



# Java Bluetooth stack Acceptance Test Plan

Version 1.0

Doc. No.:



Java Bluetooth stack	Version: 1.0
Acceptance Test Plan	Date: 2004-01-07

## Revision History

<b>Date</b>	<b>Version</b>	<b>Description</b>	<b>Author</b>
2003-12-12	0.1	Initial Draft	Marko Đurić
2004-01-07	0.5	Documentation updated	Marko Đurić
2004-01-11	1.0	Final updates	Marko Đurić

Java Bluetooth stack	Version: 1.0
Acceptance Test Plan	Date: 2004-01-07

## 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
1.5	References	6
2.	Test-plan introduction	7
3.	Test items	7
3.1	Host Controller Interface (HCI) tests	7
3.2	Generic Connection Framework (GCF) tests	7
3.3	Bluetooth Control Center (BCC) tests	8
3.4	RFCOMM tests	8
3.5	OBject EXchange protocol (OBEX) tests	8
4.	Features to be tested	8
5.	Features not to be tested	8
6.	Approach	8
6.1	Approach to configuration and installation	8
6.1.1	Installing Eclipse platform	8
6.1.2	Installing commAPI	9
6.1.3	Importing projects to Eclipse platform	10
6.1.4	Writing JUnit tests	12
6.1.5	Running JUnit tests	16
6.1.6	Creating a new JUnit Run Configuration	17
6.1.7	Creating a JUnit test suite	18
6.1.8	Installing Sun Wireless ToolKit	19
6.1.9	Checking and configuring project references	19
7.	Item pass/fail criteria	20
7.1	Installation and Configuration	21
7.2	Documentation problems	21
8.	Suspension criteria and resumption requirements	21
9.	Environmental needs	21
9.1	Hardware	21
9.2	Software	21
9.3	Other	21
10.	Test procedure	22
10.1	Test case specifications	22
10.1.1	Host Control Interface (HCI) - AllHCITests	22
10.1.2	General Connection Framework (GCF) (AllGCFTests)	23
10.1.3	Bluetooth Control Center (BCC) (BCCAllTests)	24
10.1.4	RFCOMM (AllRFCOMMTests)	25
10.1.5	OBject EXchange protocol (OBEX) (AllOBEXTests)	26

Java Bluetooth stack	Version: 1.0
Acceptance Test Plan	Date: 2004-01-07

10.2	Test plan	27
11.	Responsibilities	27
11.1	Developers	27
11.2	User representative	27
12.	Risks and contingencies	27
13.	Approvals	27

## Table of figures

Fig. 6.1.1:	Eclipse platform main window - Resource perspective	9
Fig. 6.1.2:	Blackbox example main screen	10
Fig. 6.1.3:	Opening Package explorer view	11
Fig. 6.1.4:	Importing new project into Eclipse workspace	12
Fig. 6.1.5:	Rebuilding the project	12
Fig. 6.1.6:	Opening project's properties	13
Fig. 6.1.7:	Adding <i>junit.jar</i> to the build class-path	14
Fig. 6.1.8:	Using wizard for writing new JUnit TestCase or TestSuite	15
Fig. 6.1.9:	Creating a new JUnit TestCase	15
Fig. 6.1.10:	JUnit test run status: a) test failed, b) test successfully completed	16
Fig. 6.1.11:	Creating new JUnit run configuration	17
Fig. 6.1.12:	JUnit Run test created and ready to run	18
Fig. 6.1.13:	Creating a new JUnit TestSuite	19
Fig. 6.1.14:	Checking and correcting errors with project references	20

Java Bluetooth stack	Version: 1.0
Acceptance Test Plan	Date: 2004-01-07

## 1. Introduction

### 1.1 Purpose of this document

To improve quality of the final product, there should be some testing before release. With testing, it is possible to see how does application deals with stack initialization errors, communication errors, security errors and other errors.

Specific test routines are used for testing. This document describes those routines and represents the final test specification for the Java Bluetooth stack.

### 1.2 Intended Audience

This document is mainly intended for developers using this stack project, and to DSD test group as assurance that necessary testing has been done, and the project is ready for release.

### 1.3 Scope

Scope for this document covers the requirements for setting up and running a series of JUnit tests against Java Bluetooth Stack.

### 1.4 Definitions and acronyms

#### 1.4.1 Definitions

Keyword	Definitions
Host	Computer, laptop, PDA, device controlling the Bluetooth device
Host Controller	Bluetooth device containing radio
Host Transport Layer	Layer which connects Bluetooth stack to hardware interface (UART/PCMCIA/USB)

#### 1.4.2 Acronyms and abbreviations

Acronym or abbreviation	Definitions
OS	Operating System
WTK	Sun's Wireless Tool Kit for CLDC enabled devices
commAPI	Java communication Application Programming Interface
CLDC	Connected Limited Device Configuration
HCI	Host Controller Interface
GCF	General Connection Framework
BCC	Bluetooth Control Center
RFCOMM	Radio-Frequency COMMunication
OBEX	OBject EXchange protocol

### 1.5 References

- Eclipse download - <http://www.eclipse.org/downloads/index.php>
- commapi download - <http://java.sun.com/products/javacomm/downloads/index.html>
- JUnit cookbook - <http://junit.sourceforge.net/doc/cookbook/cookbook.htm>
- Wireless Toolkit - <http://java.sun.com/products/j2mewtoolkit/>
- Design Description for Java Bluetooth stack

Java Bluetooth stack	Version: 1.0
Acceptance Test Plan	Date: 2004-01-07

## 2. Test-plan introduction

All of the coding for Java Bluetooth Stack is done in Eclipse platform. This platform has been chosen because it's functionality can be expanded by lot of available plug-ins that could be found on the Internet.

One of this plug-ins adds support for JUnit automated testing framework to Eclipse platform. Most of the testing described in this document is done by JUnit tests.

### JUnit

JUnit is an open source Java testing framework used to write and run repeatable tests. It is originally written by Erich Gamma and Kent Beck.

Why use JUnit? Because test-code writing is much faster and easier, tests can be composed into a hierarchy of test suites, and it is free for use. It includes assertion for testing expected results, test fixtures for sharing common test data, test suites for easily organizing and running tests and textual and graphical test runners.

Developers usually don't have any time to write tests for their own code. They just suppose that the code works – maybe run it a couple of times to see if there will be some error messages. JUnit is used specially for that purposes – it creates automated testings that can be run at any time and it can test some parts of programs for errors even in the middle of coding procedure.

The basic object in JUnit is TestCase – developers often have test cases in their mind, but "there is no time" to write those tests and run it. Usually, when developers do some tests, it is just based on print statements, debugger expressions or test scripts. To easily manipulate with tests, we have to make them objects and with that we can make a concrete test from implicit example in developer's mind.

If we want to run more than one test at once, there are TestSuites that are specially built for that purpose. With only running a test suite, we can run all tests at once.

## 3. Test items

Testing of the Java Bluetooth Stack project is divided in 5 groups

- Host Controller Interface (HCI) tests
- Generic Connection Framework (GCF) tests
- Bluetooth Control Center (BCC) – security related tests
- RFCOMM tests
- Object EXchange protocol (OBEX) tests

HCI and GCF tests are only a help tool in establishing correct runtime environment for higher level layers tests, namely RFCOMM and OBEX.

### 3.1 Host Controller Interface (HCI) tests

Low-level Bluetooth layer testing such as: access to serial port at specific rates (Bluetooth device is mapped as a serial (COM) port), response of UART module on the COM port presets with the preset baud rate and flow control mode, and reading of Bluetooth addresses.

### 3.2 Generic Connection Framework (GCF) tests

Inquiry tests and tests if the Generic Connection Framework functions are functioning OK.

Java Bluetooth stack	Version: 1.0
Acceptance Test Plan	Date: 2004-01-07

### 3.3 Bluetooth Control Center (BCC) tests

Bluetooth Control Center is responsible for setting the security level for device, maintaining a list of earlier discovered devices and for pairing up and authorization of the devices.

Tests made on BCC are related to initialization of the BCC, and authentication and authorization of the devices.

### 3.4 RFCOMM tests

RFCOMM is protocol used on Bluetooth enabled devices as a cable-replacement technology. Tests related to this are opening connections, establishing Data Link Connections (DLC) and communicating over DLC (reading and writing of data).

### 3.5 Object EXchange protocol (OBEX) tests

OBEX is protocol mainly developed for transferring objects over the RFCOMM or TCP link. This objects can be files, business cards (vCards), calendars (vCalendar) and other types.

Testing made on OBEX protocol are related to server side connection establishing, client side connection establishing, and objects GET operation.

## 4. Features to be tested

Basic Bluetooth stack functionality – sending HCI commands, receiving and responding to events, Generic Connection Framework basic functions, higher level layers test.

## 5. Features not to be tested

Hardware conformance to Bluetooth 1.1 core specifications, Bluetooth error correction / behavior in electromagnetically loud environments.

## 6. Approach

Tests will be fully implemented using automated testing mechanisms in Java, mainly JUnit tests which will be run using Eclipse development platform. This requires some basic skills for working with Eclipse platform, which are described later on in the document. All tests reside in the *src\_tests* project's subdirectory, and for each one tester has to create a new run configuration.

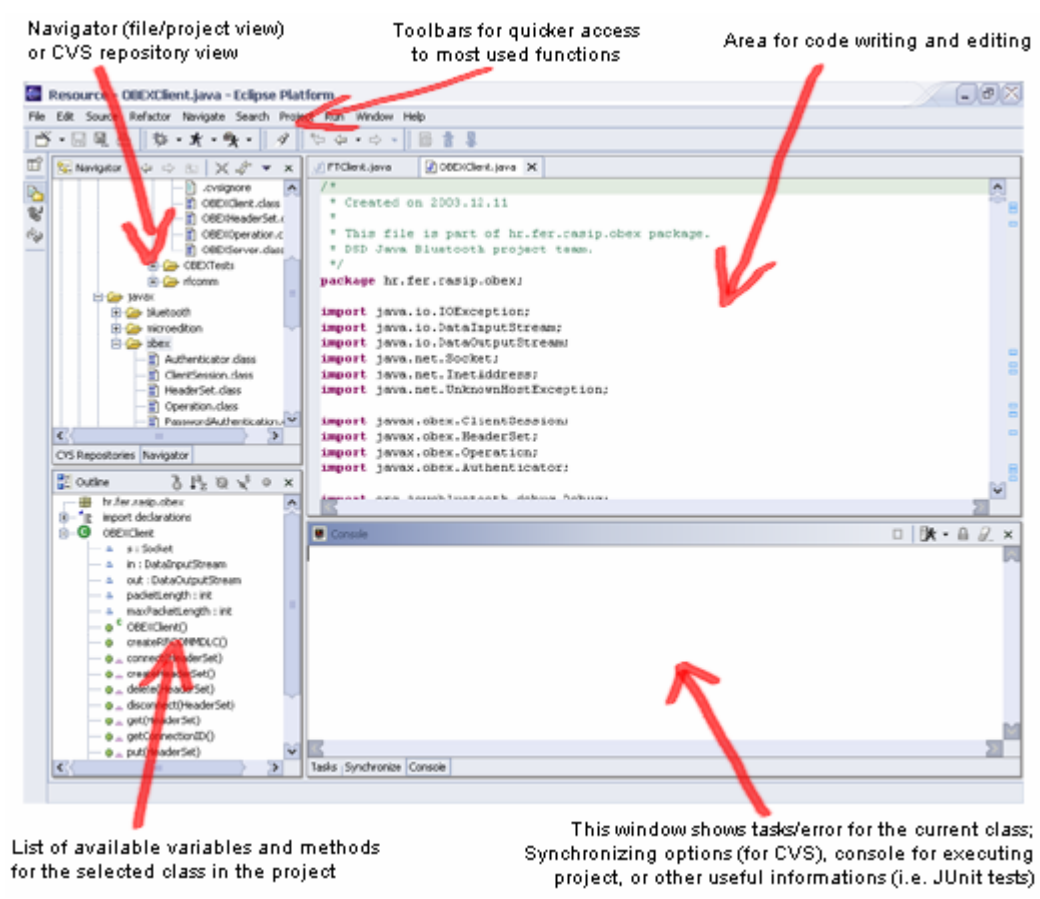
In order for any testing to be done, you need to have at least two pieces of Bluetooth H4 (UART) Host transport layer enabled hardware. For instructions for installing such devices, please refer to chapter 8 (Hardware installation guide) of the Developer guide for Java Bluetooth Stack.

### 6.1 Approach to configuration and installation

#### 6.1.1 Installing Eclipse platform

For Eclipse platform installation, please refer to the Eclipse documentation (<http://www.eclipse.org/documentation/main.html>)

Java Bluetooth stack	Version: 1.0
Acceptance Test Plan	Date: 2004-01-07



**Fig. 6.1.1: Eclipse platform main window - Resource perspective**

**6.1.2 Installing commAPI**

- Download the commAPI package from the Sun's java commAPI download site (<http://java.sun.com/products/javacomm/downloads/index.html>)
- Unzip the javacomm20-win32.zip archive. 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 archive in your c:\ partition, and your JDK installation is in C:\j2sdk1.4.2\_02.

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

- copy win32com.dll to your <JDK>\bin and to your <JDK>\jre\bin directories  
C:\>copy c:\commapi\win32com.dll c:\j2sdk1.4.2\_02\bin  
C:\>copy c:\commapi\win32com.dll c:\j2sdk1.4.2\_02\jre\bin
- copy comm.jar to your <JDK>\lib and to your <JDK>\jre\lib directories  
C:\>copy c:\commapi\comm.jar c:\j2sdk1.4.2\_02\lib  
C:\>copy c:\commapi\comm.jar c:\j2sdk1.4.2\_02\jre\lib
- copy javax.comm.properties to your <JDK>\lib and to your <JDK>\jre\lib directories

```
C:\>copy c:\commapi\javax.comm.properties c:\j2sdk1.4.2_02\lib
C:\>copy c:\commapi\javax.comm.properties c:\j2sdk1.4.2_02\jre\lib
```

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

Java Bluetooth stack	Version: 1.0
Acceptance Test Plan	Date: 2004-01-07

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
```

- 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
```

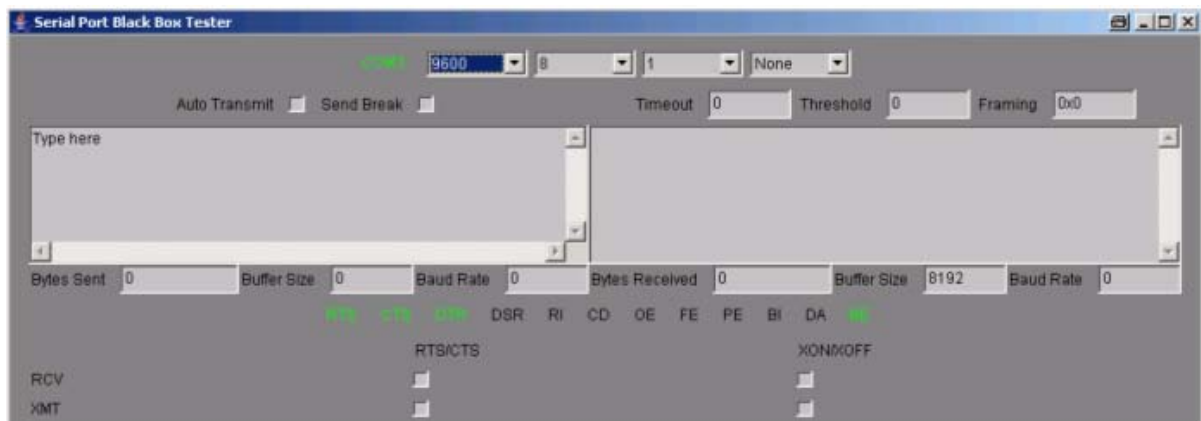


Fig. 6.1.2: Blackbox example main screen

### 6.1.3 Importing projects to Eclipse platform

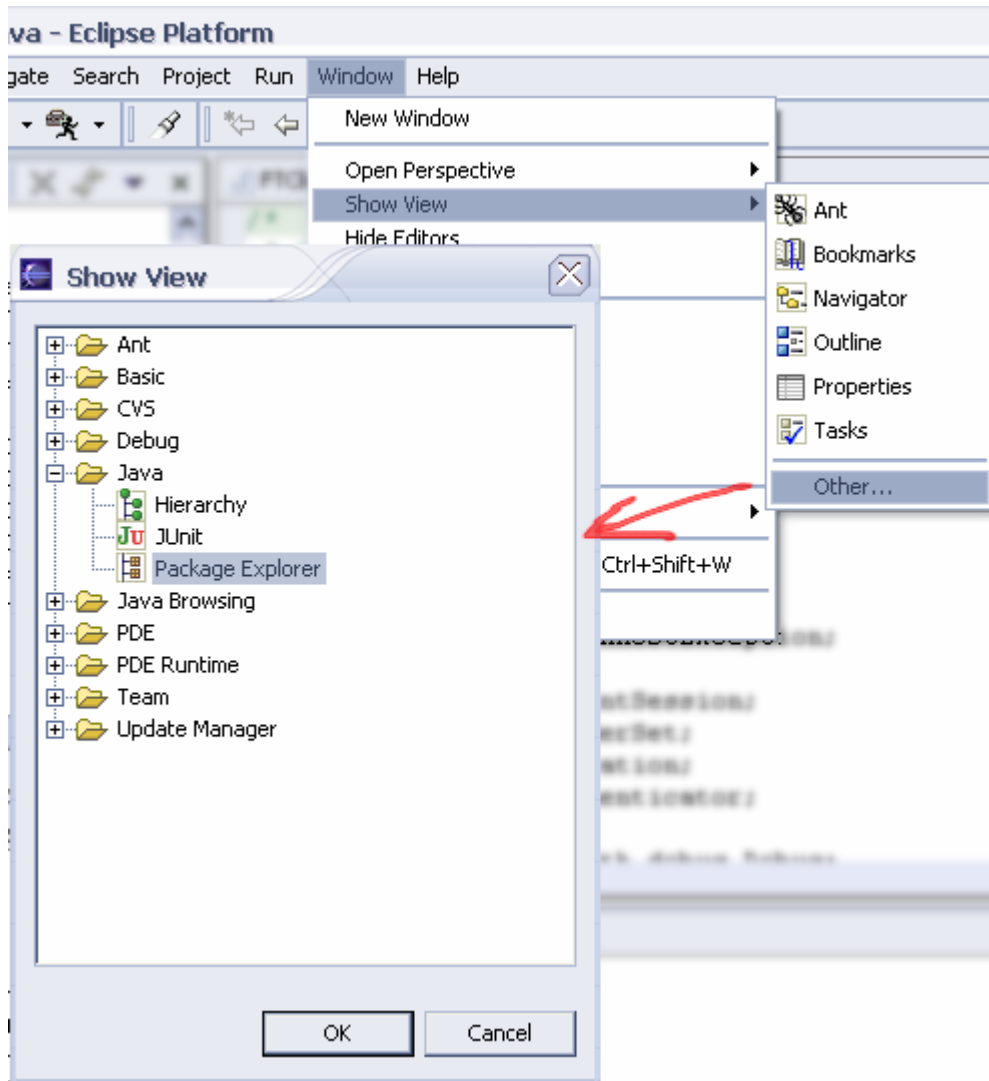
- Download the zipped project.

If the Eclipse platform is running, close it. Unzip the project into the <ECLIPSE\_HOME>\workspace directory (where <ECLIPSE\_HOME> is the directory in which you have installed the Eclipse platform), and start the Eclipse platform.

- Open package explorer view:

If you can't see it, click on **Window -> Show View -> Other ...**, and then choose **Java -> Package Explorer** (Fig. 6.1.3)

Java Bluetooth stack	Version: 1.0
Acceptance Test Plan	Date: 2004-01-07

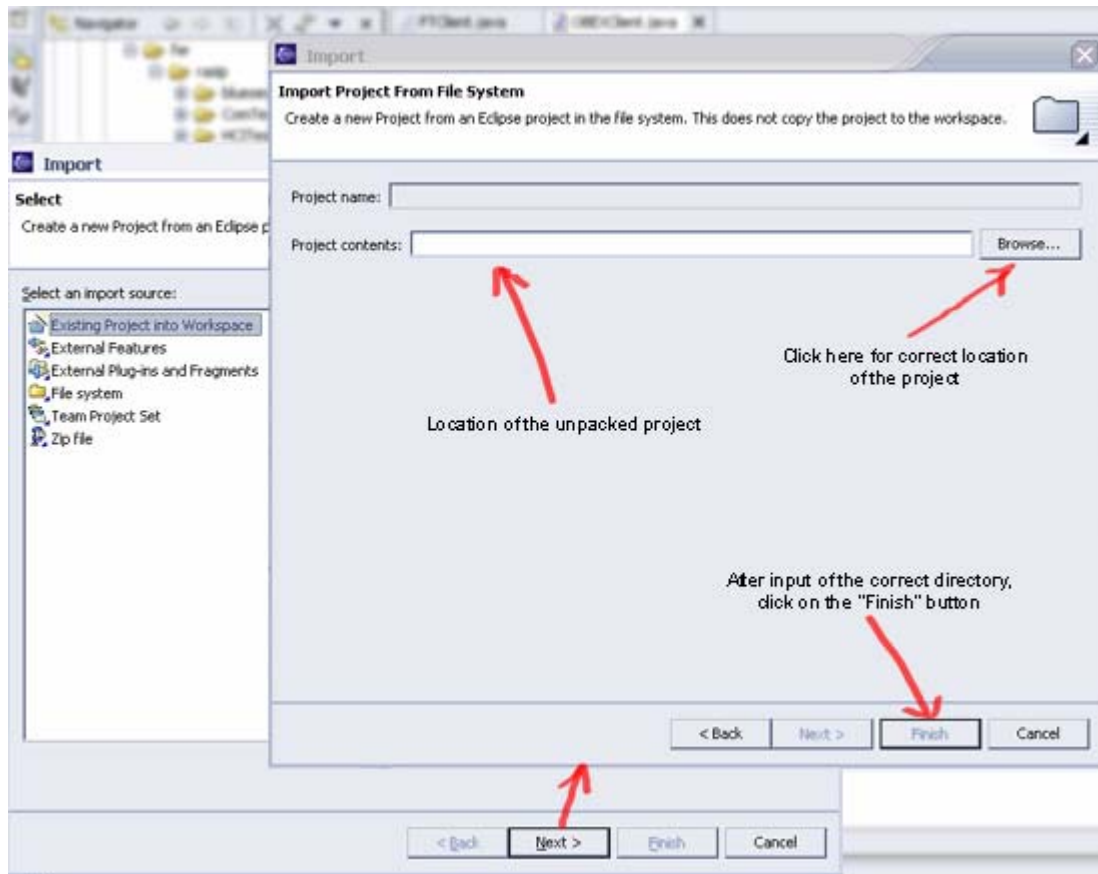


**Fig. 6.1.3: Opening Package explorer view**

- If the project already exists in the workspace, then in package explorer window click with the right mouse button, and choose refresh from the drop-down menu.

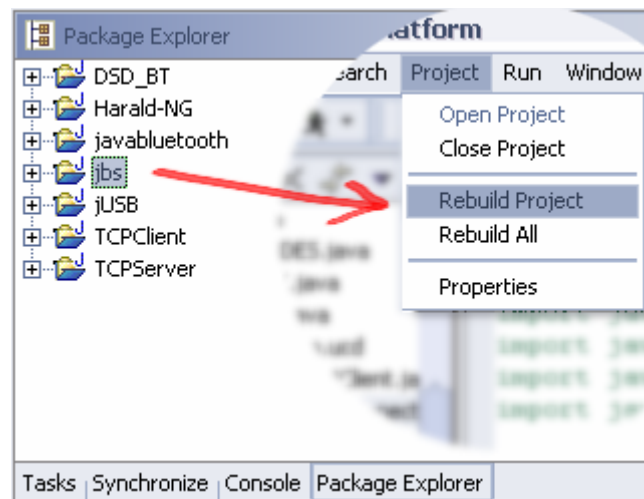
If it's a new project, instead of refresh, choose Import from drop-down menu and then choose **Existing Project into workspace** option. In the next window, select the folder where you unpacked the project, or click on **Browse** button to locate it (Fig. 6.1.4).

Java Bluetooth stack	Version: 1.0
Acceptance Test Plan	Date: 2004-01-07



**Fig. 6.1.4: Importing new project into Eclipse workspace**

- Select the project name in the package explorer window, and rebuild the project (**Project -> Rebuild Project**) (Fig. 6.1.5)



**Fig. 6.1.5: Rebuilding the project**

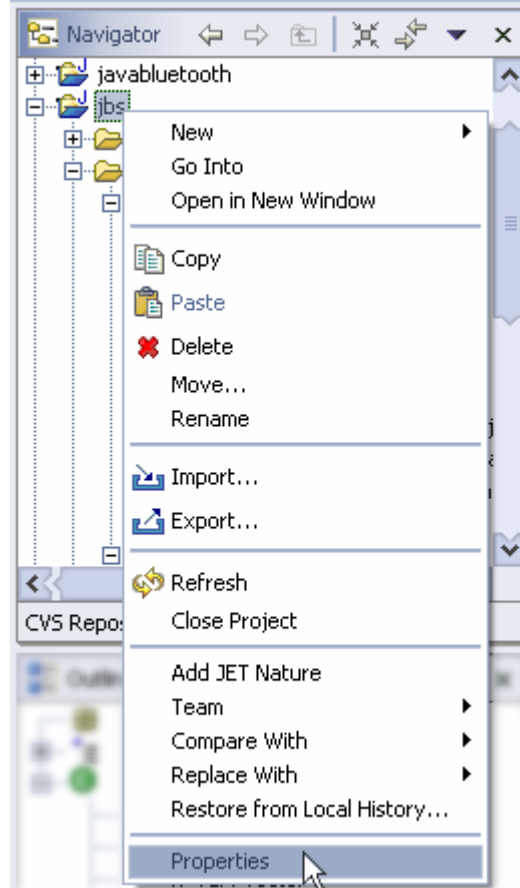
#### 6.1.4 Writing JUnit tests

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:

Java Bluetooth stack	Version: 1.0
Acceptance Test Plan	Date: 2004-01-07

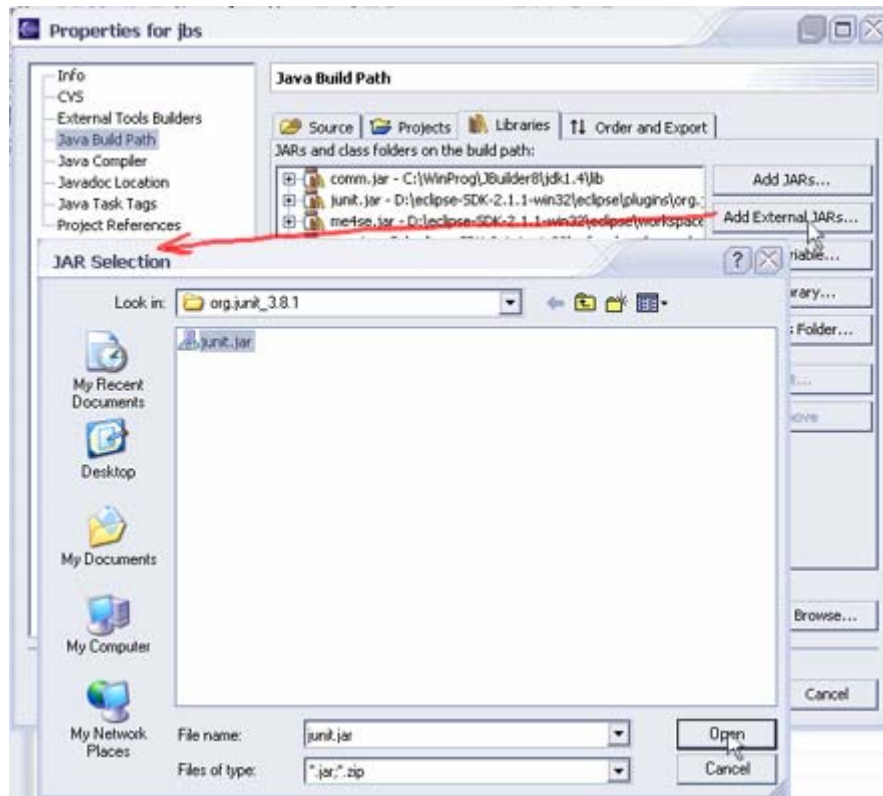
- right-click on the project's root directory in the Navigator pane and select **Properties**



**Fig. 6.1.6: 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. 6.1.7.)
- From the JAR selection window (Fig. 6.1.7.) 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

Java Bluetooth stack	Version: 1.0
Acceptance Test Plan	Date: 2004-01-07



**Fig. 6.1.7: 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. 6.1.8.)

Java Bluetooth stack	Version: 1.0
Acceptance Test Plan	Date: 2004-01-07

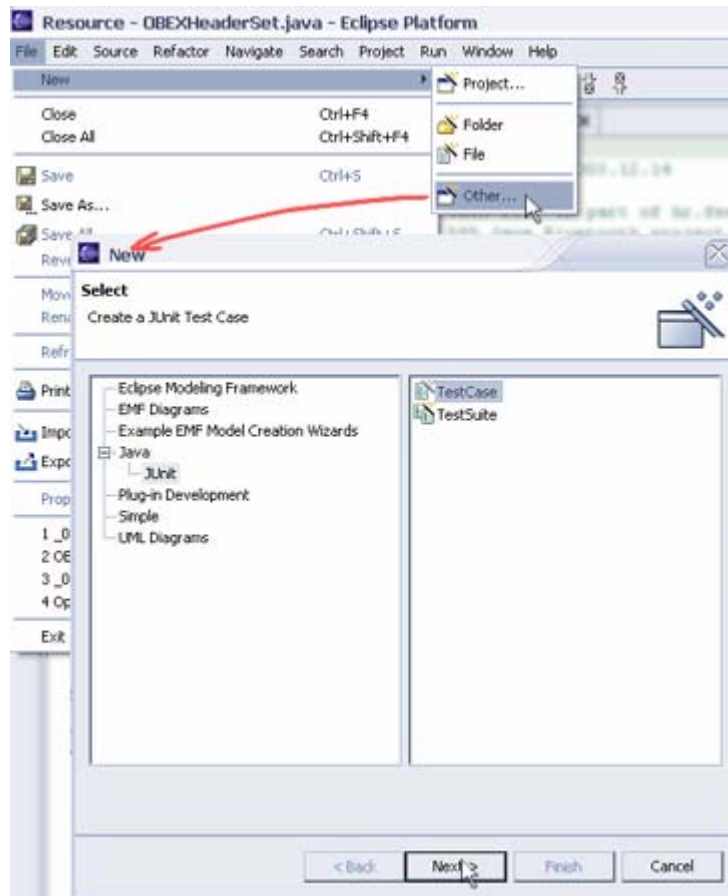


Fig. 6.1.8: Using wizard for writing new JUnit TestCase or TestSuite

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

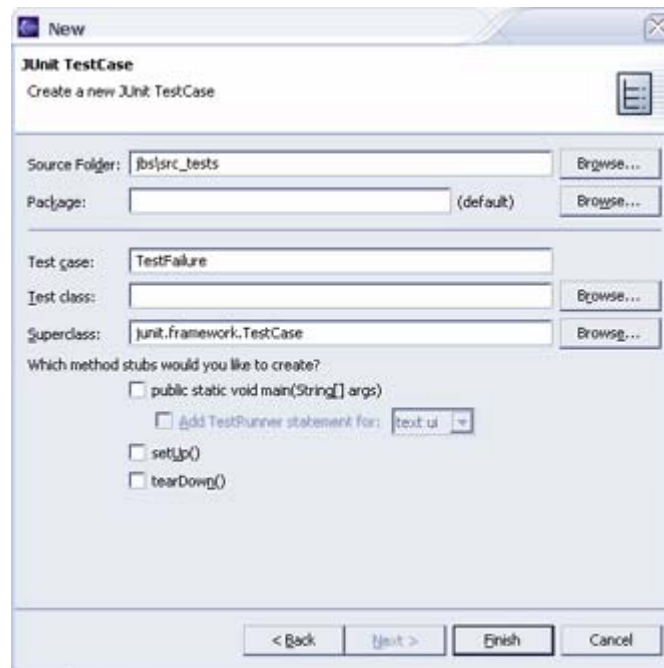


Fig. 6.1.9: Creating a new JUnit TestCase

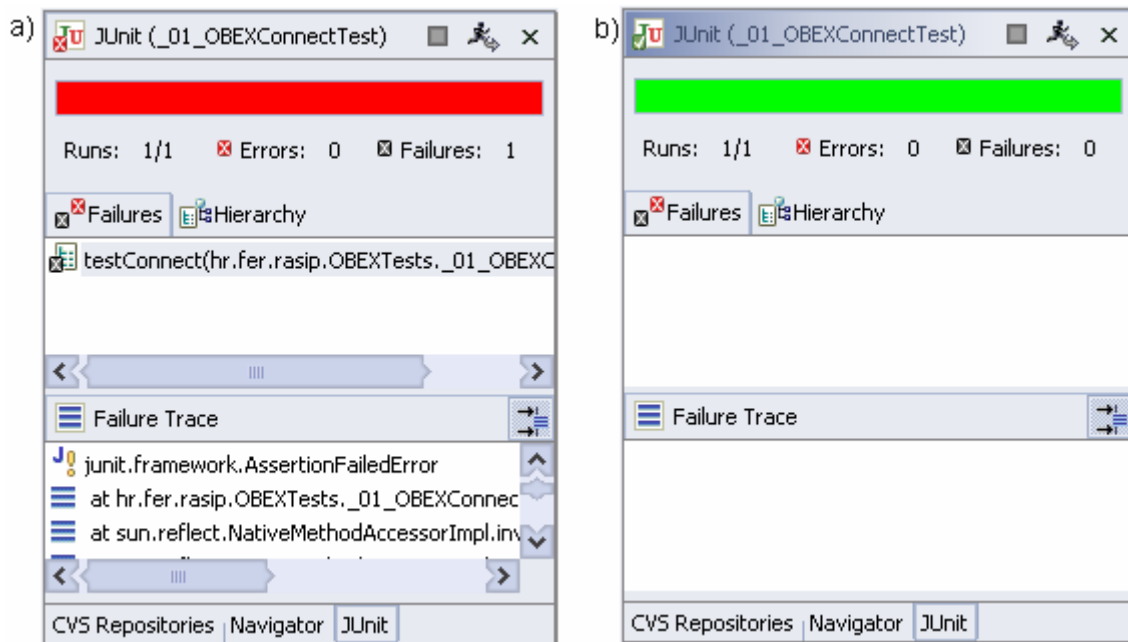
- Click **Finish** to create the test class

Java Bluetooth stack	Version: 1.0
Acceptance Test Plan	Date: 2004-01-07



- Add the testing procedure to the test class

### 6.1.5 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. 6.1.10: JUnit test run status: a) test failed, b) test successfully completed**

Fig. 6.1.10. 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

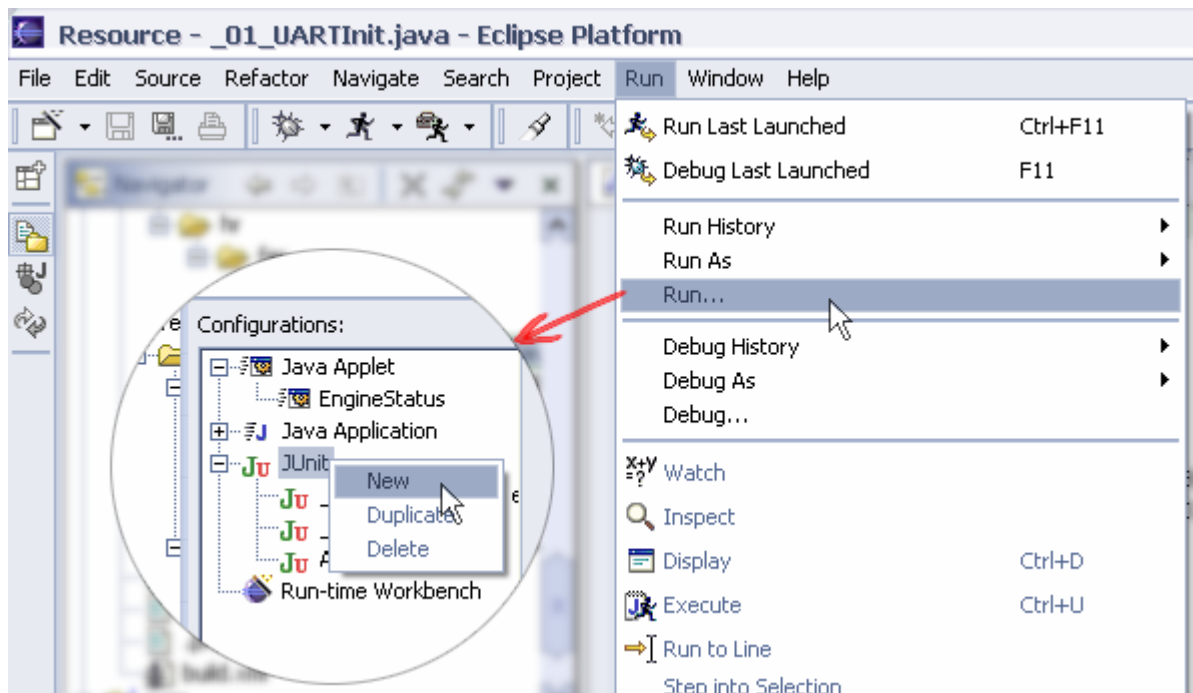
Java Bluetooth stack	Version: 1.0
Acceptance Test Plan	Date: 2004-01-07

- 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

### 6.1.6 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. 6.1.11.)



**Fig. 6.1.11: 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. 6.1.12.)

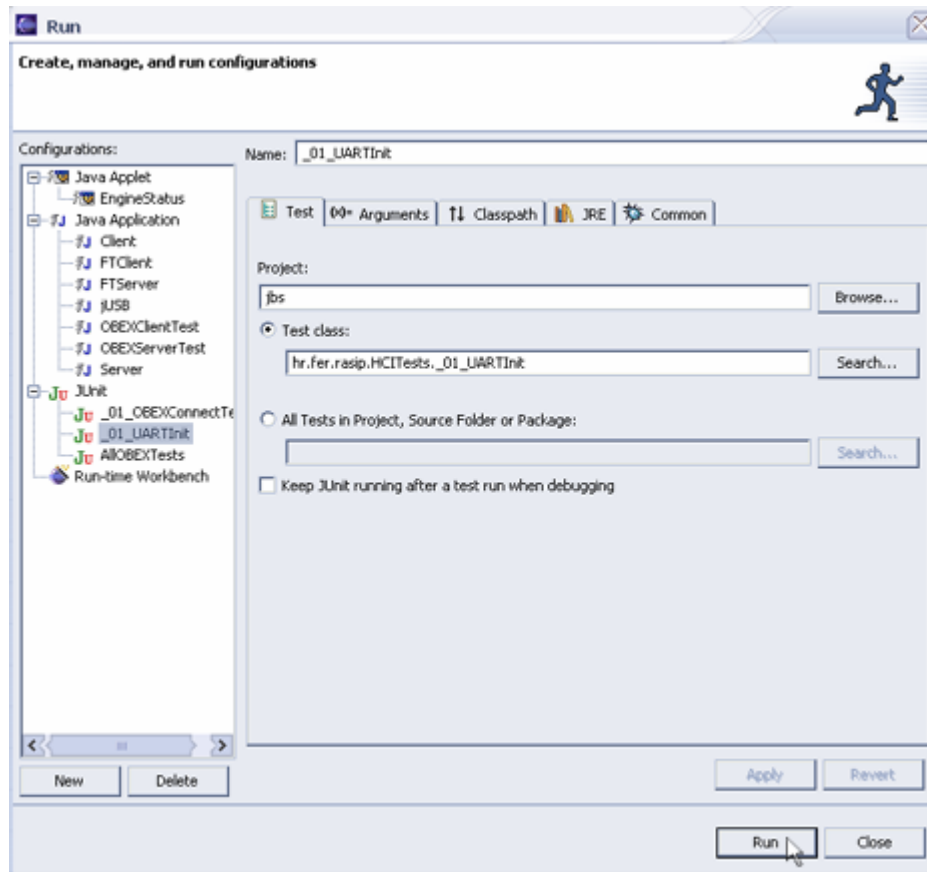


Fig. 6.1.12: JUnit Run test created and ready to run

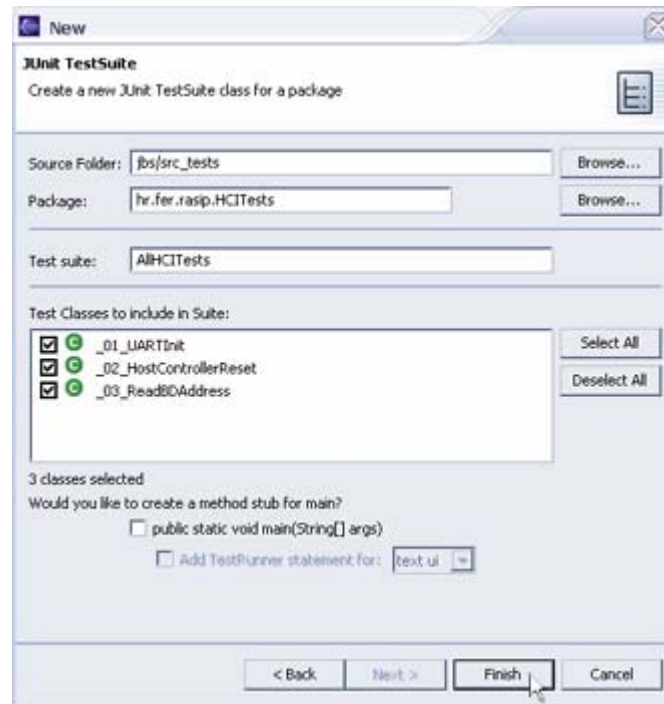
### 6.1.7 Creating a JUnit test suite

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. 6.1.8.)
- enter a name for your test suite class (the convention is to use "AllTests" or similar)

Java Bluetooth stack	Version: 1.0
Acceptance Test Plan	Date: 2004-01-07



**Fig. 6.1.13: 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.

#### 6.1.8 Installing Sun Wireless Toolkit

In order to write applications for mobile devices, we have to use Java edition that is suited for devices with small amount of memory (at least 512k). This edition of Java platform is called Java 2 Micro Edition (J2ME).

APIs for MIDP (Mobile Information Device Profile) platform are available with Sun Wireless Toolkit, so we have to install it first in order to use that library.

Installation is very simple:

- download the latest edition of Sun WTK from <http://java.sun.com/products/j2mewtoolkit/>
- follow the detailed install instructions

NOTE: During installation is very important to remember where is WTK installed, so it can be easier later to find the `midpapi.zip` (of course, it is always possible to search for that library on the disk drive in case we forgot where is WTK installed).

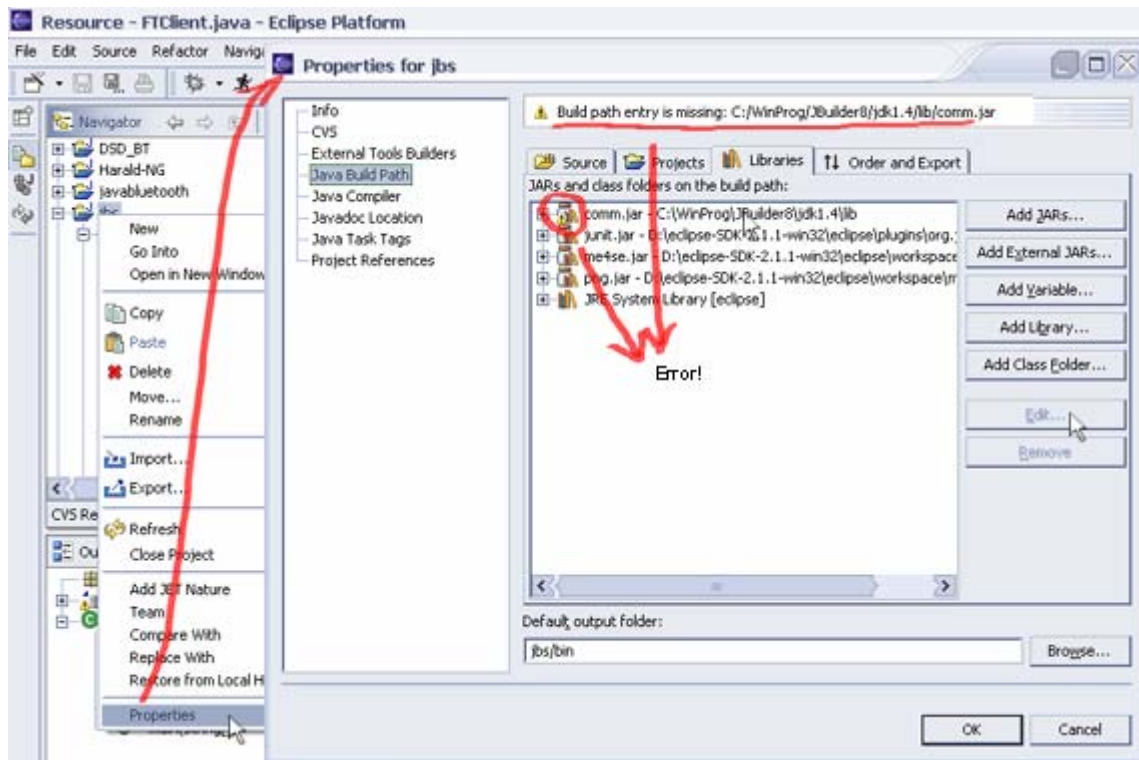
#### 6.1.9 Checking and configuring project references

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

Java Bluetooth stack	Version: 1.0
Acceptance Test Plan	Date: 2004-01-07

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 what 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. 6.1.14.)



**Fig. 6.1.14: Checking and correcting errors with project references**

As shown on Figure 6.1.14. 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

## 7. Item pass/fail criteria

All test run individually should pass. Care has to be taken when running test suites – on some machines and with some Bluetooth devices test suites can issue to many HCI commands in a too small amount of time, so the Host Controller will fail to parse them all and as a result test will fail. Although ever precaution has been taken to ensure the test suites will run on most host controllers, we cannot guarantee that for certain.

Per-test item criteria will be described, when needed, in chapter 10 for each test separately. General recommendation is to run tests one by one.

Java Bluetooth stack	Version: 1.0
Acceptance Test Plan	Date: 2004-01-07

## 7.1 Installation and Configuration

First you should add at least one piece of Bluetooth hardware to the test configuration. Any Bluetooth hardware, which has H4 (UART) Host transport layer, will suffice, as long as it is recognized by the OS as a COM port. You should take note of which COM port exactly has the device occupied, so you can specify that port number in the *Parameters tab* of the each test run configuration.

If you don't have Eclipse installed, or if you're unsure whether you have Java commAPI installed/configured right, refer to the sections 6.1.1. Installing Eclipse and 6.1.2. Installing Java commAPI.

You should download the *jbs* Eclipse project and import it into Eclipse workspace as described in section 6.1.3. Importing projects into Eclipse workspace.

In the package Explorer view you should locate the *src\_tests* source subdirectory, and in it the test packages. For each test, you must create a new run configuration as described in section 6.1.6. Creating a new JUnit Run configuration. Make sure you specify the correct COM port name in the parameters tab of the JUnit run configuration.

## 7.2 Documentation problems

Because there are different versions of the Eclipse development platform out there, there can minor discrepancies between this install instructions and the actual locations of menus and options involved in the configuration process. If some fatal error arises during setup, you should revert to Eclipse 2.1.2 and check for the location of that particular option/command, and then find it in the Eclipse version you're running.

This test plan relies heavily on the stack – if some part of source in the stack is changed tests can easily be affected.

## 8. Suspension criteria and resumption requirements

If one test fails, testing should be continued with the next one.

If more than two tests fail when run individually the testing procedure should be restarted, initial hardware and software setup reinspected, and then can the procedure be restarted.

## 9. Environmental needs

### 9.1 Hardware

- PC capable of running Eclipse at reasonable speed (any computer with 600MHz CPU or faster, and 128MB RAM or more should suffice)
- Bluetooth serial module or Bluetooth PC-CARD internally represented as COM port

### 9.2 Software

- Java 2™ SDK 1.4.0 or higher installed
- Eclipse 2.1 or higher installed (not recommended version above 3, because of some problems with needed plug-ins)
- Java commAPI installed correctly

### 9.3 Other

- Environment variables set like specified in section 6.1.2. Installing commAPI

Java Bluetooth stack	Version: 1.0
Acceptance Test Plan	Date: 2004-01-07

## 10. Test procedure

All of the testing is done by using JUnit TestCases (for individual tests) an JUnit TestSuites (for group tests).

### 10.1 Test case specifications

#### 10.1.1 Host Control Interface (HCI) - AllHCITests

##### 10.1.1.1 UART Initialization (\_01\_UARTInit)

*Description:* Test to see if the COM port which will be used by Bluetooth UART module is available to Java

*Test type:* positive (test must pass if any other tests ought to be run)

*Preconditions:* Java commAPI must be installed; Correct path to comm.jar has to be listed on the **Properties -> Java Build Path -> Libraries** tab of the JBS project properties; COM port name has to correspond to the one of the test system to which Bluetooth module is connected to; baud rate and flow control have to be set correctly for the module in question

*Input definition:* N/A

*Output definition:* N/A

*Remarks:* Tester has to change COM port parameters according to the hardware installed on the test machine

##### 10.1.1.2 Host controller reset (\_02\_HostControllerReset)

*Description:* Test to see if the Bluetooth UART module is responding on the preset COM port with the preset baud rate and flow control

*Test type:* positive (test must pass in any further tests ought be run)

*Preconditions:* Java commAPI must be installed; correct path to comm.jar has to be listed on the **Properties -> Java Build Path -> Libraries** tab of the JBS project properties; COM port name has to correspond to the one of the test system to which Bluetooth module is connected to; baud rate and flow control have to be set correctly for the module in question

*Input definition:* N/A

*Output definition:* N/A

*Remarks:* Tester has to change COM port parameters according to the hardware installed on the test machine

##### 10.1.1.3 Reading Bluetooth Device Address (\_03\_ReadBDAddress)

*Description:* Test to see if the Bluetooth UART module is responding correctly to a simple HCI Command, and if the bytes received from the COM port are being composed into packets properly

*Test type:* positive (test must pass in any further tests ought be run)

*Preconditions:* Test \_02\_HostControllerReset has to be run prior this one, and it has to succeed; this test parameters have to be set accordingly to the successfully passed \_02\_HostControllerReset test

*Input definition:* N/A

*Output definition:* Valid Bluetooth address printed on the standard output

*Remarks:* Tester has to run \_02\_HostControllerReset test prior this one!

Java Bluetooth stack	Version: 1.0
Acceptance Test Plan	Date: 2004-01-07

#### 10.1.1.4 Reading Bluetooth Device Address (batch mode) (\_03\_ReadBDAddressBatch)

*Description:* Test to see if the Bluetooth UART module is responding correctly to a simple HCI Command, and if the bytes received from the COM port are being composed into packets properly

*Test type:* positive (test must pass in any further tests ought be run)

*Preconditions:* This test has to be run only within AllHCITest test suite, and never standalone

*Input definition:* N/A

*Output definition:* Valid Bluetooth address printed on the standard output

*Remarks:* This test has to be run only within AllHCITest test suite, and never standalone

#### 10.1.2 General Connection Framework (**GCF**) (AllGCFTests)

Inquiry test requires two Bluetooth devices used in conjunction - \_02\_InquiryTestServer has to be run first, and then \_02\_InquiryTestClient, with no more than 10 – 20 seconds between starting these two tests, for successful device discovery.

This two tests take significant amount of time (about 2 minutes) to complete

##### 10.1.2.1 Write/Read friendly local name (\_01\_WriteReadFriendlyLocalName)

*Description:* Test to see if GCF (Generic Connection Framework) functions are functioning OK

*Test type:* positive (test must pass)

*Preconditions:* Test \_02\_HostControllerReset (or AllHCITestSuite test suite) has to be run prior this test, and it has to succeed; this test parameters have to be set accordingly to the successfully passed \_02\_HostControllerReset test

*Input definition:* N/A

*Output definition:* Bluetooth address and "Blue Devil" (friendly name of the device) printed on the standard output

*Remarks:* Tester has to run \_02\_HostControllerReset or AllHCITestSuite test suite prior to this test!

##### 10.1.2.2 Inquiry test – client (\_02\_InquiryTestClient)

*Description:* Test to see if local device performs inquiry correctly. This client looks for any discoverable Bluetooth device in the vicinity, reads its BDAAddress and a friendly name. \_02\_InquiryTestServer HAS to be up and waiting for someone to discover it before running this test

*Test type:* positive

*Preconditions:* Test \_02\_HostControllerReset has to be run prior this one, and it has to succeed; this test parameters have to be set accordingly to the successfully passed \_02\_HostControllerReset test

*Input definition:* N/A

*Output definition:* Bluetooth address of the first found device printed on standard output

*Remarks:* Tester has to run \_02\_HostControllerReset test or AllHCITestSuite test suite prior to this test!

##### 10.1.2.3 Inquiry test – server (\_02\_InquiryTestServer)

*Description:* Test to see if remote device responds to inquiry – this server just sets his discoverable mode to true and waits to be discovered for 60 seconds. \_02\_InquiryTestClient has to be

Java Bluetooth stack	Version: 1.0
Acceptance Test Plan	Date: 2004-01-07

run no more than 10 – 20 seconds later than \_02\_InquiryTestServer

*Test type:* since there is no way in Eclipse to monitor two or more JUnit tests simultaneously, this test won't return any result when running together with \_02\_InquiryTestClient. When running in standalone mode, this test must succeed

*Preconditions:* Test \_02\_HostControllerReset (or AllHCITestSuite test suite) has to be run prior this one, and it has to succeed; this test parameters have to be set accordingly to the successfully passed \_02\_HostControllerReset test

*Input definition:* N/A

*Output definition:* Bluetooth address and "Blue Devil" (friendly name of the device) printed on standard output

*Remarks:* Tester has to run \_02\_HostControllerReset test or AllHCITestSuite test suite prior to this test!

### 10.1.3 Bluetooth Control Center (**BCC**) (BCCAllTests)

#### 10.1.3.1 BCC initialization (BCCInit)

*Description:* Test to see if the Bluetooth Control Center will pass initialization

*Test type:* positive (test must pass, because the whole security depends on BCC initialization success).

*Preconditions:* HCIDriver initialization needs to be done first and should be successful. This must be done because BCC.init() uses HCIDriver method calls

*Input definition:* N/A

*Output definition:* N/A

*Remarks:* HCIDriver must be initialized

#### 10.1.3.2 Return value of Authenticator request (BCCauthentication)

*Description:* Test to see if the return value of the Authentication request is added to AuthDevice database (list of known BT addresses)

*Test type:* positive only if remote device supports authentication and if connection between devices can be established. In order to apply Authentication\_Requested(short conn\_handle) method, connecting method is needed to identify the connection. Connection handler is retrieved from data attached to Connection Complete Event.

*Preconditions:* ServerURL needs to point to discovered device and known service. Connection specified with ServerURL needs to be established. Connection handle added to DevHandle database.

*Input definition:* N/A

*Output definition:* N/A

*Remarks:* Connection specified with ServerURL must be established

#### 10.1.3.3 Authentication mode setting (BCCTest)

*Description:* Test to see if the authentication mode can be set at the link setup level

*Test type:* positive

*Preconditions:* HCIDriver initialization needs to be done first and should be successful

*Input definition:* N/A

*Output definition:* N/A

Java Bluetooth stack	Version: 1.0
Acceptance Test Plan	Date: 2004-01-07

*Remarks:* HCIDriver must be initialized

#### 10.1.3.4 Authorization (BCCTrusted)

*Description:* Test to see if make\_trusted(long bd\_addr) method invocation updates record specified with long parameter bd\_addr in KnownDevices DataBase

*Test type:* positive

*Preconditions:* Record for remote device, specified with bd\_addr variable (device address) must exist in database Known Device else BCCEXception is thrown

*Input definition:* N/A

*Output definition:* N/A

*Remarks:* Record for remote device must exist in database Known Device

#### 10.1.4 **RFCOMM** (AllRFCOMMTests)

##### 10.1.4.1 Connection opening – server (\_01\_OpenServerConnection)

*Description:* Tests the Connect() method of the server.

*Test type:* positive (must be in order to create RFCOMM Data Link Connection (DLC)). Test passes if the L2CAP link and the control channel are successfully created.

*Preconditions:* A client connection to the server should be initiated for test work

*Input definition:* N/A

*Output definition:* N/A

*Remarks:* L2CAP link and the control channel must be successfully created

##### 10.1.4.2 Connection opening – client (\_01\_OpenClientConnection)

*Description:* Tests the Connect() method of the client.

*Test type:* positive (must be in order to create RFCOMM Data Link Connection (DLC)). Test passes if the L2CAP link and the control channel are successfully created

*Preconditions:* A server to which the client will connect should be started for test work.

*Input definition:* N/A

*Output definition:* N/A

*Remarks:* The L2CAP link and the control channel must be successfully created

##### 10.1.4.3 Data Link Connection (DLC) establishing (\_03\_OpenDLC)

*Description:* Tests the connectNewDLC() method.

*Test type:* positive – test passes if the L2CAP link and the control channel are created and the new DLCI is opened.

*Preconditions:* A client connection to the server must be initiated for test work

*Input definition:* N/A

*Output definition:* N/A

*Remarks:* A client connection to the server must be initiated for test work

Java Bluetooth stack	Version: 1.0
Acceptance Test Plan	Date: 2004-01-07

#### 10.1.4.4 Read/Write operations (\_04\_DataReadWrite)

*Description:* Tests the sending/receiving the data over the DLC by sending the test command. When the test command is received by the other entity it responds with another test command with the same data that was received. By comparing the sent and received data we can check if everything went well.

*Test type:* positive – test passes if the L2CAP link and the control channel are created and the new DLCI is opened

*Preconditions:* A Client connection to the server must be initiated for test work

*Input definition:* N/A

*Output definition:* N/A

*Remarks:* A client connection to the server must be initiated for test work

#### 10.1.5 OBject EXchange protocol (**OBEX**) (AllOBEXTests)

##### 10.1.5.1 Server connection (\_01\_OBEXConnectTest)

*Description:* Test to see if it's possible to create connection on the server side

*Test type:* positive (must be, because if there is no server, there is no communication)

*Preconditions:* RFCOMM layer must be operational if the OBEX communication is over RFCOMM.

*Input definition:* N/A

*Output definition:* N/A

*Remarks:* N/A

##### 10.1.5.2 Client connection (\_02\_OBEXClientConnectTest)

*Description:* Test to see if the communication can be set between client and server. It also checks the response code of OBEX CONNECT request (must be OBEX\_HTTP\_OK)

*Test type:* positive (must be, because the whole communication depends on initialization on connection time)

*Preconditions:* server must be up and waiting for client's request for communication; RFCOMM layer must be operational if the OBEX communication is over RFCOMM.

*Input definition:* N/A

*Output definition:* N/A

*Remarks:* Server must waiting for client's connection request, and send response upon receiving CONNECT request

##### 10.1.5.3 OBEX GET (\_03\_OBEXGETTest)

*Description:* Test to see if the instance of javax.obex.ClientSession implementation creates javax.obex.Operation object, and if that object returns the corresponding server response code OBEX\_HTTP\_OK

*Test type:* positive only if the javax.obex.ServerRequestHandler.onGet(Operation) method is implemented by the class that extends the javax.obex.ServerRequestHandler class

*Preconditions:* server must be up and waiting for client's request for communication; RFCOMM layer must be operational if the OBEX communication is over RFCOMM

*Input definition:* N/A

Java Bluetooth stack	Version: 1.0
Acceptance Test Plan	Date: 2004-01-07

*Output definition:* N/A

*Remarks:* Server must be up and waiting for client's connection request!

## 10.2 Test plan

The most logical testing order is first to test the lowest hardware layers, and then go up to the higher levels.

Test plan has the same sequence as the test item sequence

- at first, there are testings on the lowest hardware layer (HCI and GCF)
- security tests (BCC)
- RFCOMM tests
- and at the end OBEX tests

## 11. Responsibilities

### 11.1 Developers

- |                |                     |
|----------------|---------------------|
| Tomislav Sečen | - HCI and GFC tests |
| Dražen Njerš   | - BCC tests         |
| Sanjin Goglia  | - RFCOMM tests      |
| Marko Đurić    | - OBEX tests        |

### 11.2 User representative

User representative must make sure that all of tests pass, and that performance is optimal.

## 12. Risks and contingencies

There are many risks concerning hardware and software setup for this tests. Hardware risks include correct installation of the devices and drivers for them, while software risks include installing software tools needed for testing, namely Eclipse Platform, Java Communication API (commAPI), Sun Wireless ToolKit (WTK), and configuring Eclipse Platform for running JUnit tests.

If any of these items fail, it will be impossible to conduct the testing procedure, so extra care has to be taken towards correct software and hardware configuring, which is described in detail in this Test Plan.

## 13. Approvals

Name	Title	Date yyyy-mm-dd	Signature