the PC to provide power.

Module is now ready for use with JBS; specify the COM port number in the BCC (Bluetooth Control Center) initialization code.



Fig. 6.4: Ericsson ROK 101008 BT module

b. ROK104001 module installation instructions

In the package with the module you should have received straight serial cable (female DB9 <-> male DB9 connectors). Connect the serial cable to a free COM port and to the module board. Connect the module to a standard PC power supply (either from a separate power supply, or from your computers'), or to any other power source that provides 6 – 20 V DC using the DC plug connector.

Module is now ready for use with JBS; specify the COM port number in the BCC (Bluetooth Control Center) initialization code.

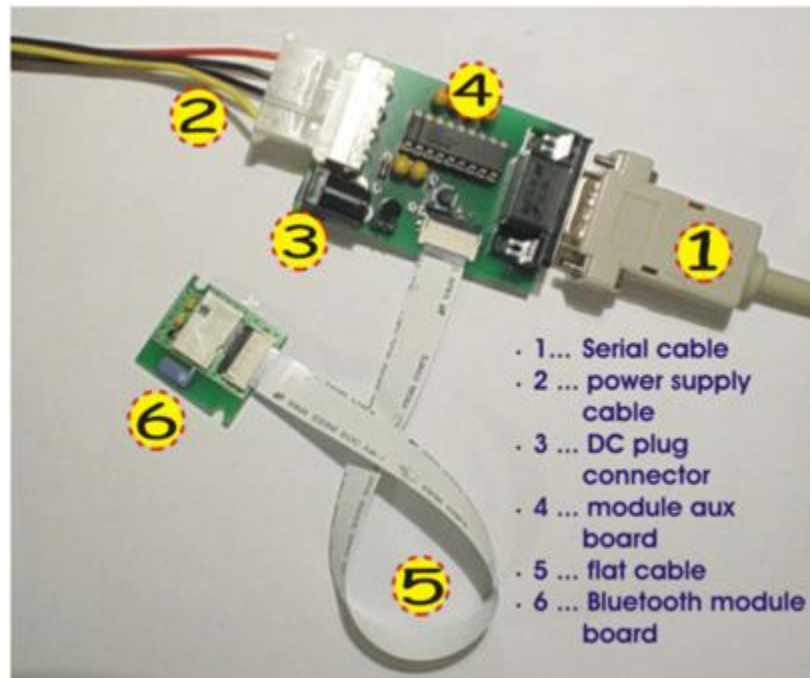


Fig. 6.5: ROK 104001 Module installation

Java Bluetooth stack	Version: 0.5
Developer's Guide	Date: 2004-01-10

Appendix A: Creating Eclipse Runtime configurations

The most practical way to create Eclipse Runtime configuration is to just let the Eclipse handle all the work:

- open the java file for which is intended to create runtime configuration into the editor of the Eclipse platform
- choose **Run -> Run ...** from the menu bar, and in the window that is opened right click on the **Java Application** in the left pane. Select the **New** option (Fig. A.1)

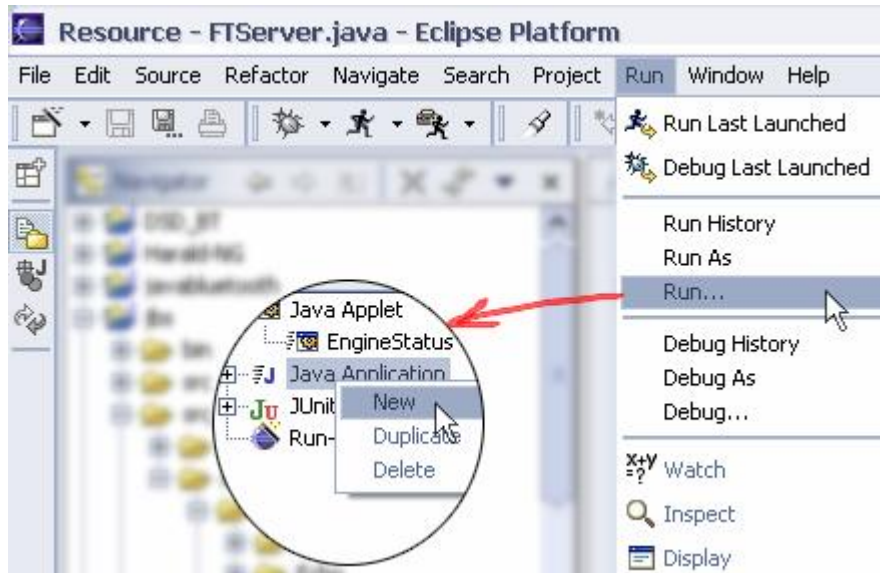


Fig. A.1: Creating the new Runtime Configuration

- After that we get an window (similar to window on Fig. C.7) with parameters set to the corresponding values – those values can be changed if needed
- Click on the **Run** button to run newly creted run configuration

Java Bluetooth stack	Version: 0.5
Developer's Guide	Date: 2004-01-10

Appendix B: Checking and configuring project references

Projects can contain more of external libraries, so it can easily happen that some error occur in the link to those libraries. The most usual error is wrong path to the external library – this can happen if the project is transferred onto another computer, where the paths to the libraries are not the same as on the computer where the libraries are linked with project. Other case in which the path error can occur is if someone accidentally deletes or uninstall some of the libraries.

No matter with the error is, we can always check (and correct those errors) by right-clicking on the projects root in the Navigator view and selecting Properties in the drop-down menu. In the opened window select the **Java Build Path** option in the left pane, and the **Libraries** on the tab in the right pane (Fig. 6.1.13.)

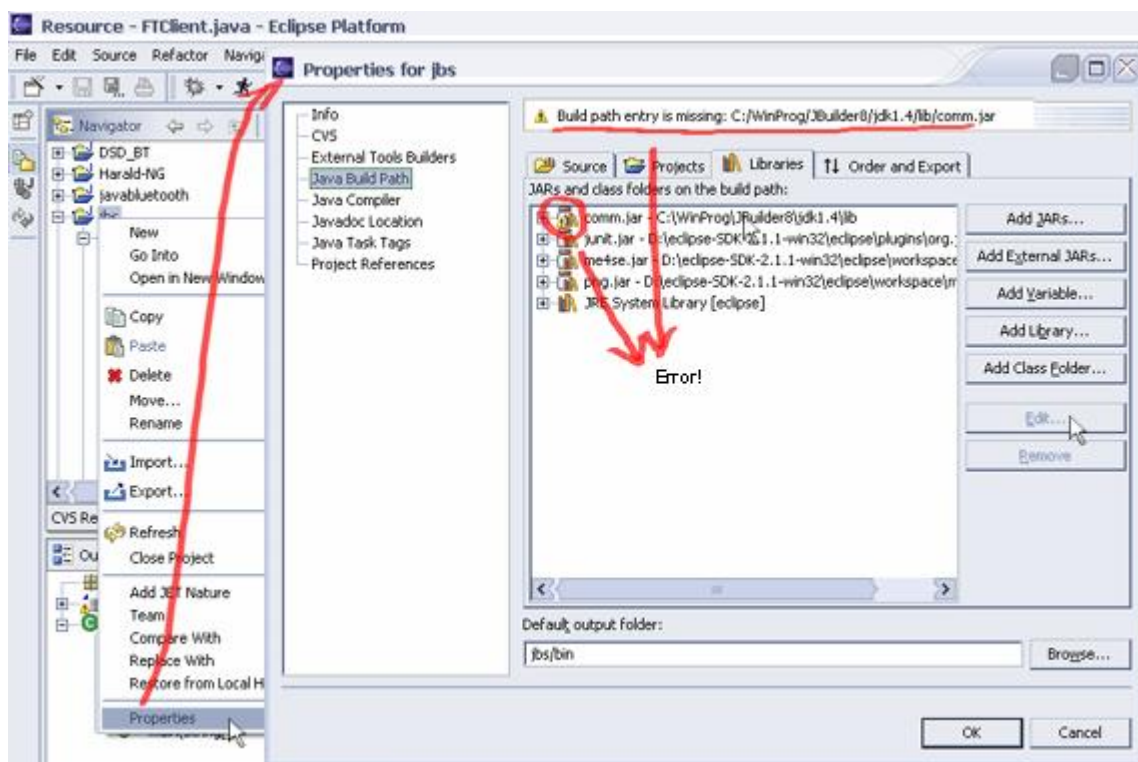


Fig. B.1: Project's properties – Java Build Path

As shown on Figure 6.1.13. there is an error with the comm.jar library path – error message is shown in the bar above. To correct this error, select the library line path (where is the yellow exclamation sign) and click on **Edit** button on the right side of the Properties window. In here is possible to manually find the missing file.

Common locations for libraries used by Java Bluetooth Stack:

- comm.jar <JAVA_HOME>\lib
- junit.jar <ECLIPSE_HOME>\plugins\org.junit_X.X.X (in this case X.X.X is 3.8.1)
- midpapi.zip <WTK_HOME>\lib

Java Bluetooth stack	Version: 0.5
Developer's Guide	Date: 2004-01-10

Appendix C: Eclipse JUnit support

Writing JUnit tests:

Eclipse installation includes JUnit in the `org.junit` plug-in. Since Eclipse version 2.0 this plug-in is part of the build.

Before any test can be written or run, `junit.jar` library must be added to the build class-path:

- right-click on the project's root directory in the Navigator pane and select **Properties**

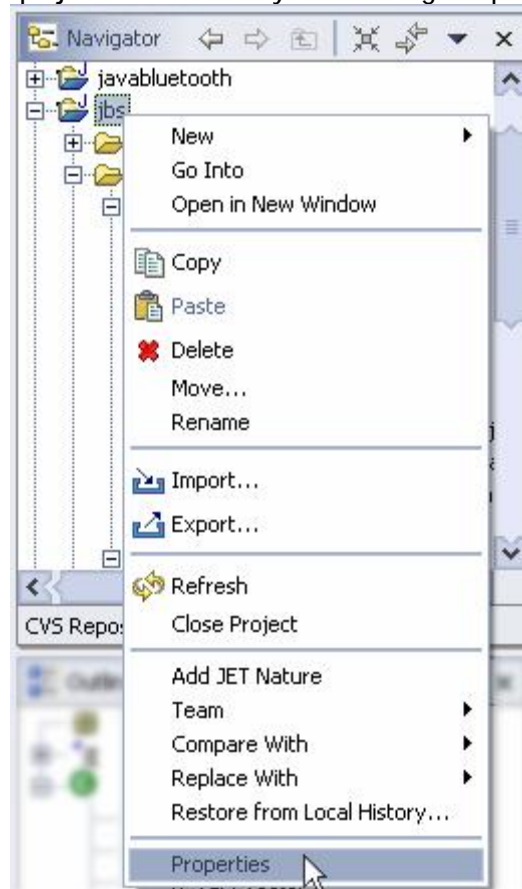


Fig. C.1: Opening project's properties

- In the opened window select **Java Build Path** in the left pane, and the **Libraries** tab-pane on the right side of the window. New external libraries are added by **Add External JARs** button (Fig. C.2)
- From the JAR selection window (Fig. C.2) in the **Look-in** path select path to the `<ECLIPSE_HOME>\plugins\org.junit_3.8.1` where is JUnit plug-in located, select the `junit.jar` file and open it

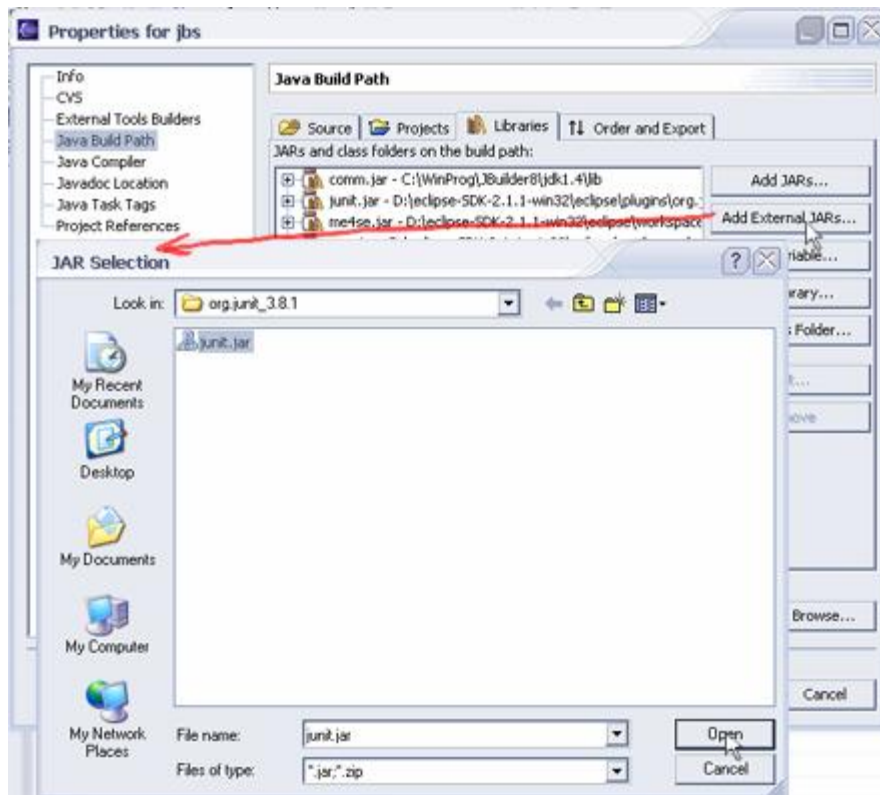


Fig. C.2: Adding *junit.jar* to the build class-path

- Click on the OK button in project's properties window to confirm properties of the project

Now when the libraries are set, it is possible to write some JUnit tests. All the JUnit tests must be the subclasses of the *TestCase* class.

The easiest way of writing JUnit tests is by using wizard:

- open the New wizard (**File -> New -> Other ...**), select **Java -> JUnit** in the left pane, and the *TestCase* in the right pane, and click **Next** (Fig. C.3)

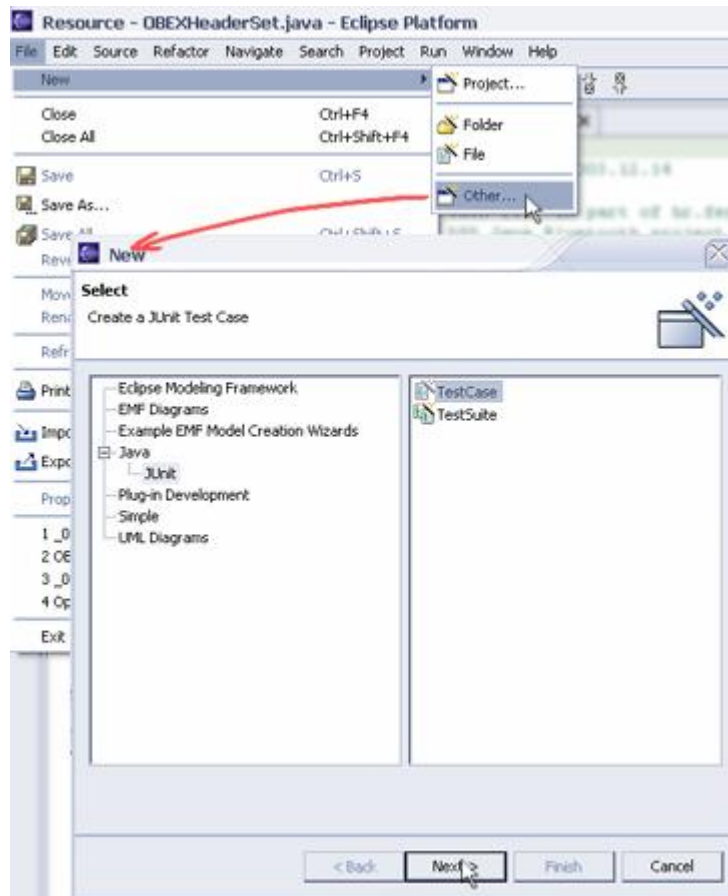


Fig. C.3: Using wizard for writing new JUnit TestCase or TestSuite

- Enter *TestFailure* as the name of your test class



Fig. C.4: Creating a new JUnit TestCase

- Click **Finish** to create the test class

Java Bluetooth stack	Version: 0.5
Developer's Guide	Date: 2004-01-10

- Add the testing procedure to the test class

Running JUnit tests

- open the test in the editor view (in Eclipse)
- activate the **Run** drop-down menu in the toolbar and select **Run as -> JUnit Test** (alternative is to create a new JUnit Run Configuration – see next section for details)
- The JUnit test window now shows you the test run progress and status

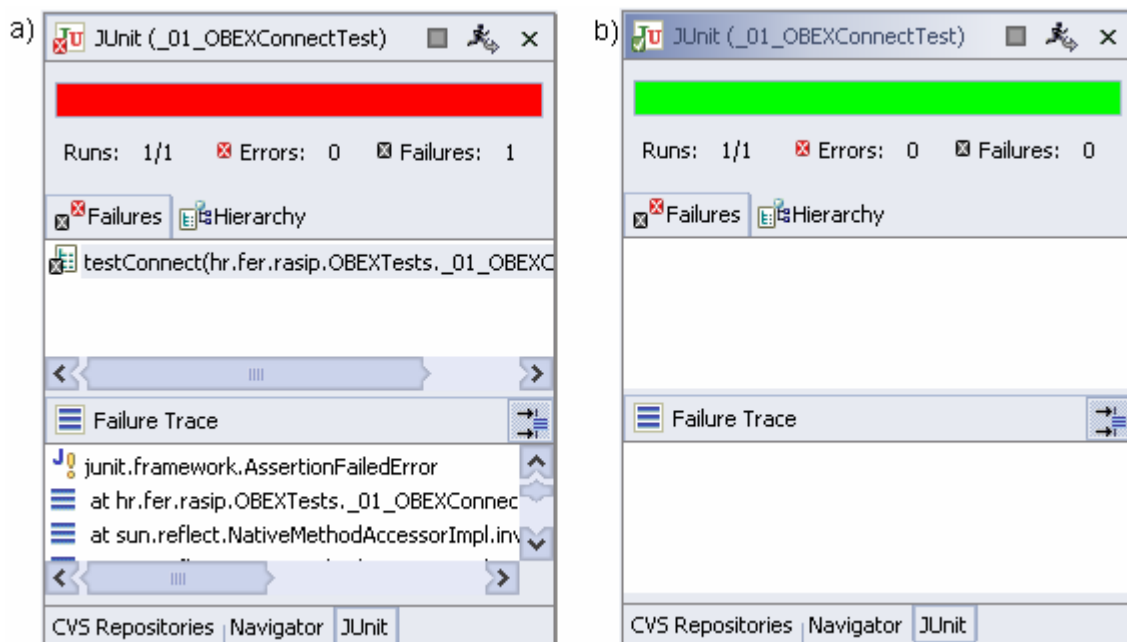





Fig. C.5: JUnit test run status: a) test failed, b) test successfully completed

Fig. C.5 shows test run status – in case a), test failed (icon  is shown in the upper-left corner of JUnit view). Description of why test failed is provided in the "Failure Trace" pane (in this case, assert test method failed) – you can click on any of the lines in this pane to go to the specific line in code to see why test failed. In case b), test has successfully finished (icon  is shown).

The JUnit view has two tabs: one shows you a list of failures and the other shows you the full test suite as a tree.

Test can be re-run either by the method previously described (Run as -> JUnit Test), by JUnit Run Configuration, or by Re-Run button () in the JUnit view.

It is possible to Run JUnit test on one or more tests at once:

- Running all tests at once: select a package or source folder and run all included tests by **Run as -> JUnit test**
- Run a single test method: select a test method in the Outline or Package Explorer and run test by **Run as -> JUnit test**
- Re-run a single test: select a test in the JUnit view and execute **Rerun** from the context menu

Debugging the test failure:

- double click the failure entry from the stack trace in the JUnit view to open the corresponding file in the editor

Java Bluetooth stack	Version: 0.5
Developer's Guide	Date: 2004-01-10

- set the breakpoint at the beginning of the test method
- select the test case and execute **Debug as -> JUnit Test** from the **Debug** drop-down menu

Creating a new JUnit Run Configuration

The most practical way to run JUnit tests is to just let the Eclipse handle the testing:

- open the JUnit test file in the editor of the Eclipse platform
- choose **Run -> Run** from menu bar, and in the window that is opened right-click on the JUnit configuration and select **New** (Fig. C.6)

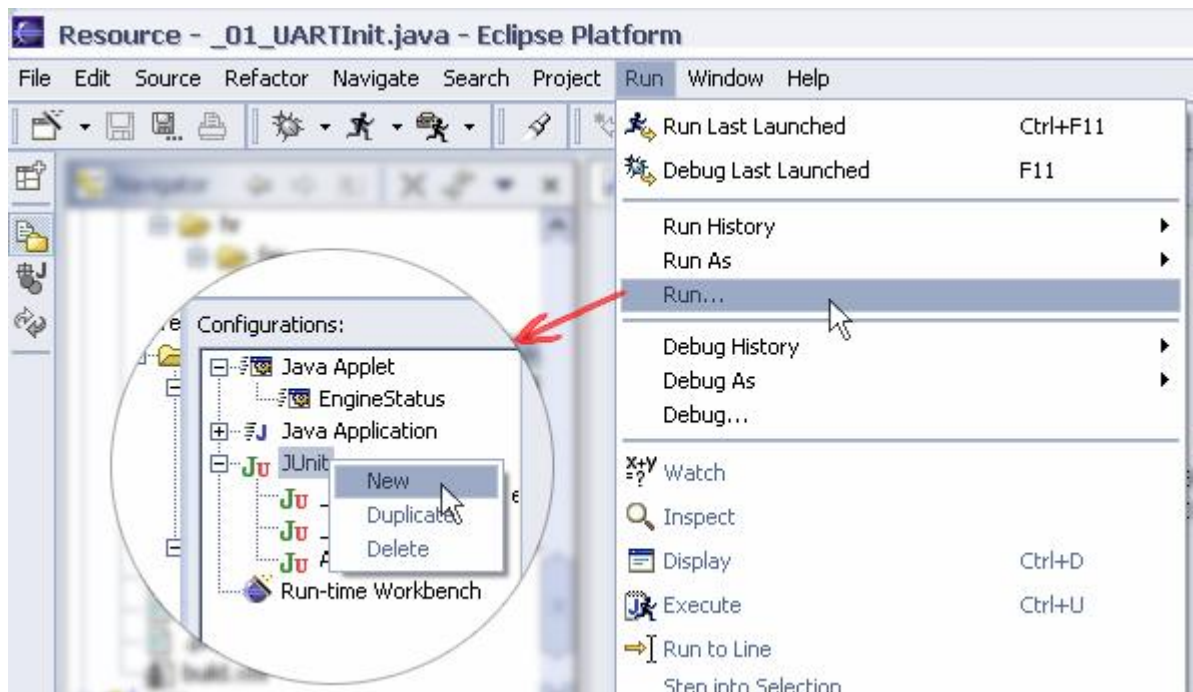


Fig. C.6: Creating new JUnit run configuration

JUnit run configuration is now automatically created. You may now click on the **Run** button to start the test, or view/change some parameters if needed (Fig. C.7)

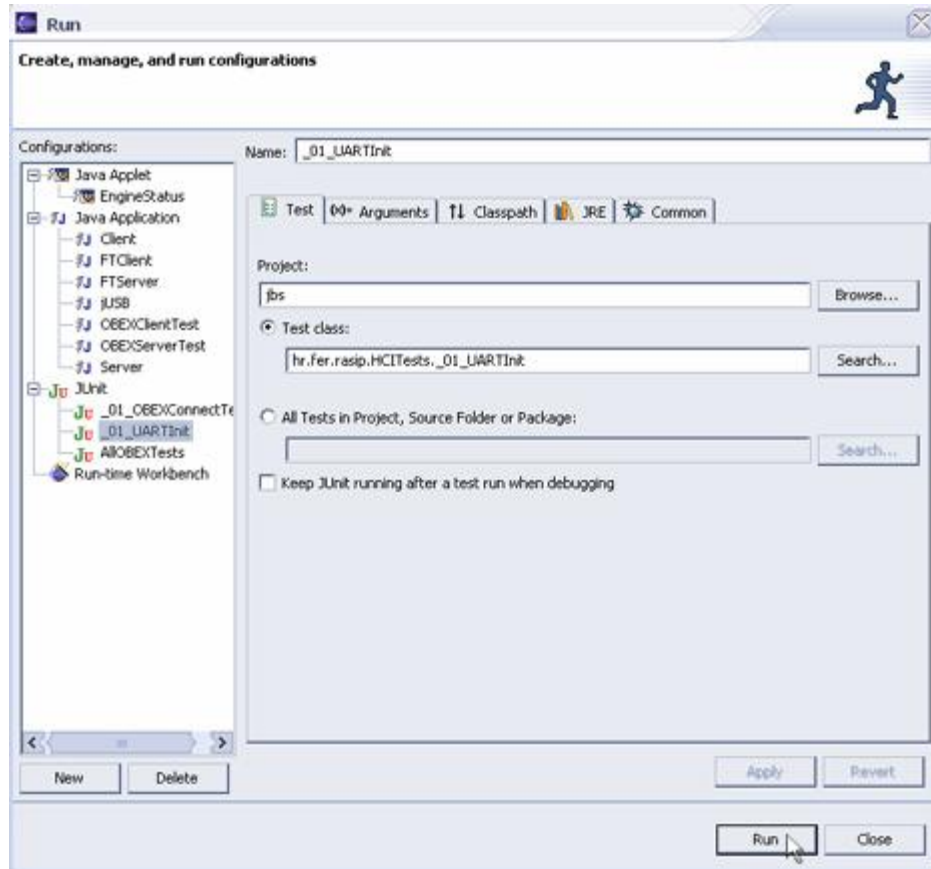


Fig. C.7: JUnit Run test created and ready to run

Creating a JUnit test suite

If someone wants to run all (or several) tests at once, there should be some automated procedure for that, because it can be pretty annoying to run tests one by one. For that purpose, JUnit has implemented the test suite.

The easiest way of creating a JUnit test suite is by using wizard:

- open the New wizard (File -> New -> Other ...)
- Select Java -> JUnit in the left pane and TestSuite in the right pane and click Next (Fig. C.8)
- enter a name for your test suite class (the convention is to use "AllTests" or similar)

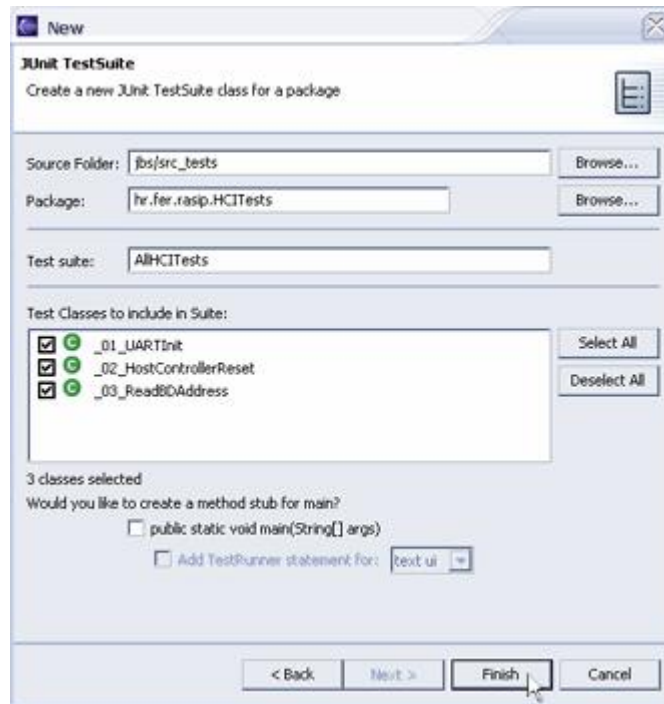


Fig. C.8: Creating a new JUnit TestSuite

- select the classes that should be included in the suite

You can add or remove test classes from the test suite:

- manually, by editing the test suite file
- by re-running the wizard and selecting the new set of test classes

NOTE: the wizard put markers `//&JUnit-BEGIN&` and `//&JUnit-END&`, into the created Test Suite class, which allows the wizard to update existing test suite classes. Editing code between this two markers is not recommended.

Java Bluetooth stack	Version: 0.5
Developer's Guide	Date: 2004-01-10

Appendix D: Eclipse & CVS

When your CVS server supports only SSHv2 (that's a good thing actually) and Eclipse has a built-in client only for an SSHv1 (*extssh* connection method), there are two options:

- Download [CVS-SSH2 Plug-in for Eclipse](#) (an [Eclipse](#) plug-in that allows CVS access on SSH2 session)
- Use *ext* connection method and an external ssh util (like *plink* from creators of *putty*) – this method involves generating RSA key pair and storing the public key on the server – it's pretty complex...

I suggest the first method – download the plug-in, kill eclipse if it's running, unzip the plug-in into the `eclipse\plugins`, and start Eclipse again.

Main CVS operations:

- Creating a CVS repository location – every active developer in the team will have to do this eventually
- Sharing a new project using CVS – this has to be done just once, by whoever has started coding or created Eclipse project – let's call him/her the initiator from now on
- Checking out a project from a CVS repository – every developer in the team (except the initiator must do this to get a local copy of the project)
- Updating code – done when needed
- Committing code – done after some changes in code
- Synchronizing with a repository – local copy is compared with repository copy

[Following chapters have been taken out of Eclipse documentation and are somewhat shortened and fulfilled with pictures]

Creating a CVS repository location

1. Open the CVS Repositories view by selecting **Window -> Show View -> Other...** on the main menu bar and then selecting **CVS -> CVS Repositories** from the Show View dialog. Alternatively, the CVS Repositories view is also shown in the CVS Repository Exploring perspective.
2. From the context menu of the CVS Repositories View, select **New -> Repository Location**. The Add CVS Repository wizard opens.
3. Enter the information required to identify and connect to the repository location:
 1. In the **Host** field, type the address of the host.
 2. In the **Repository path** field, type the path to the repository on the host (in our case something like `/var/CVSRROOT/<project_name>`)
 3. In the **User** field, type the user name under which you want to connect to the repository.
 4. In the **Password** field, type the password for the above user name.
 5. From the **Connection Type** list, select the authentication protocol of the CVS server. There are three connection methods that come with the Eclipse CVS client:
 - **pserver** - a CVS specific connection method (for read only access).
 - **extssh** - an SSH 1.0 client included with Eclipse
 - **ext** - the CVS ext connection method that uses an external tool such as SSH to connect to the repository. The tool used by ext is configured in the **Team -> CVS -> EXT Connection Method** preference page.
 - **extssh2** – a method added by CVS-SSH2 Plug-in
 6. If the host uses a custom port, enable **Use Port** and enter the port number.
4. (Optional) Select **Validate Connection on Finish** if you want to authenticate the specified user to the specified host when you close this wizard. (If you do not select this option, the user name will be authenticated later, when you try to access the contents of the repository.)

Java Bluetooth stack	Version: 0.5
Developer's Guide	Date: 2004-01-10

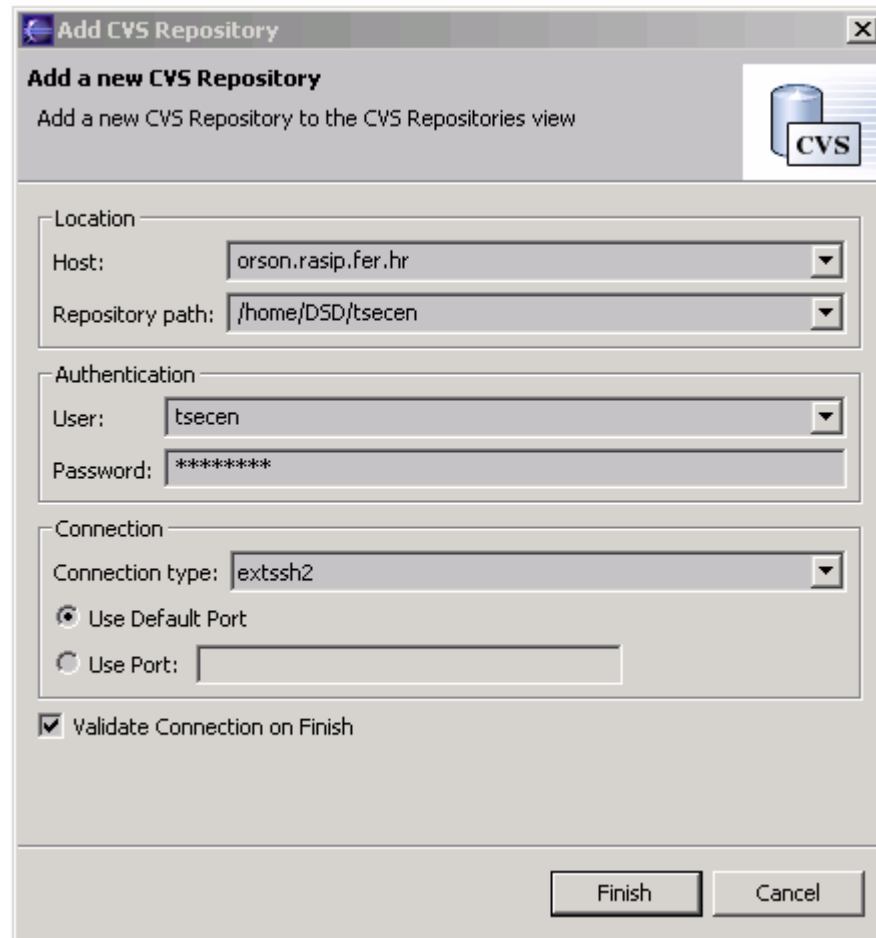


Fig. D.1: Adding the CVS repository directory

5. Click Finish. The repository location is created.

Sharing a new project using CVS

There are several scenarios that can occur when sharing a project using CVS. The most common scenario is sharing a new project using CVS when the project does not already exist remotely.

To share a new project using CVS:

1. In the Navigator view, select the project to be shared
2. Select **Team -> Share Project...** from the project's pop-up menu. The Share Project wizard opens.

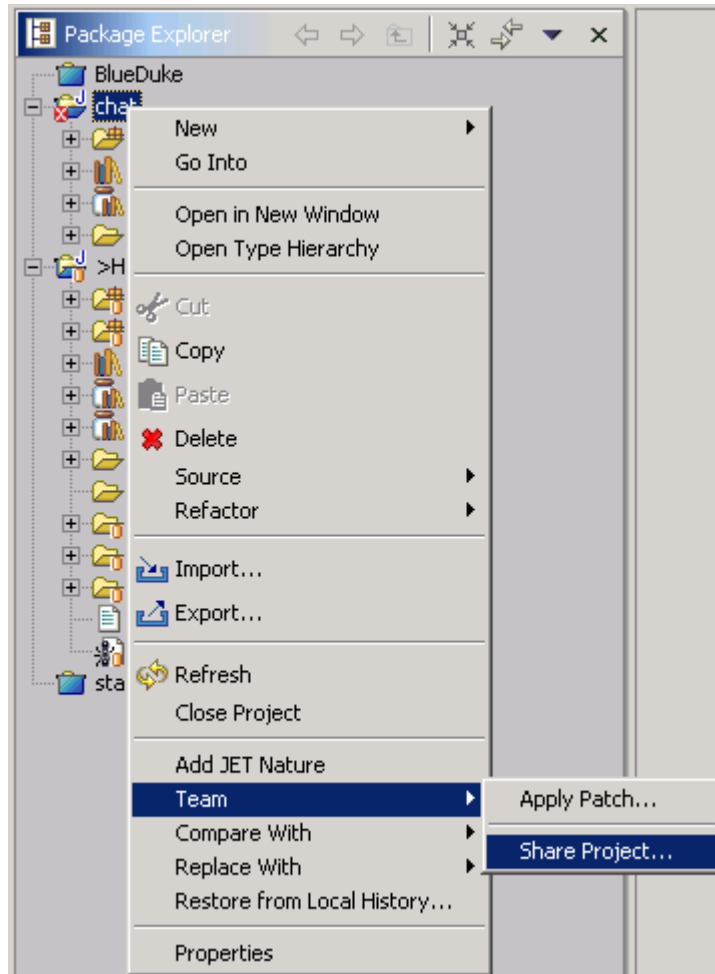


Fig. D.2: Sharing a project using CVS

3. From the list of available repository providers, choose **CVS** and click **Next** to go to the next page. (**Note**: If there is only one repository provider available, the first page will be skipped and next page will be displayed automatically.)
4. Select the target repository from the list of known repositories or, if the target repository is not in this list, choose to create a new repository location and click **Next**.
5. If entering a new repository location, enter the repository information and click **Next** when completed. (**Note**: this page is the same format as the [Creating a CVS repository location](#) wizard.)

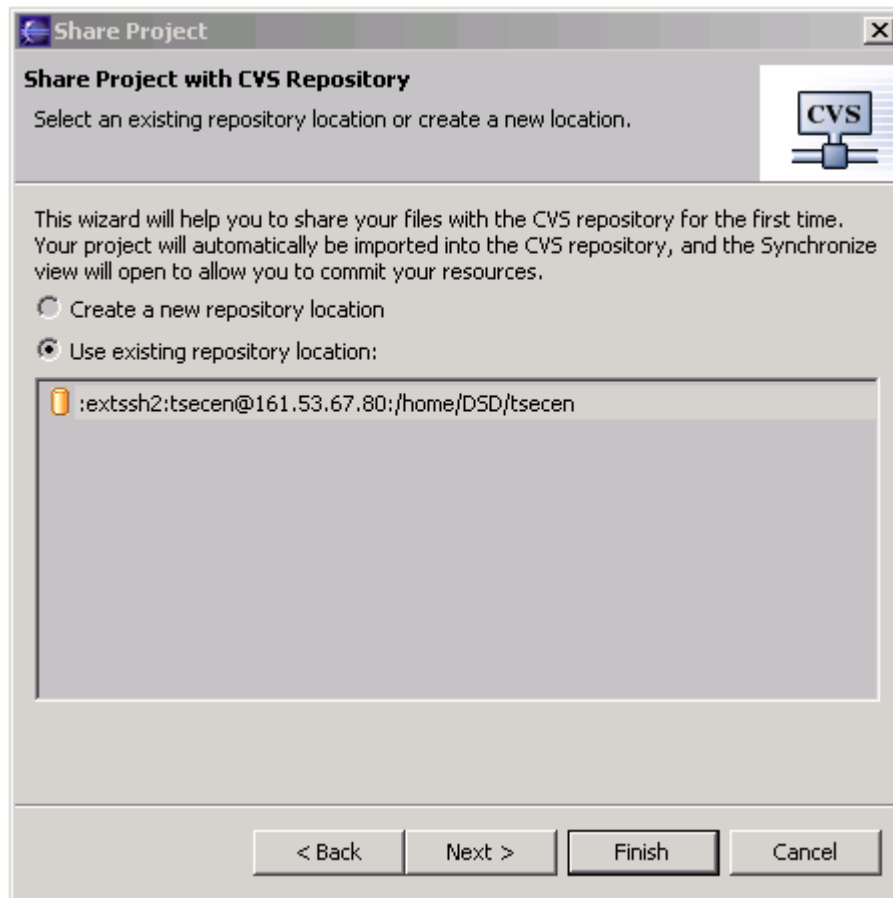


Fig. D.3: Sharing an existing CVS repository

6. (Optional) Either choose to use the name of the local project as the remote project name or enter another name.
7. Click **Finish** to share the project with the repository. The project folder will be created remotely and the Synchronize view will open and allow the committing of the project's resources. (**Note:** If the project already exists remotely, the Synchronize view will show conflicts on any files that exist both locally and remotely.)

Checking out a project from a CVS repository

To check out a project from a CVS repository to the Workbench:

1. Switch to the CVS Repository Exploring perspective or add the CVS Repositories view to the current perspective.
2. In the CVS Repositories view, expand the repository location.
3. Expand **HEAD** and select the folders that you want to add as projects to the Workbench. If you are looking for a folder in a particular version:
 - a. Expand the **Versions** category and find the folder name that you want to add to the Workbench.
 - b. Expand the folder to reveal its versions.

If you are looking for the latest folder in a branch:

 - c. Expand the **Branches** category.
 - d. Expand the branch that holds the folder that you want to add to the Workbench.
4. From the pop-up menu for the selected folders, select one of the following:

Java Bluetooth stack	Version: 0.5
Developer's Guide	Date: 2004-01-10

- a. **Check Out as Project** to check out each of the selected folders as a project in the local workspace with the same name as the folder in the repository.
- b. **Check Out As...** to check out the selected folders into a custom configured project in the local workspace. *Note:* When multiple folders are selected, this operation only allows the specification of a custom parent location for the new projects.

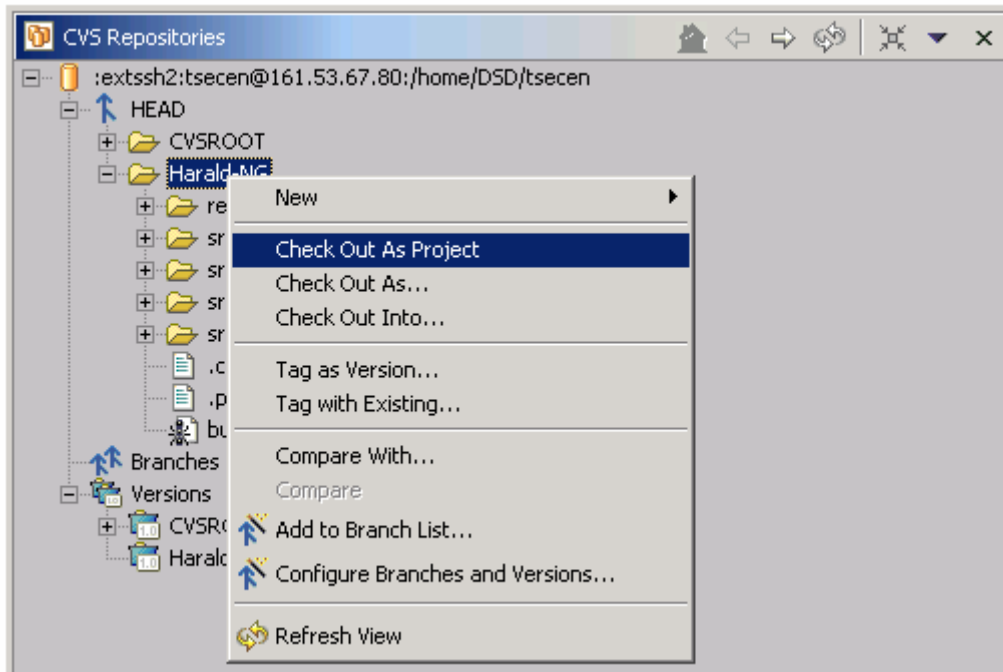


Fig. D.4: Creating a local project from CVS repository

5. If **Check Out As...** was chosen on a single project, one of two possible dialogs is displayed depending on whether the folder in the repository contains a `.project` file or not.
 - a. If there is a `.project` file, the dialog will accept a custom project name and location.
 - b. Otherwise, the dialog will be the New Project wizard that allows full customization of the project (e.g. Java project).

Tip: Any folder, including non-root folders, can be checked out from a CVS repository.

Updating

While you are working on a project in the Workbench, other members of your team may be committing changes to the copy of the project in the repository. To get these changes, you may "update" your Workbench to match the state of the branch. The changes you will see will be specific to the branch that your Workbench project is configured to share. You control when you choose to update.

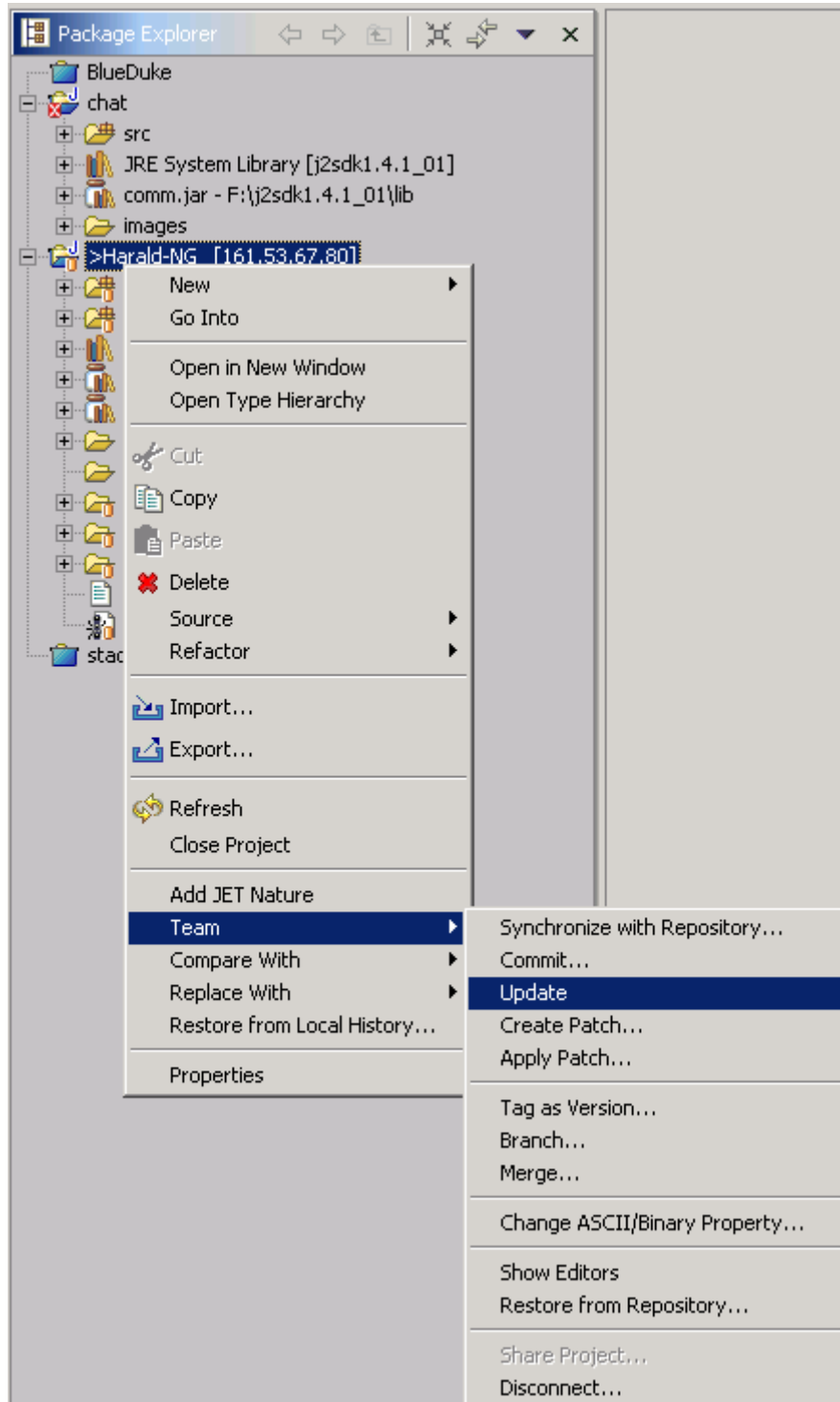


Fig. D.5: Updating the local project

The update command can be issued from two places: the **Team > Update** menu, or the **Synchronize** view. In order to understand the difference between these two commands, it is important to know about the three different kinds of incoming changes.

- A *non-conflicting* change occurs when a file has been changed remotely but has not been modified locally.
- An *automergetable conflicting* change occurs when an ASCII file has been changed both remotely and locally (i.e. has non-committed local changes) but the changes are on different lines.

Java Bluetooth stack	Version: 0.5
Developer's Guide	Date: 2004-01-10

- A *non-automergable conflicting* change occurs when one or more of the same lines of an ASCII file or when a binary file has been changed both remotely and locally (binary files are never automergable).

When you select **Team > Update**, the contents of the local resources will be updated with incoming changes of all of the above three types. For non-conflicting and automergable conflicts, there is no additional action required (for automergable conflicts, the changed local resource is moved to a file prefixed with ".#" just in case the automerge wasn't what the user wanted). However, for non-automergable conflicts, the conflicts are either merged into the local resource using special CVS specific markup text (for ASCII files) or the changed local resource is moved to a file prefixed with ".#" (for binary files). This matches the CVS command line behavior but can be problematic when combined with the Eclipse auto-build mechanism. Also, it is often desirable to know what incoming changes there are before updating any local resources. These issues are addressed by the Synchronize view.

To open the Synchronize view in incoming mode:

1. In the Navigator view, select the resources that you want to update.
2. From the pop-up menu for the selected resources, select **Team > Synchronize with Repository**. The Synchronize view will open.
3. On the toolbar of the Synchronize View, click the **incoming mode** button to filter out any modified Workbench resources (outgoing changes) that you may have.

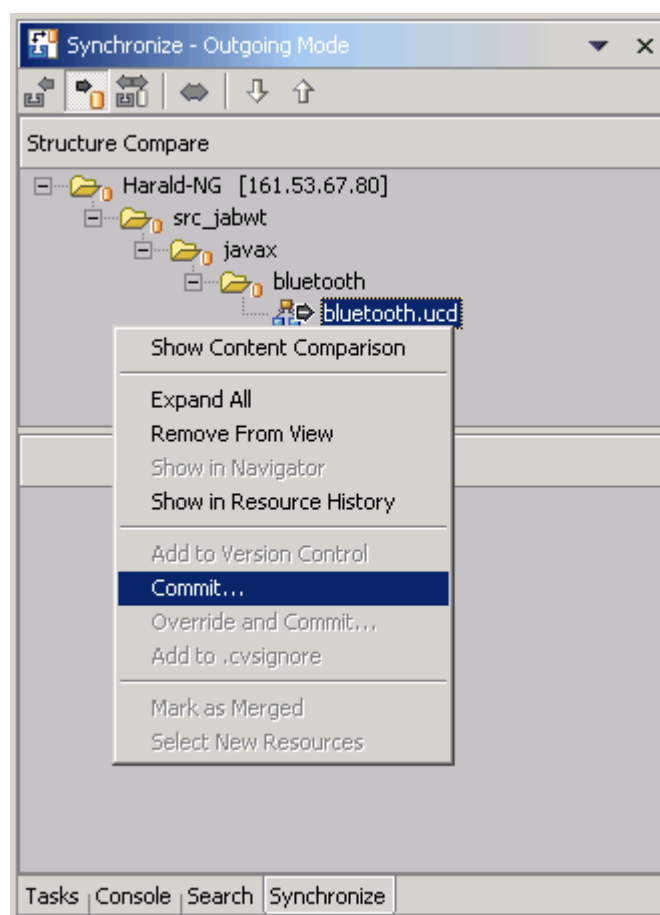


Fig. D.6: Committing the local changes

In incoming mode, you will see changes that have been committed to the branch since you last updated. The view will indicate the type of each incoming change (non-conflict, automergable

Java Bluetooth stack	Version: 0.5
Developer's Guide	Date: 2004-01-10

conflict or non-automergable conflict). There are two update commands (available from the context menu of any resource in the view) to deal with the different types of conflicts: **Update from Repository** and **Override and Update**. When you select the **Update from Repository** command in the Synchronize view, only non-conflicting changes are processed, leaving any files that have automergable or non-automergable conflicts in the view (any files that have been successfully processed are removed from the view). The **Override and Update** command operates on the two types of conflicts. After selecting this command, you will be prompted before a merge is attempted and asked if you want to auto merge the contents or overwrite them with the repository file. If you select to auto merge then only automergable conflicts will be processed and the incoming changes will be automerged with the local changes. Otherwise all conflicts will be processed and the local resources will be replaced with the remote contents. This "replace" behavior is rarely what is desired. An alternative is described later.

To update non-conflicting and automergable files:

1. The Structure Compare pane at the top of the Synchronize view contains the hierarchy of resources with incoming changes.
2. Select the non-conflicting files and choose **Update from Repository** from the pop-up menu. This will update the selected resources and remove them from the view.
3. Select the automergable conflicts and choose **Override and Update** from the pop-up menu. Select to only update automergable resources and click OK when prompted. This will update the selected resources and remove them from the view.

If your local Workbench contains any outgoing changes that are not automergable with incoming changes from the branch, then, instead of performing an **Override and Update**, you can merge the differences into your Workbench manually, as follows:

1. In the Structure Compare pane, if there is a conflict in the resource list (represented by red arrows), select it.
2. In the Text Compare area of the Synchronize view, local Workbench data is represented on the left, and repository branch data is represented on the right. Examine the differences between the two.
3. Use the text compare area to merge any changes. You can copy changes from the repository revision of the file to the Workbench copy of the file and save the merged Workbench file (using the pop-up menu in the left pane).
4. Once you are completed merging the remote changes into a local file, choose **Mark as Merged** from the pop-up menu. This will mark the local file as having been updated and allow your changes to be committed.

Note: The repository contents are not changed when you update. When you accept incoming changes, these changes are applied to your Workbench. The repository is only changed when you commit your outgoing changes.

Tip: In the Structure Compare pane, selecting an ancestor of a set of incoming changes will perform the operation on all the appropriate children. For instance, selecting the top-most folder and choosing **Update from Repository** will process all non-conflicting incoming changes and leave all other incoming changes unprocessed.

Warning: The behavior of the **Override and Update** command described above only applies to the incoming mode of the Synchronize view. In the **incoming/outgoing mode** of the view, the behavior for incoming changes and conflicts is the same but the command will revert outgoing changes to whatever the repository contents are. Exercise great caution if using this command in incoming/outgoing mode.

Java Bluetooth stack	Version: 0.5
Developer's Guide	Date: 2004-01-10

Committing

You can commit Workbench resources that you have modified to the repository so that other team members can see your work. Only those changes committed on that branch will be visible to others working on that branch. The commit command can be issued from two places: the **Team > Commit** menu, or the **Synchronize** view.

To commit changes using **Team > Commit**:

1. In the Navigator view, select the resources that you want to commit.
2. Right-click on the resources and select **Team > Commit** from the pop-up menu.
3. In the Commit Comment dialog box, provide a comment for your changes (for example, `Fixed the spelling mistakes`).

If there are conflicting changes on any files that are committed in the above fashion, the operation will fail. If this occurs, you must either perform an update or use the Synchronize view to resolve the conflicts. It is considered a more ideal workflow to always update before committing in order to ensure that you have the latest state of the repository before committing more changes.

If one or more of the resources being committed are new and not yet added to CVS control, you will be prompted to add these resources before the commit is performed. You can choose to add all, some or none of the new resources to CVS control before performing the commit. Any resources that are not under CVS control will not be committed. Committing from the Synchronize view also prompts if there are new resources.

To commit changes in the Synchronize view:

1. In the Navigator view, select the resources that you want to commit.
2. Right-click to open the pop-up menu and select **Team > Synchronize with Repository**. The Synchronize view will open.
3. On the toolbar of the Synchronize view, select the **outgoing mode** button to show any modified Workbench resources (outgoing changes) that you may have.
4. If there are conflicts (red arrows), resolve them. Use the text compare area to merge resources with conflicts. You can copy changes from the repository revision of the file to the Workbench revision of the file and save the merged Workbench resource. Once all the conflicts in the Structure Compare area have been resolved, you are ready to commit.
5. In the Structure Compare pane, right-click the top of the hierarchy that you want to commit, and select **Commit** from the pop-up menu.
6. In the Commit Comment dialog box, provide a comment for your changes (for example, `Fixed the spelling mistakes`).

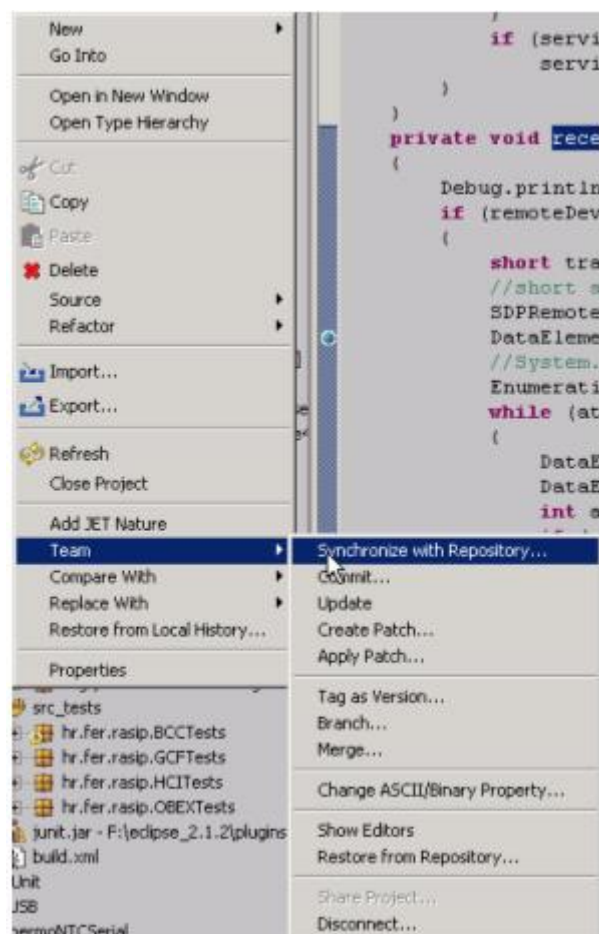


Fig. D.7: Synchronizing project with the repository

Java Bluetooth stack	Version: 0.5
Developer's Guide	Date: 2004-01-10

Tip: You can commit files that are in conflict by performing an **Override and Commit**. This will commit the Workbench copy of the resource into the repository and thus remove any of the incoming changes.

Warning: The behavior of the **Override and Commit** command described above only applies to the outgoing mode of the Synchronize view. In the **incoming/outgoing mode** of the view, the behavior for outgoing changes and conflicts is the same but the command will revert incoming changes to whatever the local Workbench contents are. Exercise great caution if using this command in incoming/outgoing mode.

Synchronizing with a CVS repository

In the CVS team-programming environment, there are two distinct processes involved in synchronizing resources: *updating* with the latest changes from a branch and *committing* to the branch.

When you make changes in the Workbench, the resources are saved locally. Eventually you will want to commit your changes to the branch so others can have access to them. Meanwhile, others may have committed changes to the branch. You will want to update your Workbench resources with their changes.

Important! It is preferable to update *before* committing, in case there are conflicts with the resources in your Workbench and the resources currently in the branch.

The synchronize view contains filters to control whether you want to view only *incoming changes* or *outgoing changes*. Incoming changes come from the branch. If accepted, they will update the Workbench resource to the latest version currently committed into the branch. Outgoing changes come from the Workbench. If committed, they will change the branch resources to match those currently present in the Workbench.

Regardless of which mode (filter) you select, the Synchronize view always shows you conflicts that arise when you have locally modified a resource for which a more recent version is available in the branch. In this situation you can choose to do one of three things: update the resource from the branch, commit your version of the resource to the branch, or merge your work with the changes in the branch resource. Typically you will want to merge, as the other two options will result in loss of work.

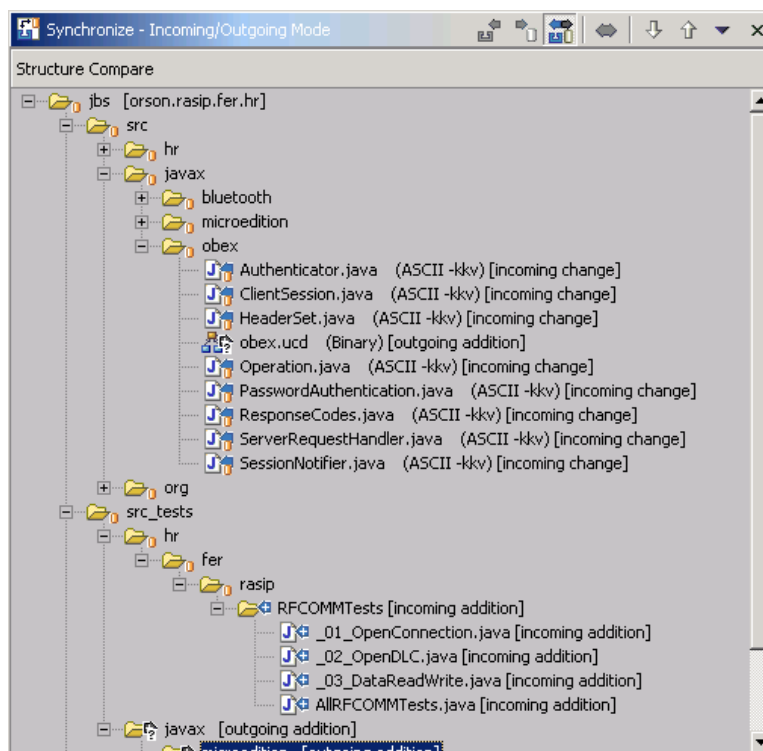


Fig. D.8: Synchronizing view

Java Bluetooth stack	Version: 0.5
Developer's Guide	Date: 2004-01-10

Literature and resources

Books

- [1] B. Hopkins, R. Anthony: *Bluetooth for Java*, Apress 2003.
- [2] Bluetooth SIG: *Specification of the Bluetooth System v1.1, vol. 1*, Bluetooth SIG, February 2001.

Links on the Internet

- [1] <http://sourceforge.net/projects/javablueetooth> - Javablueetooth on SourceForge
- [2] - BlueZ - Bluetooth stack for Linux
- [3] <http://java.sun.com/products/javacomm/downloads/index.html> - commAPI download
- [4] <http://www.eclipse.org> – Eclipse platform homepage
- [5] <http://www.junit.org> – JUnit homepage
- [6] [http://kobjects.dyndns.org/kobjects/auto?self=\\$81d91ea1000000f5d0738100](http://kobjects.dyndns.org/kobjects/auto?self=$81d91ea1000000f5d0738100) – ME4SE homepage
- [7] <http://www.sixlegs.com/software/png/> - sixlegs PNG library (classpath necessity for ME4SE)
- [8] - Eclipse CVSssh2 plug-in
- [9] <http://java.sun.com/products/j2mewtoolkit/> - Sun Wireless ToolKit
- [10] <http://developers.sun.com/techtopics/mobility/midp/articles/bluetooth1/> - deploying Wireless Java applications
- [11] <http://developers.sun.com/techtopics/mobility/midp/articles/bluetooth2/> - Wireless Application programming with J2ME and Bluetooth
- [12] <http://developers.sun.com/techtopics/mobility/midp/articles/threading2/> - using threads in J2ME applications
- [13] <http://jbs.sourceforge.net/> - JavaBluetoothStack on SourceForge.net
- [14] - JSR-82 specification page

Index