Preliminary Reference Information

PRI Name	eWON Java Toolkit User guide		
Access	DIST		
Since revision	5.2s0		
PRI Number	PRI-0002-0	**Build**	75
Mod date	05/06/2012		

eWON Java Toolkit User guide

Summary:

Explains how to install the JAVA developpment environment and how to create and debug an application for the eWON with JAVA

Table of content

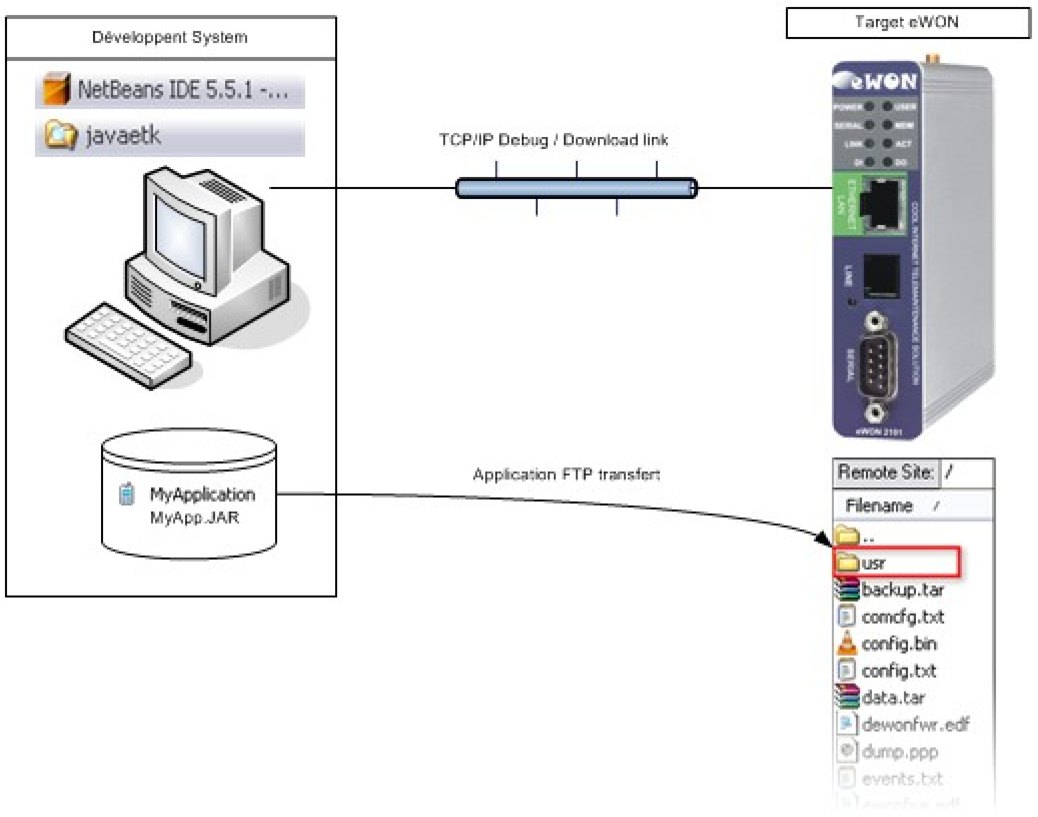
Introduction: J2ME and the eWON

The eWON JAVA toolkit is designed around the J2ME (JAVA Micro Edition) technology.

J2ME has been introduced for resource constrained mobility devices (like GSM). The eWON is such a device and the TCP/IP connectivity at the heart of the JAVA technology is well suited to the eWON environment.

The J2ME is also very interesting for the eWON because it separates very strongly the user's application from the eWON kernel application, while at the same time providing performance and flexibility for the user's application.

eWON JAVA development Platform

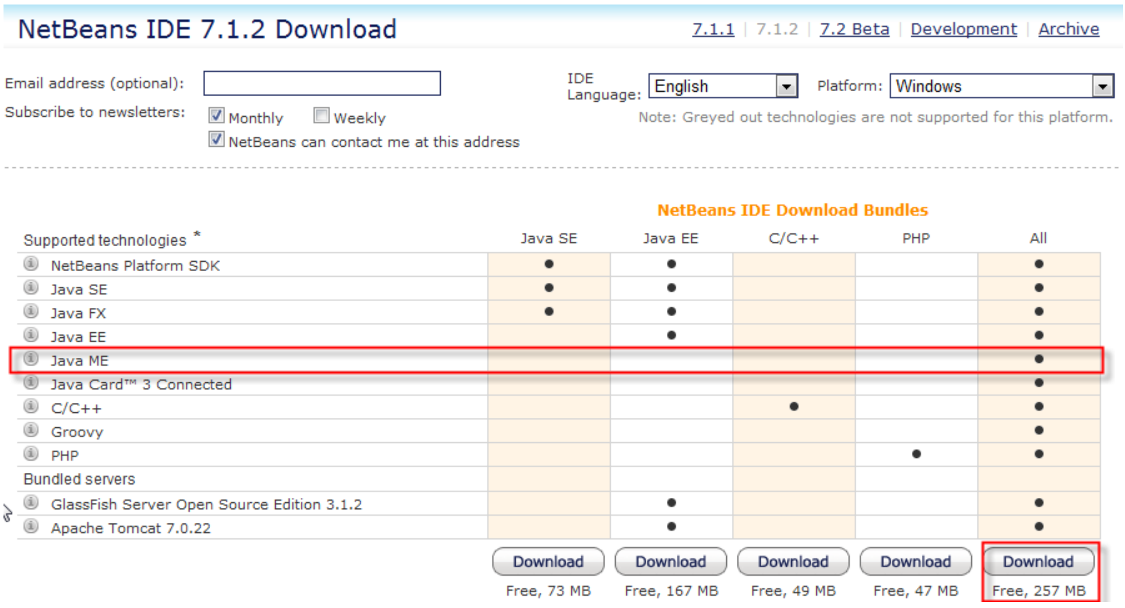


The JAVA application is coded and debugged using the Netbeans IDE.

Netbeans is a free project provided at www.netbeans.org and supported by ORACLE^{tm}.

The Netbeans IDE support is followed at eWON and the latest version tested is version 7.1.2 at the time this document was written. In case you have problems with a more recent version, please contact support@ewon.biz.

The NetBeans editor will use the Java ME environment. You should check the NetBeans index page to check which version of NetBeans is required to support the Java ME environment. At the time this document was written, the full version was required.



NetBeans IDE 7.1.2 Download

7.1.1 | 7.1.2 | 7.2 Beta | Development | Archive

Email address (optional):

Subscribe to newsletters: ☑ Monthly ☐ Weekly ☑ NetBeans can contact me at this address

IDE Language: English Platform: Windows

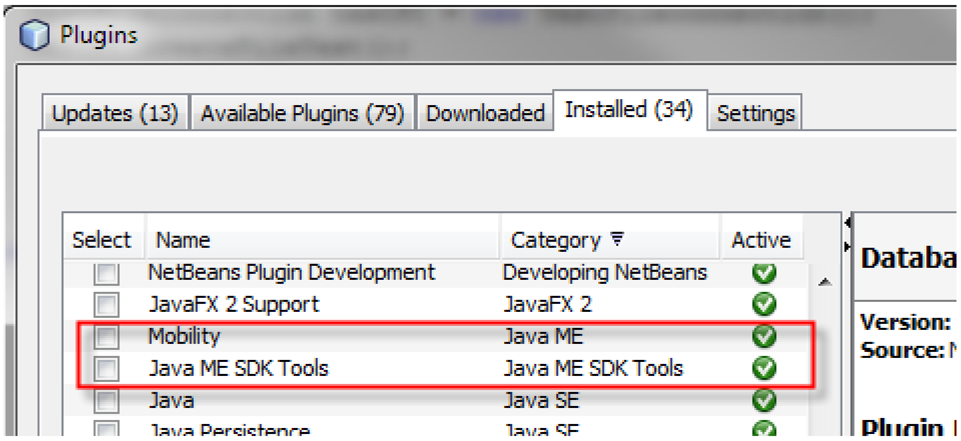
Note: Greyed out technologies are not supported for this platform.

NetBeans IDE Download Bundles

Supported technologies *	Java SE	Java EE	C/C++	PHP	All
NetBeans Platform SDK	●	●			●
Java SE	●	●			●
Java FX	●	●			●
Java EE		●			●
Java ME					●
Java Card™ 3 Connected					●
C/C++			●		●
Groovy					●
PHP				●	●
Bundled servers					
GlassFish Server Open Source Edition 3.1.2		●			●
Apache Tomcat 7.0.22		●			●
	Download	Download	Download	Download	Download
	Free, 73 MB	Free, 167 MB	Free, 49 MB	Free, 47 MB	Free, 257 MB

You can also install a lighter version of NetBeans and add the required JavaME plugin after installation via the NetBeans Tools → Plugins menu.

When that option is selected, you must select the correct plugin in the “Availlable Plugins” tab.



Select	Name	Category	Active
☐	NetBeans Plugin Development	Developing NetBeans	✓
☐	JavaFX 2 Support	JavaFX 2	✓
☐	Mobility	Java ME	✓
☐	Java ME SDK Tools	Java ME SDK Tools	✓
☐	Java	Java SE	✓
☐	Java Persistence	Java SE	✓

A special Java ME Platform, must be configured in NetBeans, to support the specific eWON target. This is also done via the “Tools” menu from the NetBeans IDE.

NOTE

Eclipse users: Although eclipse is a fine environment for JAVA development, we checking the compliance of the eWON toolkit with that environment, though you can try and use it.

NOTE

NetBeans version and this documentation: Some screenshots in this documentation may be related to previous versions of NetBeans as long as it remains applicable.

Typical development work flow

- Develop and build your application with NetBeans.
- Download your application from NetBeans using FTP transfert.
- Debug you application with NetBeans using all facilities:
  - breakpoint
  - watches
  - step-by-step

    - ...
- Download your final JAR in the eWON /usr partition
- Create a *jvmrun* text file in the ewon /usr directory to autorun your JAVA application at eWON boot time.

Installing the JAVA environment

NOTE

Even if another JAVA JDK or Netbeans version is installed, these eWON compatible versions can be installed as well. Multiple versions can coexist and have their own configuration.

The JAVA IDE recommended and supported for developing J2ME applications on the eWON is **NetBeans** (www.netbeans.org).

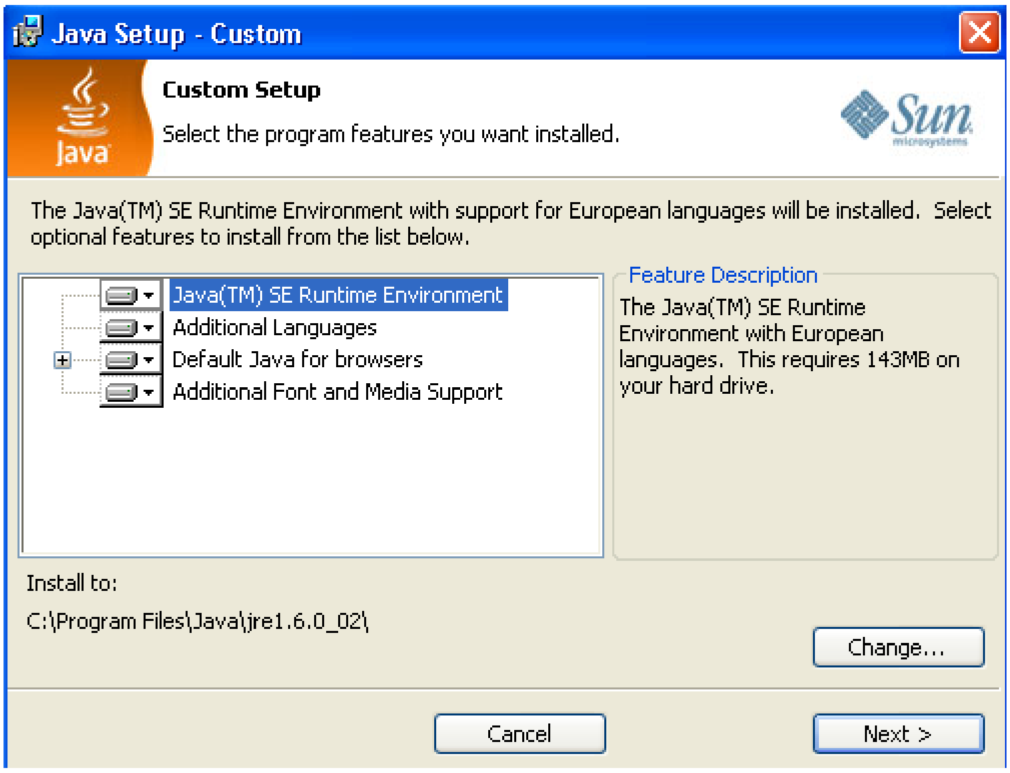
Please install the NetBeans IDE with Java ME plugin before you can start installing the eWON JAVA Toolkit.

When this environement is installed you will have to install the JAVA ETK itself.

Install JDK

If you want to use the NetBeans environment you will at least need a 1.6 compliant JDK environment.

Launch



The Java(TM) SE Runtime Environment with support for European languages will be installed. Select optional features to install from the list below.

- Java(TM) SE Runtime Environment
- Additional Languages
- Default Java for browsers
- Additional Font and Media Support

Feature Description

The Java(TM) SE Runtime Environment with European languages. This requires 143MB on your hard drive.

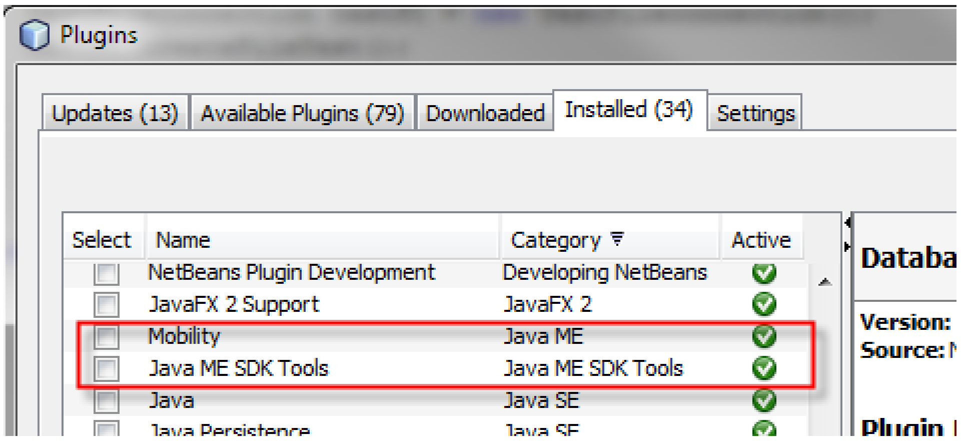
Install to:

C:\Program Files\Java\jre1.6.0_02\

Install Netbeans

Either install the full NetBeans bundle, or install a basic NetBeans version and add the Jave ME plugins:

From the Tools → Plugins menu:



Select	Name	Category	Active
☐	NetBeans Plugin Development	Developing NetBeans	✓
☐	JavaFX 2 Support	JavaFX 2	✓
☐	Mobility	Java ME	✓
☐	Java ME SDK Tools	Java ME SDK Tools	✓
☐	Java	Java SE	✓
☐	Java Persistence	Java SE	✓

Running NetBeans

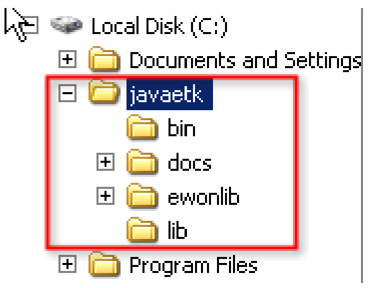
Netbeans can be launched with the NetBeans icon or from the start menu.

If you have previous settings you are proposed to import them, but starting from scratch is recommended for compatibility with this manual.

Install the eWON JTK

Install the *javaetk* toolkit on your system

javaetk is the eWON Java toolkit, it is provided as a ZIP file with a tree of files that must be extracted at some place on your system.

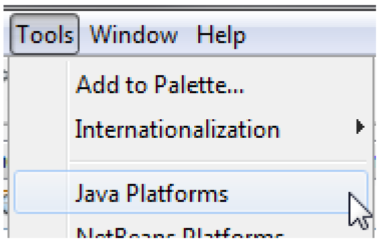


Configure the eWON toolkit in NetBeans

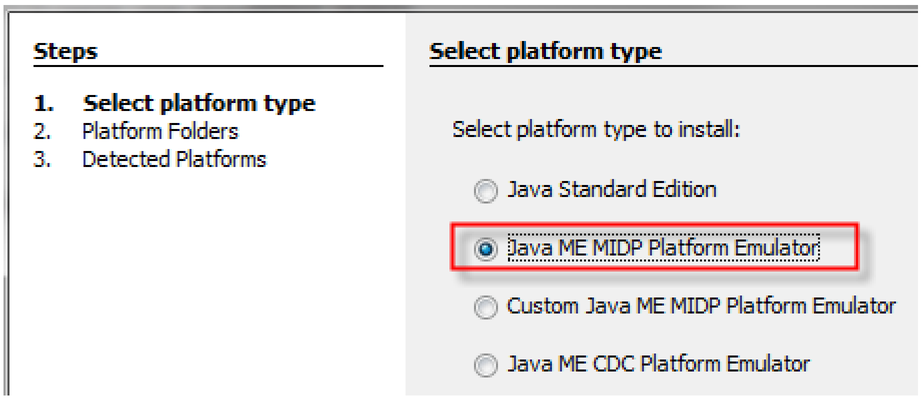
Inside Netbeans, you must install support for creating applications for the eWON. This support is provided by the JAVA eWON Toolkit (javaetk).

From the Netbeans editor, select

Tools → Java Platform



Then from the main dialog, select installation click “Add Platform” button:


Steps
1. Select platform type
2. Platform Folders
3. Detected Platforms
Select platform type
Select platform type to install:
Java Standard Edition
Java ME MIDP Platform Emulator
Custom Java ME MIDP Platform Emulator
Java ME CDC Platform Emulator

Then select installation of a Java ME MIDP Platform Emulator (though eWON is not MIDP, but IMPNG):

In the next step, you need to browse your disk and select the folder where you have unziped the javaetk.

NOTE The platform name is part of the project parameters. So although you can choose any name, it is better to keep the same platform name (***javaetk***), so that you don't need to reconfigure the project when moving between Netbeans installations.

Develop a JAVA application for eWON

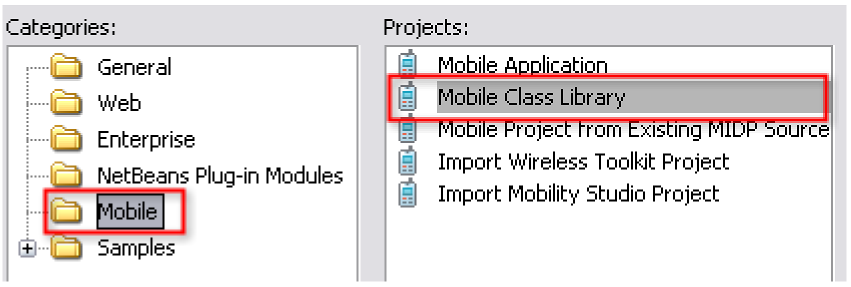
The “Hello World” application is a simple example that shows how to create, build, debug and execute a project. We will explain step-by-step how to proceed in this chapter.

Create the “Hello World” project

From the Netbeans IDE, select:

File → New project...

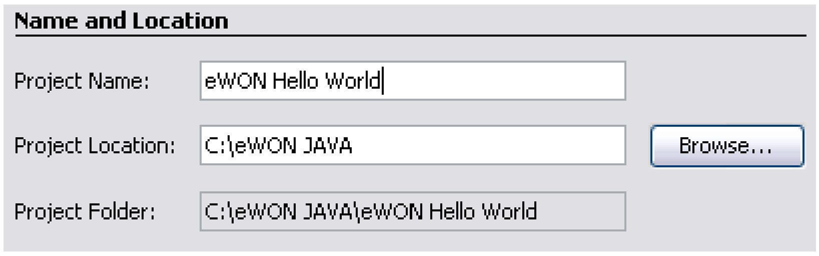
Then, from the project Wizard, select “Mobile” and “Mobile Class Library”



NOTE As explain in “J2ME deviations in the eWON“ at page 25, the eWON does not use midlet, this is why you **need** to create a “Mobile Class Library” and not a “~~Mobile Application~~”

At next Wizard step, select a project name and a folder to locate your project root.

With Netbeans, all files of the project will be stored in a folder tree with the same name as the project.



Name and Location

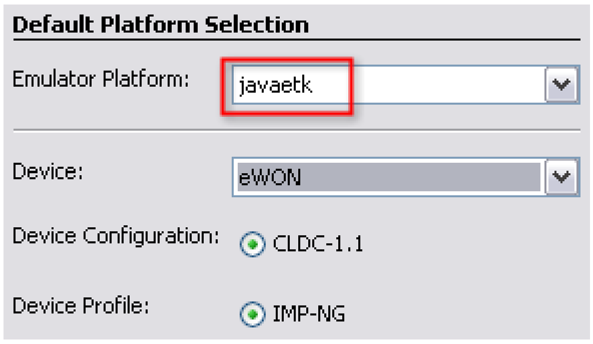
Project Name: eWON Hello World

Project Location: C:\eWON JAVA

Project Folder: C:\eWON JAVA\eWON Hello World

At next Wizard step, select the eWON's "Emulator Platform": ***javaetk***

The other options are implied by this choice.



Default Platform Selection

Emulator Platform: javaetk

Device: eWON

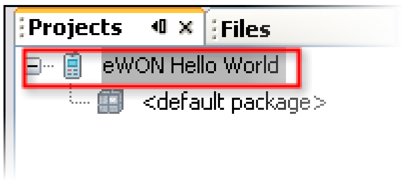
Device Configuration: CLDC-1.1

Device Profile: IMP-NG

NOTE

eWON currently provides a **CLDC 1.1, IMP-NG 1, JSR-075** compatibility. This implies a set of JAVA libraries available for development.

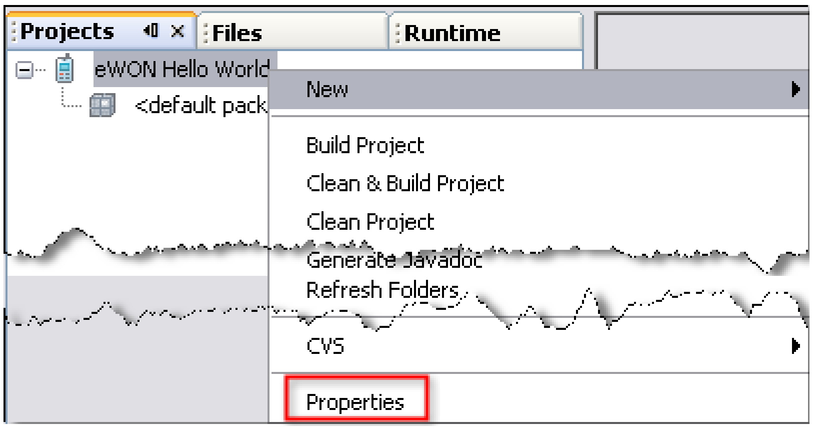
When the project Wizard ends, you will find your empty project in the "Projects" tree.



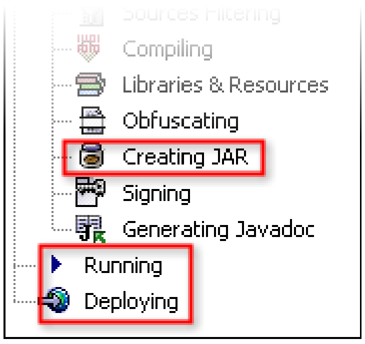
Configure the "Project Properties"

The project properties must be setup correctly to enable download and debug of the application in the eWON.

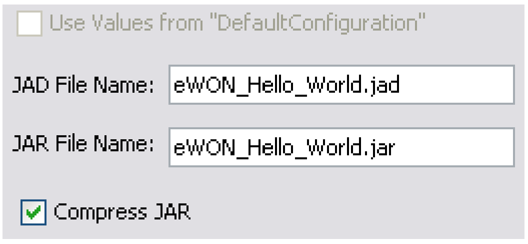
From the "Projects" tree, right click on the project name, and select the "Properties" from the context menu.



From the properties panel, only 3 panels require configuration: "Creating JAR", "Running", "Deploying"



Creating JAR



☐ Use Values from "DefaultConfiguration"

JAD File Name: eWON_Hello_World.jad

JAR File Name: eWON_Hello_World.jar

✓ Compress JAR

This panel does not actually requires any modification, but it is interesting to see that during the project build process, 2 final files will be created:

eWON_Hello_World.jad and **eWON_Hello_World.jar**

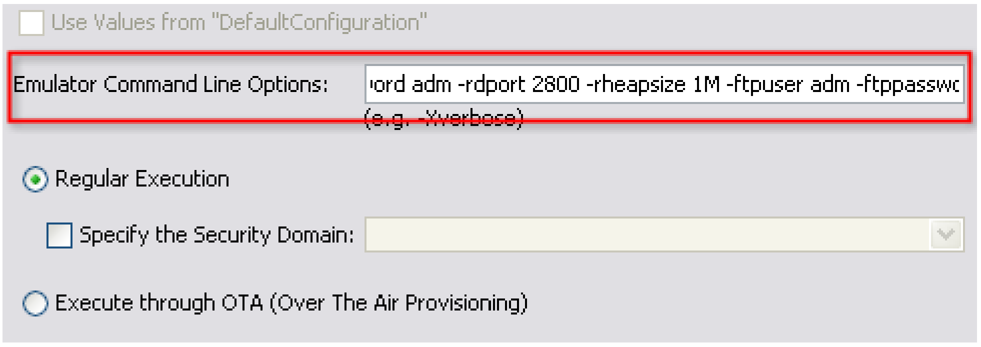
All J2ME application contains these 2 files which need to be downoaded to the eWON for execution.

Running

This panel is used to configure the eWON Emulator.

The emulator is an program used by all J2ME IDE environments (not only Netbeans) to create an interface between the IDE and the eWON itself.

For more information about the eWON Emulator program, see "eWON "Emulator" reference" on page 26.


Use Values from "DefaultConfiguration"
Emulator Command Line Options: ord adm -rdport 2800 -rheapsize 1M -ftpuser adm -ftppasswc

Regular Execution
Specify the Security Domain:
Execute through OTA (Over The Air Provisioning)

For this example application enter the following emulator line:

```
-ewonipaddr 10.0.0.53 -rclasspath /usr/eWON_Hello_World.jar -remain HelloMain
-httpport 80 -httpuser adm -httppassword adm -rdport 2800 -rheapsize 1M -ftpuser adm
-ftppassword adm
```

Important: 10.0.0.53 must be replaced by your eWON IP address. This example also supposes that the administrator default username and password has not been changed.

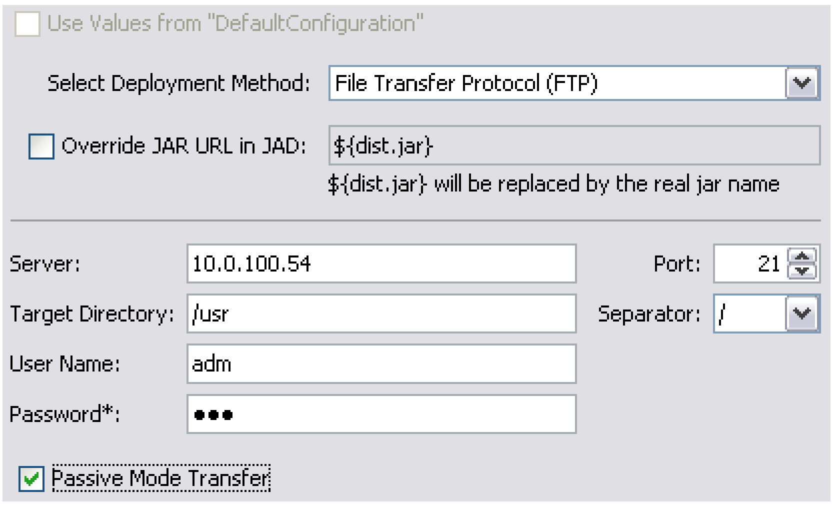
Explanation of the parameters for this example:

-rclasspath /usr/eWON_Hello_World.jar	This is the JAR containing the application. It is passed as an argument to the eWON for execution.
-remain HelloMain	This is the JAVA class containing the **main** function. You will create it in the next step.
-httpuser adm	EWON user for remote JAVA execution: adm. In the eWON configuration, the user must have ✓ Control Java JVM user's right to control JAVA.
-httppassword adm	Password of the administrator
...	see "eWON "Emulator" reference" on

	page 26.

Deploying

The deploying panel is used to configure application's FTP transfer to the eWON.



☐ Use Values from "DefaultConfiguration"

Select Deployment Method: File Transfer Protocol (FTP)

☐ Override JAR URL in JAD: ${dist.jar}

${dist.jar} will be replaced by the real jar name

Server:	10.0.100.54	Port:	21
Target Directory:	/usr	Separator:	/
User Name:	adm		
Password*:	•••		

☑ Passive Mode Transfer

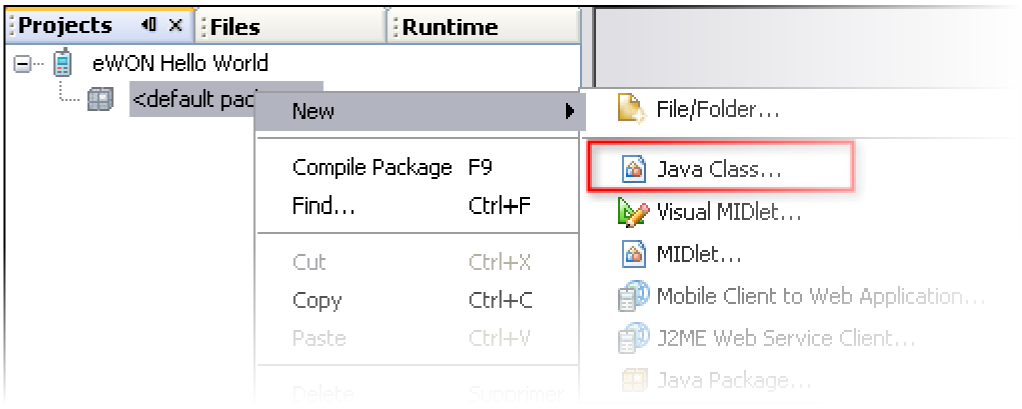
All parameters should be configured according to the eWON's user used for transfer.

The “Target Directory” selected here can be changed to any other directory, but in that case, the emulator command must be changed also (see **-rclasspath**).

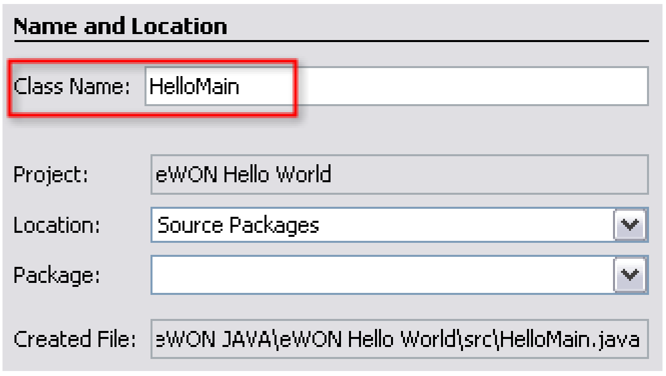
Create your JAVA project

Until now we have only created the project container and confiugred it, we will now actually create the JAVA code.

From the “Projects” panel, select the <default package> and rght click to create a new “Java Class”



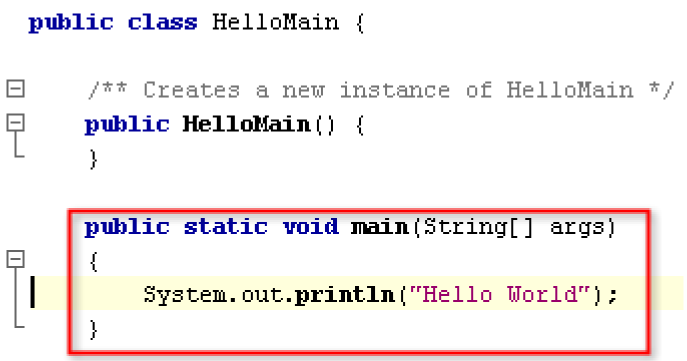
And call it **HelloMain**


Name and Location

Class Name:	HelloMain
Project:	eWON Hello World
Location:	Source Packages
Package:	
Created File:	ɜWON JAVA\eWON Hello World\src\HelloMain.java

For the purpose of this example, the application will only output a single "Hello World" to the eWON real time event.

Enter the following code in editor.

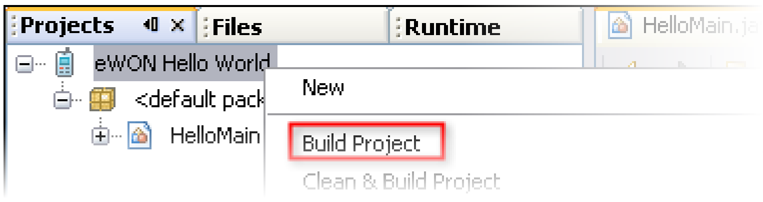


```
public class HelloMain {

    /** Creates a new instance of HelloMain */
    public HelloMain() {
    }

    public static void main(String[] args)
    {
        System.out.println("Hello World");
    }
```

Only the surrounded part has to be typed, the rest is generated automatically by Netbeans class wizard.

Build your project

From the Project panel again, right click on the project name and select "Build"



REM: you can also select Clean & Build Project.

Build execution will take place and a build report is produced in the "Output" panel

Bellow is a typical output:

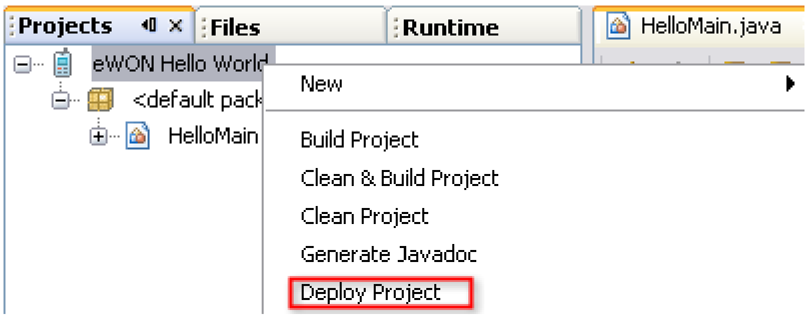
```
Building jar: C:\eWON JAVA\eWON Hello World\dist\eWON_Hello_World.jar
Updating application descriptor: C:\eWON JAVA\eWON Hello World\dist\eWON_Hello_World.jad
Application descriptor does not declare any MIDlet. Direct execution is not allowed.
Generated "C:\eWON JAVA\eWON Hello World\dist\eWON_Hello_World.jar" is 817 bytes.
post-jar:
BUILD SUCCESSFUL (total time: 0 seconds)
```

The red warning reported is not relevant with eWON, as already explained, the eWON does not use midlet, Netbeans considers this as a problem but it is not.

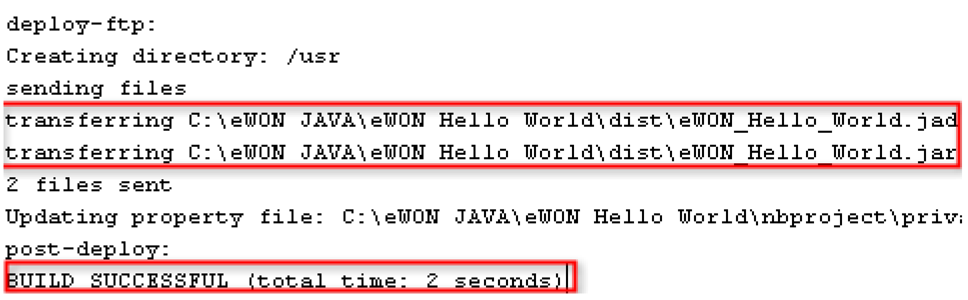
"Build Successful" on the other hand is required.

Transfer application to the eWON

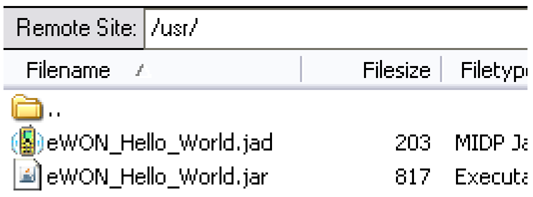
From the Project panel, right click on the project and select "Deploy Project"



The Output panel will report the transfer status. You can control the transfer of both JAR and JAD files:


```
deploy-ftp:
Creating directory: /usr
sending files
transferring C:\eWON JAVA\eWON Hello World\dist\eWON_Hello_World.jad
transferring C:\eWON JAVA\eWON Hello World\dist\eWON_Hello_World.jar
2 files sent
Updating property file: C:\eWON JAVA\eWON Hello World\nbproject\priv
post-deploy:
BUILD SUCCESSFUL (total time: 2 seconds)
```

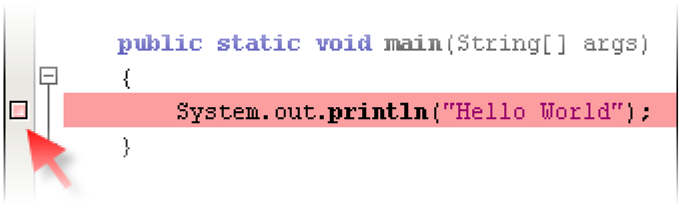
Using an FTP client you can log on the eWON and check in the /usr directory that both file are present:


```
Remote Site: /usr/
Filename            Filesize  Filetyp
..
eWON_Hello_World.jad    203   MIDP Ja
eWON_Hello_World.jar    817   Executa
```

Debug application

First place a breakpoint in the program on the System.err.println("Hello World") line.

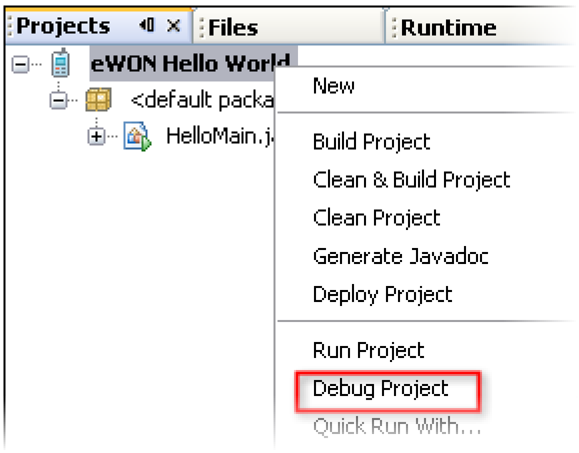
Click in the margin to toggle the breakpoint:


```
public static void main(String[] args)
{
    System.out.println("Hello World");
}
```

IMPORTANT Always place breakpoint on valid lines, any remove any breakpoint in files which are not up to date or not in your project.

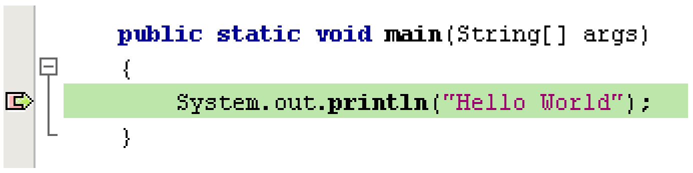
The system behave strangely in that case. Typically, the debug starts and then stop immediately with SUCCESS, yet the program never starts.

From the Project panel, right click on the program name and select “Debug Project”



NOTE Project execution takes some time to start. Don't worry.
In case there is a problem, check the eWON's events.txt file and the eWON's Realtime log files. You should have more information about the failure reason.
You should also check the Netbeans “Output” panel for diagnosis.

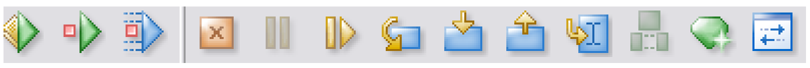
In case all works, you should end on your breakpoint



```
public static void main(String[] args)
{
    System.out.println("Hello World");
}
```

You can inspect variables, step in your program install new breakpoints.

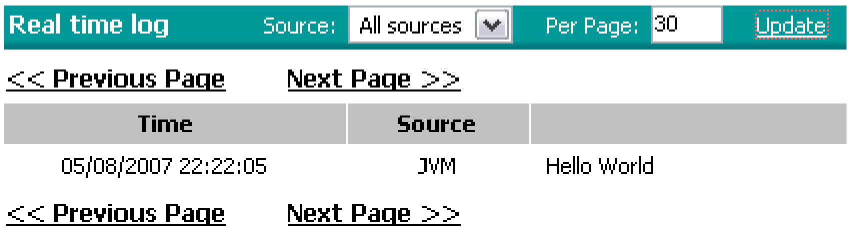
Move your mouse on the debug toolbar to check all the options



IMPORTANT The emulator will lways try to restart the program in case you launch a new debug session. Yet, it is always a good idea to “Reset” a running program before you start a new one.

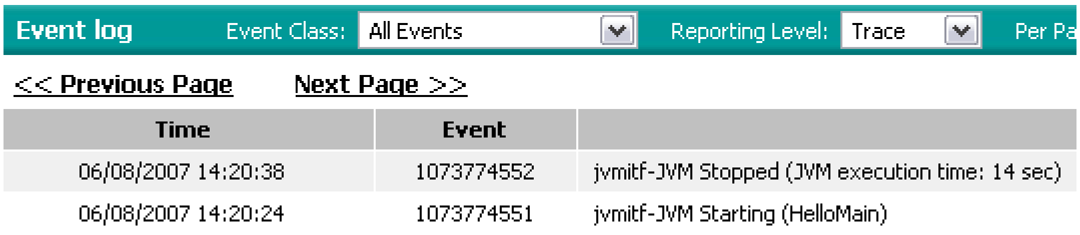
Now click on the “Continue” ( ) button, an check the eWON “Real time log”.

You will see the System.out.println



Time	Source	
05/08/2007 22:22:05	JVM	Hello World

In the eWON's events log, you can also find information about the JVM execution:



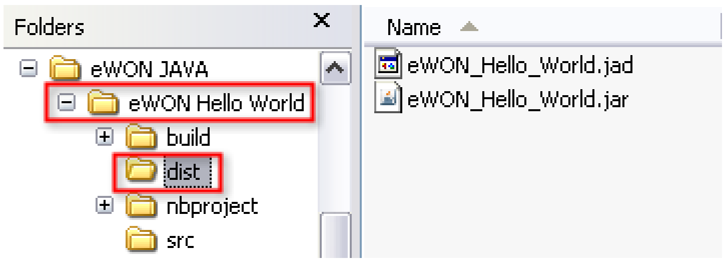
Time	Event	
06/08/2007 14:20:38	1073774552	jvmitf-JVM Stopped (JVM execution time: 14 sec)
06/08/2007 14:20:24	1073774551	jvmitf-JVM Starting (HelloMain)

Transfer application manually in the eWON (no IDE)

At one time in the development cycle you will need to get rid of the IDE, when the application development is finished and your application must work in production.

When you have executed the “Build Project” in Netbeans (see Build your project on page 16), Netbeans has produced the 2 JAR and JAD files.

If you look at the directory structure produced by Netbeans, you will find a “dist” directory in your project directory.



The “dist” (for distribute) directory contains the final output files of your project.

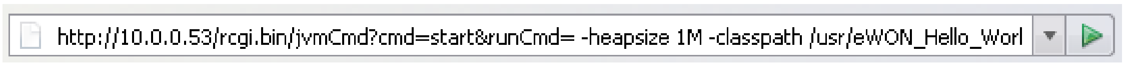
You can download these files manually to the eWON /usr (or any /usr subdirectory you choose) and then execute this application.

Execute application manually using Web interface

JAVA execution can be manually started through the eWON web interface by calling a specific web form.

This mode of operation is normally reserved to Netbeans when the Netbeans IDE needs to start or stop the application remotely.

You can start execution of the JVM by typing the execution URL directly in a Web browser:



```
http://10.0.0.53/rcgi.bin/jvmCmd?cmd=start&runCmd= -heapsize 1M -classpath /usr/eWON_Hello_Worl
```

To run the Hello World application, the following command must be typed:

```
http://10.0.0.53/rcgi.bin/jvmCmd?cmd=start&runCmd= -heapsize 1M -classpath
/usr/eWON_Hello_World.jar -emain HelloMain
```

Important: 10.0.0.53 must be replaced by your eWON IP address.

The Web form is **jvmCmd** and the parameters are:

- cmd=start: start the JVM
- runCmd= -heapsize......: arguments to pass to the JVM as follow:

-heapsize 1M	Memory allocated to JVM
-classpath /usr/eWON_Hello_World.jar	Path to application classes repository
-emain HelloMain	Name of class with main function.

This is a short summary of the JVM possible arguments, please see “JAVA Virtual Machine (JVM) arguments“ on page 30 for more informations on JVM parameters.

Configure application execution at eWON boot time

During eWON startup, the eWON search for a file calld **jvmrun** in the following directories:

- /dev
- /dev/oem
- /sys

- **/usr**

Actually, /usr is the more common place as the /dev are reserved for manufacturer applications and /sys is normally reserved for eWON configuration.

“jvmrun” syntax

The jvmrun file is a text file with one or more lines.

If file's line starts with a # character, it is considered as comment line and is skipped.

The file's line not starting with a # is considered as the JVM Run Command and is passed for JVM execution.

See “JAVA Virtual Machine (JVM) arguments“ on page 30 for more informations on JVM parameters.

Example of jvmrun file content:

```
#jvmrun file, place it in /usr
#This file will trigger execution of the JVM at boot time
-watchon -heapsize 1M -classpath /usr/eWON_Hello_World.jar -emain HelloMain
```

Create a J2ME application manually

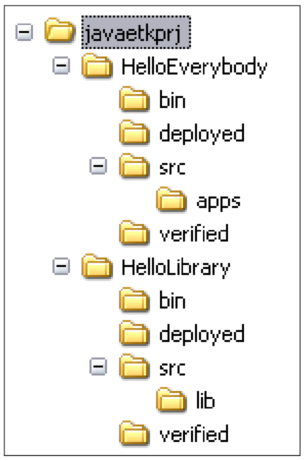
The previous paragraph has explained the normal procedure to build a JAVA project for the eWON.

This paragraph will explain with an example how to create a project with the **SUN**tm **JDK** and the **javaetk** only. This is interesting for those who want to automate some tasks or those who want to understand the process handled internally by Netbeans.

Important: the files described in this example are bundled in the *javaetkprj.zip* file.

Directories layout

In order to simplify the example, the following directory structure has been used.



There are 2 projects: one for the library and one for the application.

In each project we find a number of directories:

- *src*: contains the source tree. The application is in an “*apps*” package. The library is in a “*lib*” package.
- *bin*: contains the compiled files. In J2ME, the files must be verified after they have been compiled. So this directory contains the result of the javac tool but cannot be used directly in the device.
- *verified*: contains the files produced by the *preverifier*. The *preverifier* is provided with the javaetk, it works on the files produced by the *javac* compiler and it produces files that can be used in the J2ME device.
- *Deployed*: this directory will contain the JAR files, these JAR files are produced by the JavaSDK *JAR* tool.
- *build.cmd*: this is an MS-DOS command file that

Build to library and application

Build cmd: introduction

The *build.cmd* is the batch file used to compile the library and the application.

The batch starts with a number of defines to locate your application, your java toolkit (a 1,5 version is used here). The Iavaetk library is also located.

Build library

a. Compile the java files

```
%JSDK%\javac -target 1.4 -source 1.4 -g -classpath %CP% -d %HLIB%\bin %HLIB
%\src\lib\HelloPrint.java
```

Compiler here is version 5 so force 1.4 compatibility.
Classpath: path to the javaetk library (IMPNG classes + eWON). When building the *apps* we will add the HelloLibrary to this path.
-d %HLIB%\bin: output directory (unverified class binaries)
%HLIB%\src\lib\HelloPrint.java: path to one java file to compile, must be called for each source file.

b. Verify the compiled class files

```
%JAVAETK%\bin\preverify.exe -classpath %CP% -d %HLIB%\verified %HLIB%\bin
```

-d %HLIB%\verified: output directory. This is the root to all verified class files.
%HLIB%\bin: root of packages to verify. Please note that we do not pass bin\lib. The whole tree is verified.

c. Create JAR file for library

```
%JSDK%\jar cf %HLIB%\deployed\HelloLibrary.jar -C %HLIB%\verified lib
```

%HLIB%\deployed\HelloLibrary.jar: the library we want to build
-C %HLIB%\verified lib: change directory to %HLIB%\verified and add the lib directory to the jar file.
The output of this process is the **HelloLibrary.jar** located in the *deployed* directory.

Build the application

a. Extend classpath

```
set CPAPP=%CP%;%HLIB%\deployed\HelloLibrary.jar
```

Classpath for application must include library we created before also.

b. Build the application JAR

```
%JSDK%\javac -target 1.4 -source 1.4 -g -classpath %CPAPP% -d %HEBD%\bin %HEBD
%\src\apps\HelloEverybody.java

%JAVAETK%\bin\preverify.exe -classpath %CPAPP% -d %HEBD%\verified %HEBD%\bin

%JSDK%\jar cf %HEBD%\deployed\HelloEverybody.jar -C %HEBD%\verified apps
```

The process is exactly the same as for the library. The only difference is the classpath used which includes the library.

The output of this process is the **HelloLibrary.jar** located in the *deployed* directory.

Merge multiple JAR in a single JAR

The *single_jar.cmd* batch file shows how to bundle the *HelloLibrary.jar* and *HelloEverybody.jar* into a single *HelloFull.jar* file.

The command file starts with the same defines defined in the *build.cmd* file.

Then the batch moves to the HelloEverybody *deployed* directory.

Then the following 3 lines:

```
%JSDK%\jar x <%HLIB%\deployed\HelloLibrary.jar
%JSDK%\jar x <%HEBD%\deployed\HelloEverybody.jar
%JSDK%\jar cf HelloFull.jar lib apps
```

Will first extract both JAR files in the deployed directory (rebuilding the trees and merging them).
Then the last line will recreate a *HelloFull.jar* containing the *lib* and the *apps* packages.

J2ME deviations in the eWON

The J2ME in the eWON has some deviations compared to J2ME standard.

The language itself is conform to the J2ME CLDC1.1 and IMP-NG 1.0, all classes defined by these standard and all features of the language are conform.

The specificity is related to the following aspects:

- Application Midlets: A standard J2ME application is derived from the Midlet class, multiple midlets can run at the same time, they can be started or stopped.
  In the eWON there is no midlet. An application is started like a standard JAVA application by passing a class with a main function to the JVM.
  Only one application can run at a given time.
  Threading is supported, so multiple threads can run in the application.

- Application encryption and signature: The eWON does not support the standard application signature with RSA certificates specified for the J2ME platform.
  An equivalent signature mechanism is provided but it is not a standard mechanism with regards to the J2ME standard.

eWON “Emulator” reference

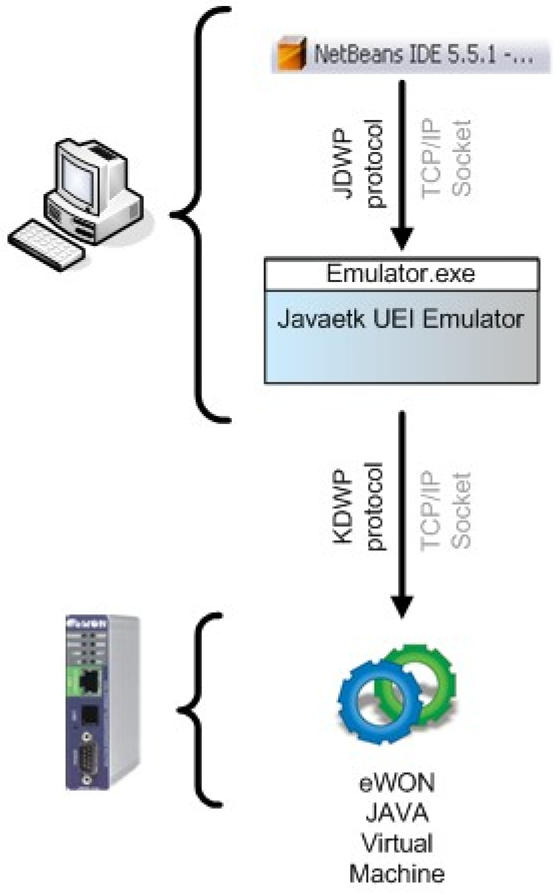
The emulator is a program used by all J2ME IDE environments (not only Netbeans) to create an interface between the IDE and the eWON itself.

Every J2ME device must have their own **emulator.exe** program. This program has a standard command line interface defined in the Universal Emulator Interface[1] specification (see also http://java.sun.com/j2me/docs/uei_specs.pdf)

An UEI emulator is at least defined to provide remote debugging capability between the IDE and the J2ME device.

An emulator program is required for remote debugging because the standard debug interface in JAVA is JPDA using the Java Debug Wire Protocol – JDWP, but J2ME has defined an other debug protocol for resource constrained devices called KDWP.

The emulator will act as a JPDA compliant server for the Netbeans IDE and will connect to the eWON JVM when required using the KDWP protocol.



The javaetk emulator also provides additional functionalities like:

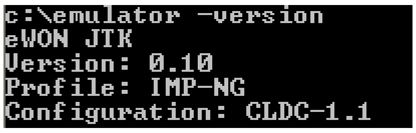
- Start remotely the JAVA application

1 UEI is a standard for interaction between IDEs and device emulators. IDE vendors who implement the UEI specification know that their products will work with a wide variety of device emulators. Device manufacturers who implement the UEI specification in an emulator are assured that their emulator will work with a wide variety of development tools.

- Download the application in the eWON (though FTP)
  REM: in Netbeans there is a “deploy” tool that provides an ftp client, so emulator is not used for application transfer.

The emulator is a command line program, it has not GUI and does not produce any output when called wit no argument.

When “emulator -version” is called, it produces the following:



```
c:\emulator -version
eWON JTK
Version: 0.10
Profile: IMP-NG
Configuration: CLDC-1.1
```

Here are the emulator possible arguments. EWON specific arguments are described here, refer to the UEI specification also for more information about standard paramters.

Argument	**Description**
Standard UEI parameters	
-help	Display simple help
-version	
-Xquery	
-Xdescriptor	
-Xdebug	
Xrunjdwp:...	
Advanced arguments	
-nokill	**advanced** When the emulator program starts, it checks if another occurrence of the program is running and tries and kill this occurrence. This is very useful in case the user starts a new debug session while another is pending. The -nokill argument prevents this behavior. REM: Multiple emulator.exe running may have unpredictable result.
-nojvmstop	**advanced** When the emulator starts a new debug session in the eWON, it will first send a “Stop” to the eWON to prevent the eWON from hanging when a new run is requested. This argument prevents this behavior.
-debugemu	**advanced** emulator will produce a debugemu.txt file in its own directory with debug information. This option may be useful to debug all commands passed by the IDE to the emulator.
-javaproxy	**advanced**

	The emulator can use 2 proxy implementations: ● KVMDebugProxy-win.exe: is a windows application ● kdp.jar: is a JAVA implementation The default is to use the Windows application, but JAVA version can be used instead. REM: the proxy is the actual application that provides the JDWP to KDWP translation.
-ewonipaddr IP	**required** IP is the eWON remote IP address the emulator will use to connect to the eWON.
FTP client: arguments related to the emulator embedded FTP client for program transfer. These arguments can be omitted if FTP transfer is not required.	
-ftpport P	P is the ftp port **Default:** 21
-ftpuser U	U is the user that will be used to connect to the eWON FTP server. User U must have FTP access : **Default:** adm
-ftppassword P	P is the password used by the FTP client to connect to the eWON. **Default:** adm
-ftppassive	If specified, the FTP client with use PASIVE mode for FTP transfert. **Default:** passive mode not used.
-ftpdnl S D	S is the source file path on the local disk D is the destination path on the eWON ftp server -ftpdnl specify one file to transfert. -ftpdnl can by specified up to 10 times in the command line, so up to 10 files can be transfered. If -ftpdnl is not specified in the command line, thenthe FTP transfer operation is skipped, and none of FTP Client parameters are used.
HTTP client argument: The emulator uses HTTP for JVM remote commands (start, stop) invocation, so these parameters are required.	
-httpport P	HTTP port to communicate with the eWON **Default:** 80
-httpuser U	U is the user used for HTTP forms invocations. User U must “Control Java JVM” access: **Default:** adm
-httppassword P	P is the password for the httpuser. **Default:** adm
Remote JVM configuration parameters: As described in “JAVA Virtual Machine (JVM) arguments“ on page 30, the JVM need a number of arguments to run.	

These arguments are forwarded by the emulator.	
-rheapsize S	Heap size allocated to the remote JVM (S is passed as-is to -heapsize parameter)
-rclasspath CP	Class path passed to the remote JVM (CP is passed as-is to the -classpath parameter)
-remain E	Class containing the main function, passed to the remote JVM (E is passed as-is to the -emain parameter)
-rdport P	P is the remote port passed to the eWON for debug connection (passed to the -port parameter)
-classpath CP -pclasspath PCP	CP;PCP is a path passed to the emulator debug proxy as -cp parameter. The Debug proxy needs a complete path to all the classes used by the remote application to provide the debug feature. There is no distinction between these two arguments, they are proposed for compatibility with the various IDE and debug situations. REM: the application JAR file passed as -Xdescriptor argument (implied by .JAD) is automatically added to the proxy classpath.

Example arguments passed to emulator during the Hello World debug session:

```
Arg[0]: C:\javaetk\bin\emulator.exe
Arg[1]: -Xdevice:eWON
Arg[2]: -Xdescriptor:C:\eWON JAVA\eWON Hello World\dist\nbrun6653\eWON_Hello_World.jad
Arg[3]: -Xdebug
Arg[4]: -Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=1336
Arg[5]: -ewonipaddr
Arg[6]: 10.0.0.53
Arg[7]: -rclasspath
Arg[8]: /usr/eWON_Hello_World.jar
Arg[9]: -remain
Arg[10]: HelloMain
Arg[11]: -httpport
Arg[12]: 80
Arg[13]: -httpuser
Arg[14]: adm
Arg[15]: -httppassword
Arg[16]: adm
Arg[17]: -rdport
Arg[18]: 2800
Arg[19]: -rheapsize
Arg[20]: 1M
```

JAVA Virtual Machine (JVM) arguments

This appendix describes the parameters available to configure JVM execution.

This list of arguments is also called the **JVM Run Command**

The JVM execution can be started by one of the following means:

- Using the eWON HTTP server by using the jvmForm (see “Configure application execution at eWON boot time“ on page 20).
- By creating a jvmrun file in the eWON /usr root directory (see “Configure application execution at eWON boot time“ on page 20).
- By using the com.ewon.ewonitf.RuntimeControl.configureNextRunCommand JAVA function (see JAVA Doc).

When the JVM is started it receives a number of argument to configure JVM execution. The following arguments are defined.

-watchon	This command is related to the eWON JAVA watchdog mechanism. When watchon is specified, the JAVA watchdog is enabled with a timeout of 1 minute. The watchdog can be reconfigured (or disabled) with the functions in com.ewon.ewonitf.RuntimeControl, but if it is not reconfigured, com.ewon.ewonitf.RuntimeControl.**refreshWatchdog** must be called by the user's application at least one minute after the JVM was started. Please see the JAVA Doc for more information about watchdog. **Default:** no watchdog
-debugger	This option is used with the eWON must listen for a debugger remote connection. This function is normally only used by Netbeans. REM: this option implies the -suspend option bellow. If suspend is not required, -nosuspend must be specified. **Default:** no debugger
-suspend	This option force the JVM to be suspend after startup to wait for debugging connection. This is only used with -debugger and is implied by the -debugger option. **Default:** true if -debugger specified, false otherwise.
-nosuspend	This option will release execution of the program when -debugger option is specified and will not stop the JVM until the remote debugger is connected. **Default:** false
`-port N`	N is the port number on which the eWON JVM will wait for a remote debugging connection. During normal debugging this port is managed and defined by Netbeans, but in case the user wants to for a specific port, this option can be used. This option is only relevant if -debugger is specified. **Default:** 2800

-heapsize N -heapsize Nk -heapsize Nm	N is the amount of memory allocated to JVM execution. This memory will be used to store JAVA classes and user allocate variables. The trailing k or m can be used to specify the heap size in kilobytes or megabytes. A **1MByte** size is a minimum for a normal execution. Out of memory errors (exceptions) will be reported during JVM execution in case the heap size is too small. **Default:** 64KBytes
-classpath CP	C is the eWON JVM classpath. (CP should not contain any space) The classpath is a list of JAR files or class files separated by ; Each path must be an absolute path. All paths are case sensitive. Classpath maximum length is 1048 characters. Example of valid classfiles: /usr/MyApp.jar /usr/MyApp.jar;/usr/lib/MyLib.jar /usr/MyApp.class;/usr/lib/MyLib.jar **Default:** none
-emain MC	MC This is the name of the JAVA class of the user's application containing the main function. Only the name of the class must be specified and is case sensitive. **Default:** none