# SIEMENS

# JAVA User's Guide

**Siemens Cellular Engine**

**Supported products: TC65, TC65 Terminal, AC75**

JAVA ™ Users Guide

| | |
|---|---|
| Document Name: | **JAVA User's Guide** |
| Supported products: | **TC65, TC65 Terminal, AC75** |
| Version: | **08** |
| Date: | **June 12, 2006** |
| DocId: | **TC65_AC75_JAVA User's Guide_V08** |
| Status: | **Confidential / Released** |

**General Notes**

Product is deemed accepted by recipient and is provided without interface to recipient's products. The documentation and/or product are provided for testing, evaluation, integration and information purposes. The documentation and/or product are provided on an "as is" basis only and may contain deficiencies or inadequacies. The documentation and/or product are provided without warranty of any kind, express or implied. To the maximum extent permitted by applicable law, Siemens further disclaims all warranties, including without limitation any implied warranties of merchantability, completeness, fitness for a particular purpose and non-infringement of third-party rights. The entire risk arising out of the use or performance of the product and documentation remains with recipient. This product is not intended for use in life support appliances, devices or systems where a malfunction of the product can reasonably be expected to result in personal injury. Applications incorporating the described product must be designed to be in accordance with the technical specifications provided in these guidelines. Failure to comply with any of the required procedures can result in malfunctions or serious discrepancies in results. Furthermore, all safety instructions regarding the use of mobile technical systems, including GSM products, which also apply to cellular phones must be followed. Siemens or its suppliers shall, regardless of any legal theory upon which the claim is based, not be liable for any consequential, incidental, direct, indirect, punitive or other damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information or data, or other pecuniary loss) arising out the use of or inability to use the documentation and/or product, even if Siemens has been advised of the possibility of such damages. The foregoing limitations of liability shall not apply in case of mandatory liability, e.g. under the German Product Liability Act, in case of intent, gross negligence, injury of life, body or health, or breach of a condition which goes to the root of the contract. However, claims for damages arising from a breach of a condition, which goes to the root of the contract, shall be limited to the foreseeable damage, which is intrinsic to the contract, unless caused by intent or gross negligence or based on liability for injury of life, body or health. The above provision does not imply a change on the burden of proof to the detriment of the recipient. Subject to change without notice at any time. The interpretation of this general note shall be governed and construed according to German law without reference to any other substantive law.

**Copyright**

Transmittal, reproduction, dissemination and/or editing of this document as well as utilization of its contents and communication thereof to others without express authorization are prohibited. Offenders will be held liable for payment of damages. All rights created by patent grant or registration of a utility model or design patent are reserved.

Copyright © Siemens AG 2006

**Trademark notices**

MS Windows® is a registered trademark of Microsoft Corporation.
Java™ and Sun™ NetBeans IDE 5.0 are registered trademarks of Sun Microsystems Inc.
Borland® JBuilder® is a registered trademark of Borland Software Corporation

# Table of Contents

## Figures

## Tables

# 1 Preface

This document covers the full range of IMP-NG Java products from Siemens, currently including:

- TC65 Module
- TC65 Terminal
- AC75 Module

Differences between the products are noted in the particular chapters. Throughout the document, all supported products are referred to as ME (Mobile Equipment). For use in file, directory or path names, the string "<productname>" represents the actual name of a product, for example TC65. Screenshots are provided as examples and, unless otherwise stated, apply to all supported products.

# 2 Overview

The ME features an ultra-low profile and low-power consumption for data (CSD and GPRS), voice, SMS and fax. Java technology and several peripheral interfaces on the module allow you to easily integrate your application.

This document explains how to work with the ME, the installation CD and the tools provided on the installation CD.
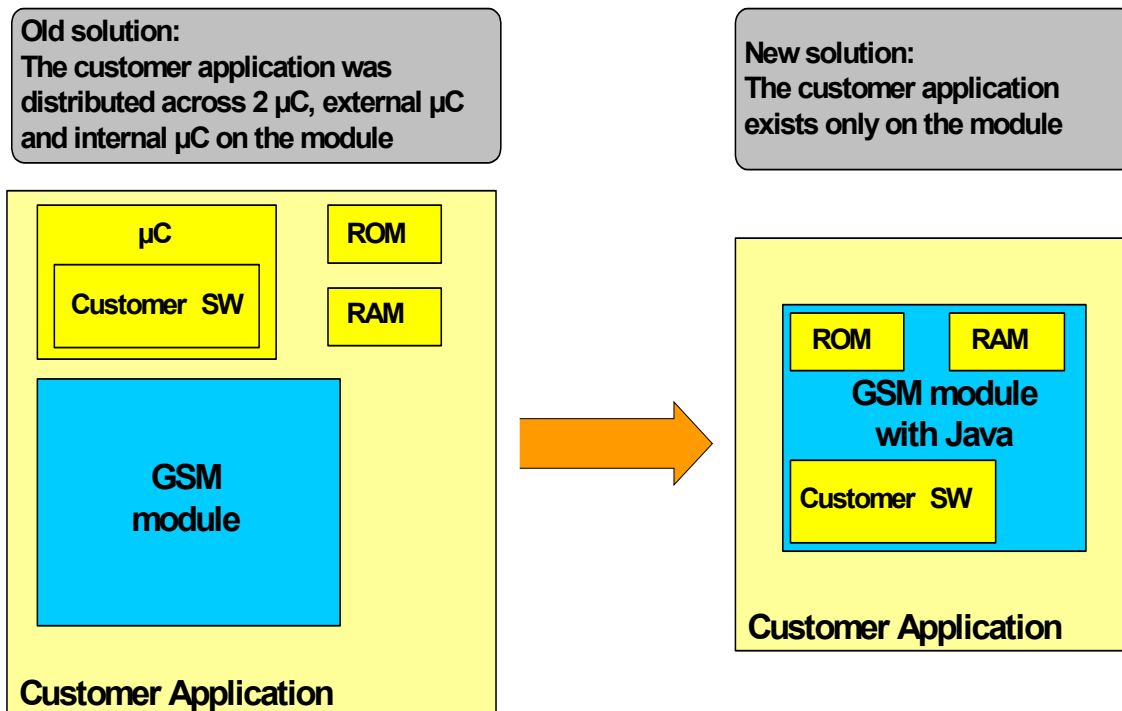


Figure 1: Overview

## 2.1 Related Documents

[1] Multiplexer Installation Guide

[2] DSB75 Support Box - Evaluation Kit for Siemens Cellular Engines

[3] AT Command Set of your Siemens Wireless product

[4] Hardware Interface Description of your Siemens Wireless product

[5] Java doc \wtk\doc\html\index.html

[6] IMP-NG, JSR228, Standard

[7] Application Note 24: Application Developer's Guide

## 2.2     Terms and Abbreviations

| Abbreviation | Description |
|---|---|
| API | Application Program Interface |
| ASC | Asynchronous Serial Controller |
| CLDC | Connected Limited Device Configuration |
| CSD | Circuit-Switched Data |
| DAI | Digital Audio Interface |
| DCD | Data Carrier Detect |
| DSR | Data Set Ready |
| GPIO | General Purpose I/O |
| GPRS | General Packet Radio Service |
| GPS | Global Positioning System |
| HTTP | Hypertext Transfer Protocol |
| I/O | Input/Output |
| IDE | Integrated Development Environment |
| IP | Internet Protocol |
| J2ME™ | Java 2 Mobile Edition |
| J2SE™ | Java 2 Standard Edition |
| JAD | Java Application Description |
| JAR | Java Archive |
| JDK | Java Development Kit |
| JVM | Java Virtual Machine |
| LED | Light Emitting Diode |
| ME | Mobile Equipment |
| MES | Module Exchange Suite |
| MIDP | Mobile Information Device Protocol |
| OTA | Over The Air |
| OTAP | Over The Air Provisioning of Java Applications |
| PDP | Packet Data Protocol |
| PDU | Protocol Data Unit |
| SDK | Standard Development Kit |
| SMS | Short Message Service |
| SMTK | Siemens Mobile Toolkit |
| TCP | Transfer Control Protocol |
| URC | Unsolicited Result Code |
| URL | Universal Resource Locator |
| VBS | Visual Basic Script |
| WTK | Wireless Toolkit |

# 3 Installation

## 3.1 System Requirements

The Siemens Mobility Toolkit (SMTK) requires that you have:
1. Windows 2000 or Windows XP installed
2. 40Mbytes free disk space for SMTK
3. Administration privileges
4. Java 2 SDK, Standard Edition 1.4. To install the JDK version 1.4.2_09 provided, follow the instructions in Section 3.4.1.

If a Java IDE such as NetBeans IDE 5.0, Eclipse 3.0.1, Eclipse 3.0.2, Eclipse 3.1.0, Eclipse 3.1.1, Eclipse 3.1.2, JBuilder X, 2005 or 2006 Enterprise is installed, it can be integrated into the SMTK environment during the installation of the SMTK. To install one of the IDEs, follow the installation instructions in Section 3.4.3 and Section 3.4.4 respectively.

## 3.2 Installation CD

The Siemens Mobility Toolkit Installation CD includes:
- Module Exchange Suite
- WTK
  bin
    - various tools
  doc
    - html
    - java docs for APIs
  lib
    - classes.zip
  src
    - various examples
- Java SDK
  J2sdk-1_4_2_09-windows-i586-p.exe
- NetBeans IDE 5.0
  netbeans-5_0-windows.exe (NetBeans IDE 5.0)
  netbeans_mobility-5_0-win.exe (NetBeans Mobility package 5.0)
- Eclipse 3.1.2
  eclipse-SDK-3.1.2-win32.zip        (Eclipse 3.1.2)
- EclipseME plugin 1.2.3 [1]
  eclipseme.feature_1.2.3_site.zip  (Eclipse ME Plugin 1.2.3)
- EclipseME plugin WM (only required for Eclipse 3.0.1 and Eclipse 3.0.2)
- Documents:
  DSB75_HW_Description.pdf
  <productname>_AT_Command_Set.pdf
  <productname>_HW_Description.pdf
  <productname>_HW_Description.pdf
  <productname>_ReleaseNote.pdf
  WM_AN_24_Dev_Guide.pdf
  <productname>_Java_UserGuide.pdf (this document)

Some of the content can only be accessed after the installation.

---

[1] EclipseME plugin 1.2.1 is also supported, but not distributed as part of the CD.

## 3.2.1 Components

### 3.2.1.1 Module Exchange Suite

The Module Exchange Suite allows the developer to access the Flash file system on the cellular engine from the development environment over a serial interface. File transfers from PC to module are greatly facilitated by this suite.

The Module Exchange Suite (MES) will be installed under the following directories:
- MES executables (e.g. MESCopy.exe, etc.) in the windows system directory
  (e.g. C:\WINDOWS\system32)
  The version of the MES executables can be read out by using the Windows Explorer context menu -> Properties and selecting the Version tab (e.g. MESCopy.exe Version 1.0.0.12, etc.)
- MES server files under "\ModuleExchange" directory (e.g. C:\Program Files\Module-Exchange)
  The version of the dll files can be read out by using the Windows Explorer context menu -> *Properties* and selecting the Version tab (e.g. MESShellExt.dll Version: 2.0.0.19, MESServer.exe Version: 2.0.0.38, MESSearchApp.exe Version: 2.0.0.5    )

The MES installation version can be found under *Control Panel->Add or Remove Programs* selecting the installed MES program and clicking on support information (e.g. version 1.00.0008)

### 3.2.1.2 WTK

wtk is the directory where all the necessary components for product specific Java application creation and debugging are stored.

The version of the installed WTK is stored under the root of the WTK directory in a text file. (e.g. C:\Program Files\Siemens\SMTK\<product name>\WTK\VersionWTK.txt)

### 3.2.1.3 SDK / JDK

This is software provided by SUN to support Java application development.

SMTK emulator uses the following JDK (stored in JavaSoft Registry key) for starting the Debug Agent:
- Registry path:
  HKEY_LOCAL_MACHINE\SOFTWARE\JavaSoft\Java Development Kit\1.4
- Registry key:
  JavaHome    (e.g. "C:\j2sdk1.4.2_09")
- Registry path:
  HKEY_LOCAL_MACHINE\SOFTWARE\JavaSoft\Java Runtime Environment\1.4
- Registry key:
  JavaHome (e.g. "C:\Program Files\Java\j2re1.4.2_09")

### 3.2.1.4    NetBeans IDE 5.0

This is a Java IDE provided by SUN to support Java application development.


### 3.2.1.5    Eclipse 3.1.2 IDE and Eclipse ME Plugin 1.2.3

This is a Java IDE provided by the Eclipse Foundation to support Java application development.

The installed Eclipse Version and ME Plugin  can be read out after starting Eclipse and selecting:
- Eclipse menu *Help->About Eclipse SDK* and looking for the Eclipse version:
  e.g. Version: 3.1.2 Build id: M20060118-1600
- Eclipse menu *Help->About Eclipse SDK,* press button <Feature Details> and look for the ME Plugin version:
  e.g. EclipseME J2ME Development Tools for the Eclipse Version: 1.2.3)

Used JDK from IDE (e.g. selecting Eclipse menu *Help->About Eclipse SDK->Configuration Details* and look for "java.version" and "java.vm.version")

## 3.3 Setup Java Development Environment with Eclipse IDE (Quick Start-up)

Please complete the following steps to set up your Eclipse Java development environment:

**Step 1:**   Install JDK 1.4.2_09
-> JDK 1.4\j2sdk-1_4_2_09-windows-i586-p.exe

**Step 2:**   Install Eclipse 3.1.2
-> Unzip from CD ("\Eclipse\eclipse-SDK-3.1.2-win32.zip") to
"C:\Program Files\Eclipse"

**Step 3:**   Installation Eclipse ME plugin 1.2.3
- Start Eclipse IDE
- Select the default workspace suggested by Eclipse
- Start ME plugin installer using Eclipse menu
  *Help->Software Updates->Find and Install.*
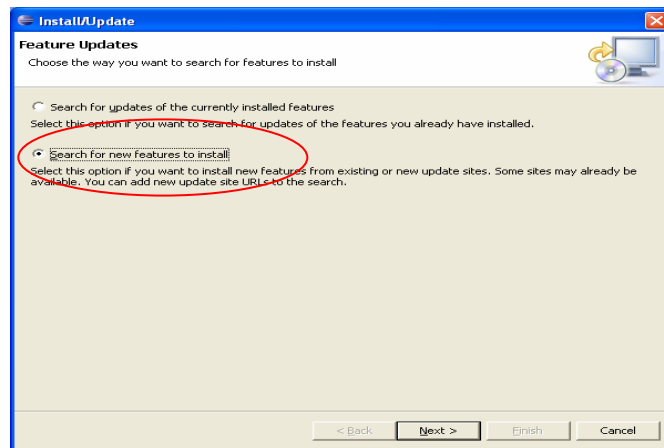


Figure 2: Installation of Eclipse ME-Plugin 1.2.3: Dialog box *Feature Updates*
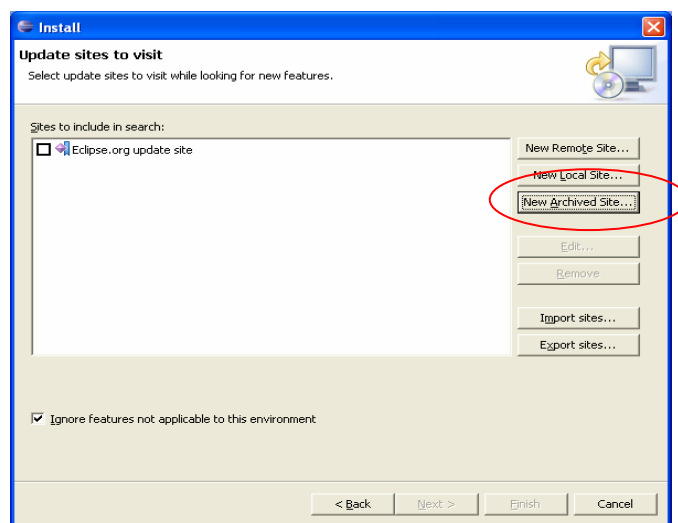


Figure 3: Installation of Eclipse ME-Plugin 1.2.3: Dialog box *Updates sites to visit*

- Select path where Eclipse ME plugin is located on CD
("\EclipseMEplugin_123\.....") and select "eclipseme.feature_1.2.3_site.zip"
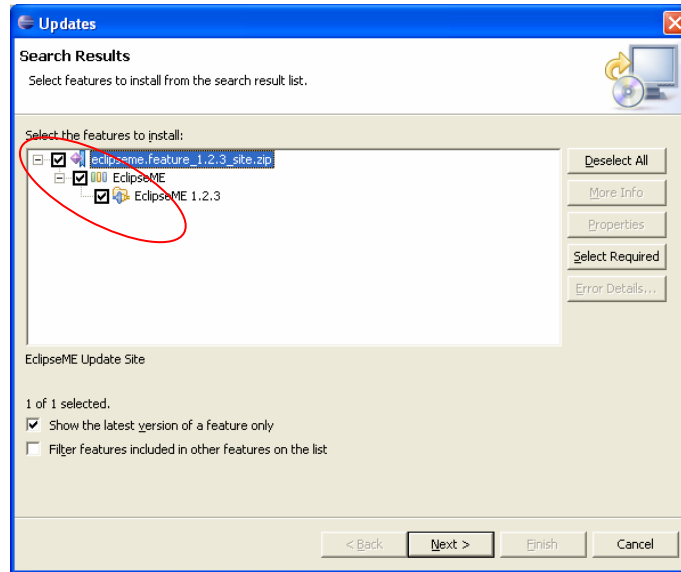


Figure 4: Installation of Eclipse ME-Plugin 1.2.3: dialog box *Search Results*

- Select "Install All" in dialog box *"verification"*
- Select "Yes" for restart Eclipse in dialog box *"Install/Update"*
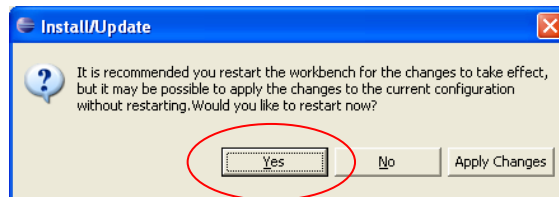


Figure 5: Installation of Eclipse ME-Plugin 1.2.3: dialog box *Restart Eclipse*

- Eclipse IDE is restarted. Please close Eclipse IDE before continuing with installation of SMTK!

***Step 4:*** Installation of the product's SMTK
- Start "setup.exe" from CD root path
- Choose Destination Folder of Eclipse "C:\Program Files\Eclipse" in dialog box "Select Eclipse Folder"
- Select SMTK (e.g. C:\Program Files\Siemens\SMTK) in dialog box "Destination Folder"
- Select COM port and set baud rate (e.g. 115200) for debugging connection and MES transfer for downloading the MIDlet to the module in dialog box "Select COM Port"
- After that MES will be installed without further user action. Please wait until setup is complete.
- Please click <Finish> in the summary dialog to confirm the completed installation.

A detailed description of the SMTK installation is given in the following chapters.

## 3.4      Siemens Mobility Toolkit Installation

The SMTK comes with an installation CD. The installation program automatically installs the necessary components and IDE integrations. Software can be uninstalled and updated with the install program. The next sections cover the installation and removal of the SMTK and the installation of the SDK and the supported IDEs.

### 3.4.1      Installing the Standard Development Toolkit

1.  The JDK version 1.4.2_09 is provided on the SMTK installation disk in the subdirectory "JDK 1.4". To begin the installation, start the "j2sdk-1_4_2_09-windows-i586-p.exe" and follow the instructions of the JDK setup procedure. If there is no JDK installed on the target machine the installation of the provided JDK will be offered automatically during the SMTK installation process.
2.  Once the toolkit has been installed, the environment variable "path" can be altered to comfortably use the JDK tools. This is not necessary for using the Siemens SMTK.
3.  Open the Control Panel.
    a)  Open *System.*
    b)  Click on *Advanced.*
    c)  Click on the *Environment Variables* button.
    d)  Choose *path* from the list of system variables.
    e)  Append the path for the bin directory of the newly installed SDK to the list of directories for the *path* variable.

### 3.4.2      Installing the SMTK Environment

Before you start the installation please make sure all applications, especially the IDEs are closed.

1.  Insert CD, start setup.exe. When the dialog box appears simply press the "Next" button to continue the procedure.
2.  You will be asked to read the license agreement. If you accept the agreement, press "Yes" to continue with the installation.
3.  A file including special information about the installation and use of the SMTK is shown. Press "Next" to continue.
4.  You will be asked to enter the path name where Eclipse 3.0.1, 3.0.2, 3.1.0, 3.1.1 or 3.1.2 is installed.
    Please type in the folder where Eclipse with the ME plugin is installed and press "Next".
    If you have not installed Eclipse or do not want to integrate the SMTK into Eclipse,
    please press "Next" without typing in a selected folder.
5.  The installation software checks for the Java SDK. If there is no SDK on the system the installation procedure now offers to install the provided JDK. If this step is refused, the setup process will not continue because a properly installed JDK is mandatory for using the SMTK environment.
6.  At this point, the installation software checks for a Java IDE to be integrated with the SMTK. A Java IDE is not necessary to use the SMTK. The IDE installation can be done at any time even if the SMTK is already installed. To integrate the SMTK into the Java IDE run the SMTK setup program in maintenance mode again. However, you can continue the setup procedure and install the IDE installation later or cancel the setup program at this stage and restart it after installing one of the supported Java IDEs. In case you wish to install a Java IDE please follow the instructions below and in Section

3.4.3. If no installed IDE is found the SMTK offers to install NetBeans IDE 5.0 and NetBeans Mobility package 5.0. Alternatively, you can install the SUN IDE by following the instructions in Section 3.4.3.

7. If the SDK and one or more Java IDEs are found, you will be asked to choose which IDE you want integrated into the development environment. Once an IDE has been found and selected, press "Next" to continue. Ensure that your Java IDE is closed.

8. Select the folder where the SMTK will be installed. A folder will be suggested to you but you may browse to select a different one.

9. Choose the path that SMTK will appear under in the Start Menu.

10. You will be asked to select the COM port for installing the standard modem and Dial-Up Network connection (DUN), needed for on-device debugging. If standard modem and Dial-Up Network connection (DUN) is already installed for another Siemens IMP-NG product, you will be asked to reconfigure this connection (e.g. selecting another COM port for all IMP-NG on-device debugging) by pressing "Yes" or to use the configured COM settings by pressing "No".
Note:
You can reconfigure the COM port for the Dial-Up Network connection any time after installation by starting SMTK setup in maintenance mode and selecting "Repair" mode.

If you like to use an USB modem connection, please install and configure that USB modem connection after SMTK installation manually. Therefore please plug in the USB cable in the module and PC, install the USB modem driver and select the USB modem connection using windows *Control Panel -> Network Connections* for the network connection "IP connection for remote debugging".

11. A brief summary of all entries made is shown. Press "Next" to continue.

12. A status message box will inform you that the Module Exchange Suite (MES) will be installed in the next step. A separate setup wizard for the Module Exchange Suite will be opened. Please follow the setup wizard's instructions.

13. After step 12, all necessary files will be copied from the CD into the target folder.

14. This is the final step. Again, a listing of all installed components appears. Please press "Finish" to end the installation.

Using "*Control Panel -> Add or Remove Programs*" for SMTK installation and selecting "Modify" or "Repair":

Please keep in mind, that windows system requires the same location of the installation CD, if you like to use "Modify" or "Repair" for SMTK installation. The installer is searching for *.msi file and an installer message box will pop up, telling the user to locate the installation CD in the corresponding drive and path. If the installation CD is not located on the displayed drive and path, please insert the CD or copy the CD contents into the required location (drive and path).

Note for installing the SMTK environment on Windows 2000 systems:
Please delete all modems with device name "Standard 19200 bps Modem" before installing the SMTK environment on a Windows 2000 system, because this modem device is used during automatic modem installation and on-device debugging.

### 3.4.3    Installing NetBeans IDE 5.0

1. NetBeans IDE 5.0 is provided on the SMTK installation disk in the subdirectory "NETBEANS_5_0". To begin installation, start first "netbeans-5_0-windows.exe" and follow the NetBeans IDE 5.0 setup procedure instructions.
2. Please install next the Mobility package of Sun™ NetBeans IDE 5.0. The Mobility package is required before integration of SMTK!

Note: The integration of the SMTK into Sun™ NetBeans IDE 5.0 is done for all users.

### 3.4.4    Installing Eclipse 3.0 or Eclipse 3.1

Eclipse 3.1.2 (CD directory "\Eclipse\") and Eclipse ME Plugin 1.2.3 (CD directory "\EclipseME_123\") are provided on the SMTK CD. The installation of Eclipse is shortly described in chapter 3.3.

Eclipse can be freely downloaded from http://www.eclipse.org. In order to use Eclipse with the ME the EclipseME plug-in is also needed. It can be downloaded from http://eclipseme.org/.

***Using Eclipse 3.0.1 or Eclipse 3.0.2:***
A customized version of this plug-in (CD directory "\EclipseME_WM\") for Eclipse 3.0.1 and 3.0.2 comes with SMTK. It is required to use this plug-in for Eclipse 3.0.1 and Eclipse 3.0.2, because EclipseME plugin 1.2.3 requires Eclipse 3.1.0 and later and not installable for the older Eclipse versions 3.0.1 and 3.0.2.

***Using Eclipse 3.1.0, Eclipse 3.1.1 or Eclipse 3.1.2:***
Please use EclipseME plug-in 1.2.1 or 1.2.3 for Eclipse version 3.1.0, Eclipse 3.1.1 or Eclipse 3.1.2.

Note: Only one selection for an Eclipse version at a time for SMTK integration is possible.

### 3.4.5    Installing Borland JBuilder X, 2005 and 2006 Enterprise/Developer

Borland JBuilder can be purchased from http://www.borland.com/jbuilder.

Note: The installation path name of JBuilder should not include space characters.

There are also 30 days trial versions available on the website. Installation instructions can be found on the web page.

### 3.4.6    Installing Module Exchange Suite (MES)

The Module Exchange Suite (MES) is installed during the SMTK installation. If you would like to install the Module Exchange Suite separately, repair or remove it, please use the Module Exchange Suite (MES) setup.exe, which is located on the SMTK installation disk in the subdirectory "MES".

## 3.5 SMTK Uninstall

The SMTK install package comes with an uninstall facility. The entire SMTK or parts of the package can be removed. To start the uninstall facility, open the *Control Panel*, select *Add/Remove Programs*, select the desired SMTK, e.g. *TC65 Software Development Kit* and follow the instructions. The standard modem and Dial-Up Network connection (DUN) are uninstalled automatically, if no SMTK IMP-NG product is installed in parallel. The user will be asked for uninstall of standard modem and Dial-Up Network (DUN) in the case, that another SMTK IMP-NG product is installed.

The Module Exchange Suite (MES) is not uninstalled automatically with the SMTK. If you would like to uninstall the Module Exchange Suite (MES) as well, please run the MES uninstall facility. To run the uninstall program, open the *Control Panel*, select *Add/Remove Programs*, select *Siemens Module Exchange Suite (MES)* and follow the instructions.

Please keep in mind, that standard modem, Dial-Up Network connection and Module Exchange Suite (MES) are required for a proper work of SMTK on-device debugging.

Note for customers migrating from TC45 to TC65:
If you uninstall the TC45 software after the installation of TC65 you need to run the TC65 SDK and MES setup in maintenance mode to restore several files required for TC65. To do so, open the Windows Control Panel, double-click Add or Remove Programs, select "TC65 Software Development Kit", press the Change/Remove button and select Repair. In the same manner, reinstall the "Siemens Module Exchange Suite (MES)".
As an alternative, to avoid this step, remove the TC45 package, before installing TC65 SDK and MES.

## 3.6 Upgrades

The SMTK can be modified, repaired or removed by running the setup program on the Installation CD.

# 4 Software Platform

In this chapter, we discuss the software architecture of the SMTK and the interfaces to it.

## 4.1 Software Architecture

The SMTK enables a customer to develop a Java application on a PC and have it be executable on the Java enabled module. The application is then loaded onto the module. The platform comprises:

- Java™ 2 Micro Edition (J2ME™), which forms the base of the architecture.
  The J2ME™ is provided by SUN Microsystems, http://java.sun.com/j2me/. It is specifically designed for embedded systems and has a small memory footprint. The ME uses:
  CLDC 1.1 HI, the connected limited device configuration hot spot implementation.
  IMP-NG, the information module profile $2^{nd}$ generation, this is for the most part identical to MIDP 2.0 but without the lcdui package.

- Additional Java virtual machine interfaces:
  AT Command API
  File I/O API

  The data flow through these interfaces is shown in Figure 7 and Figure 30.

- Memory space for Java programs:
  Flash File System: around 1700k
  RAM: around 400k

  Application code and data share the space in the flash file system and in RAM.

- Additional accessible periphery for Java applications
  - A maximum of ten digital I/O pins usable, for example, as:
    Output: status LEDs
  - Input: Emergency Button
  - One I2C/SPI Interface.
  - One Digital Analog Converter and two Analog Digital Converters.
  - Serial interface (RS-232 API): This standard serial interface could be used, for example, with an external GPS device or a current meter.
    For detailed information see chapter 4.2.

## 4.2 Interfaces

### 4.2.1 ASC0 - Serial Device

ASC0, an Asynchronous Serial Controller, is a 9-wire serial interface. It is described in the Hardware Interface Description [4]. Without a running Java application the module can be controlled by sending AT commands over ASC0. Furthermore, ASC0 is designed for transferring files from the development PC to the module. When a Java application is started, ASC0 can be used as an RS-232 port or/and System.out. Refer to the Java doc [5] for details.

### 4.2.2 General Purpose I/O

There are ten I/O pins that can be configured for general purpose I/O. One pin can be configured as a pulse counter. All lines can be accessed under Java by AT commands. See [3] and [4] for information about usage and startup behavior.

### 4.2.3 DAC/ADC

There are two analogue input lines and one analogue output line. They are accessed by AT commands. See [3] and [4] for details.

The TC65 Terminal does not feature DAC interface.

### 4.2.4 ASC1

ASC1 is the second serial interface on the module. This is a 4-pin interface (RX, TX, RTS, CTS). It can be used as a second AT interface when a Java application is not running or by a running Java application as RS-232 port or/and System.out.

The TC65 Terminal does not feature ASC1.

### 4.2.5 Digital Audio Interface (DAI)

The ME has a seven-line serial interface with one input data clock line and input/output data and frame lines to support the DAI. Refer to the AT Command Set [3] and Hardware Interface Description [4] documents for more information.

TC65 Terminal does not feature a DAI interface.

### 4.2.6 I2C/SPI

There is a 4 line serial interface which can be used as I2C or SPI interface. It is described in the Hardware Interface Description [4]. The at^sspi at command configures and drives this interface. For details see [4].

## 4.2.7 JVM Interfaces

| IMP-NG | File API | AT Command API |
|---|---|---|
| Connected Limited Device Configuration (CLDC) | | |
| J2ME | | |

Figure 6: Interface Configuration

J2ME, CLDC and MIDP were implemented by SUN. IMP-NG is a stripped down version of MIDP 2.0 prepared by Siemens and does not include the graphical interface LCDUI. Siemens developed the File I/O API and the AT command API. Documentation for J2ME and CLDC can be found at http://java.sun.com/j2me/. Documentation for the other APIs is found in …/ Java doc [5].

### 4.2.7.1 IP Networking

IMP-NG provides access to TCP/IP similarly to MIDP 2.0.

Because the used network connection, CSD or GPRS, is fully transparent to the Java interface, the CSD and GPRS parameters must be defined separately either by the AT command at^sjnet [3] or by parameters given to the connector open method, see Java doc [5].

### 4.2.7.2 Media

Media package is not supported. See Java doc [5].

### 4.2.7.3 Other Interfaces

Neither the PushRegistry interfaces and mechanisms nor any URL schemes for the PlatformRequest method are supported. See Java doc [5].

## 4.3    Data Flow of a Java Application Running on the Module
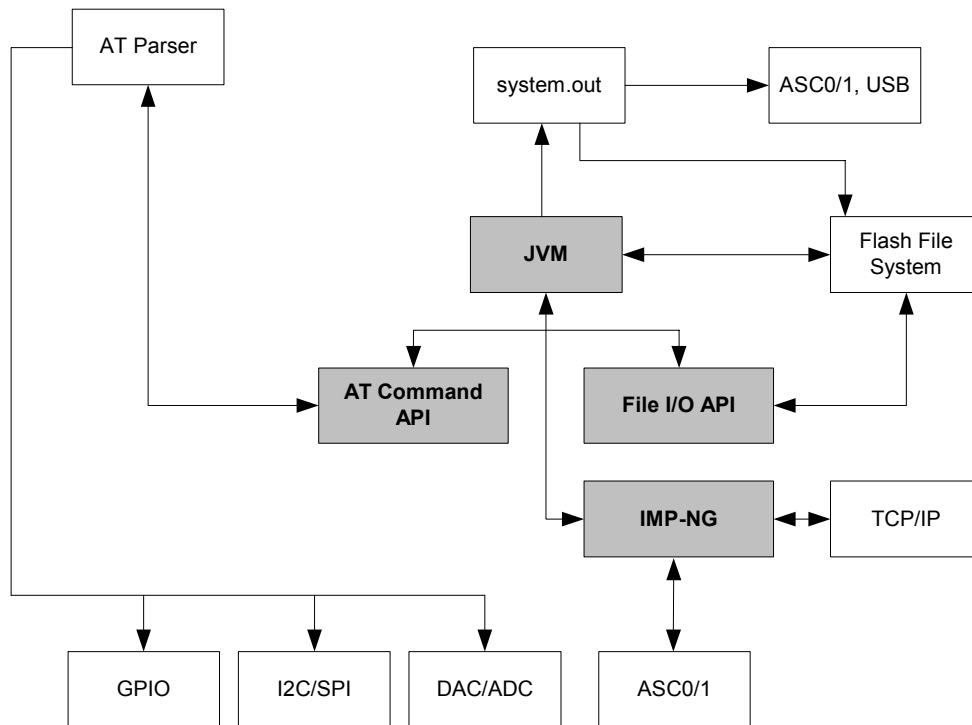


Figure 7: Data flow of a Java application running on the module.

The diagram shows the data flow of a Java application running on the module. The data flow of a Java application running in the debug environment can be found in Figure 30.

The compiled Java applications are stored as JAR files in the Flash File System of module. When the application is started, the JVM interprets the JAR file and calls the interfaces to the module environment.

The module environment consists of the:

- Flash File System:        available memory for Java applications
- TCP/IP:        module internal TCP/IP stack
- GPIO:        general purpose I/O
- ASC0:        Asynchronous serial interface 0
- ASC1:        Asynchronous serial interface 1
- I2C:        I2C bus interface
- SPI:        Serial Peripheral Interface
- DAC:        digital analog converter
- ADC:        analog digital converter
- AT parser:        accessible AT parser

The Java environment on the module consists of the:

- JVM:        Java Virtual Machine
- AT command API:        Java API to AT parser
- File API:        Java API to Flash File System
- IMP-NG:        Java API to TCP/IP and ASC0

## 4.4 Handling Interfaces and Data Service Resources

To develop Java applications the developer must know which resources, data services and hardware access are available.

- There are three AT parsers available
- There is hardware access over
  - two serial interfaces: ASC1 and ASC0 (both fully accessible).
  - general purpose I/O. To configure the hardware access, please refer to the AT Command Set [3] and the Hardware Interface Description [4].
  - I2C/SPI
  - All restrictions of combinations are described in section 4.4.1.

- A Java application has:
  - three instances of the AT command class, one with CSD and two without, each of which would, in turn, be attached to one of the three AT parsers.
  - two instances of access to a serial interface, ASC0 and ASC1, through the CommConnection API.
  - System.out over any serial interface or into the file system

## 4.4.1 Module States

The module can exist in the following six states in relation to a Java application, the serial interfaces, GPIO and I2C/SPI. See the AT Command Set [3] for information about any AT commands referenced. A state transition diagram is shown in Figure 12.

This section shows how Java applications must share AT parsers, GPIO pins and I2C/SPI resources. DAC, ADC and DAI are not discussed here.

| Default configuration of module |
| --- |

| Default configuration when Java application is started |
| --- |

| configured by AT Command |
| --- |
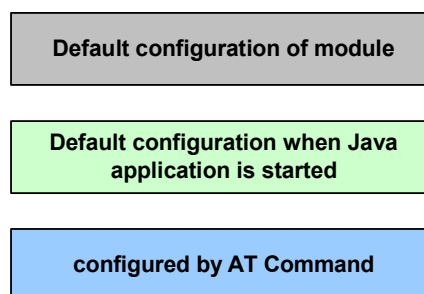
Color legend for the following figures

### 4.4.1.1    State 1: Default – No Java Running

This is the default state. The Java application is inactive and there is an AT interface with CSD on ASC0 as well as ASC1. The initial state of the pins is according to [4].

| AT parser with CSD | AT parser with CSD | 10 GPIO, I2C/SPI pins (unused) |
|---|---|---|
| ASC 0 | ASC1 or USB | |
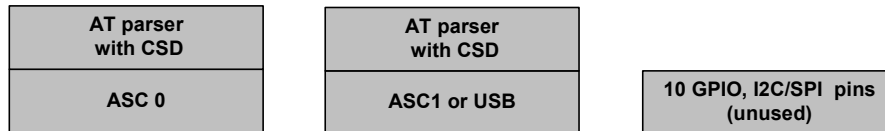
Figure 8: Module State 1

### 4.4.1.2    State 2: No Java Running, General Purpose I/O and I2C/SPI

The Java application is inactive. There is an AT parser with CSD on ASC0 as well as ASC1. Up to ten I/O pins are used as general purpose I/O plus a I2C/SPI interface. The pins are configured *by* at^scpin and at^sspi (refer *to* AT Command Set [3]).

| AT parser with CSD | AT parser with CSD | up to 10 GPIO pins | I2C/SPI |
|---|---|---|---|
| ASC0 | ASC 1 or USB | | |
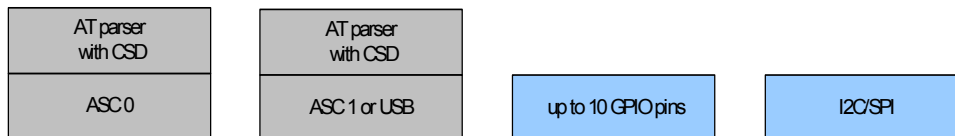
Figure 9: Module State 2

### 4.4.1.3    State 4: Default – Java Application Active

The Java application is active and ASC0 and ASC1 are used as System.out and/or CommConnection. Java instances of AT commands are connected to the available AT parsers. The Java application is activated with *at^sjra* (refer to AT Command Set [3]) or autostart.

| Java access to serial interface (CommConnection) | System.out | AT parser with CSD | AT parser without CSD | AT parser without CSD |
|---|---|---|---|---|
| ASC 0/1 | ASC0/1 | Java AT command API with CSD | Java AT command API without CSD | Java AT command API without CSD |
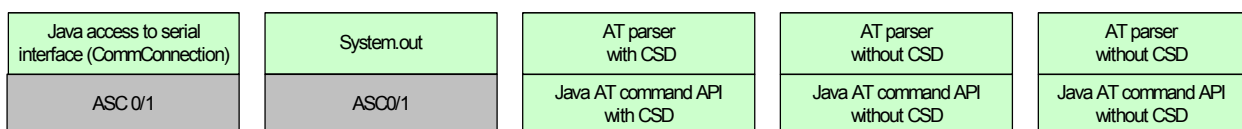
Figure 10: Module State 4

### 4.4.1.4    State 5: Java Application Active, General Purpose I/O and I2C/SPI

The Java application is active and ASC0 and ASC1 are used as System.out and/or CommConnection. The Java application is activated with *at^sjra.* The I/O pins are configured with at^scpin, the I2C/SPI interface with at^sspi. Refer to the AT Command Set [3] for AT command details.

| Java access to serial interface (CommConnection) | System out | AT parser 0 with CSD | AT parser without CSD | AT parser without CSD | up to 10 GPIO pins | I2C/SPI |
|---|---|---|---|---|---|---|
| ASC0/1 | ASC0/1 | Java AT command API with CSD | Java AT command API without CSD | Java AT command API without CSD | | |
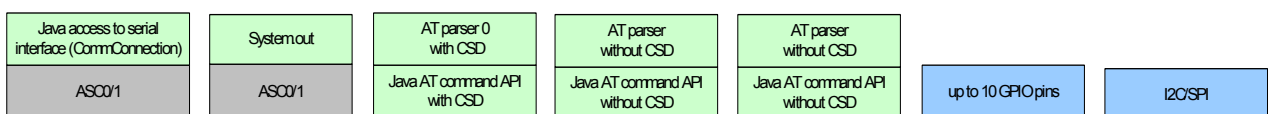
Figure 11: Module State 5

## 4.4.2    Module State Transitions
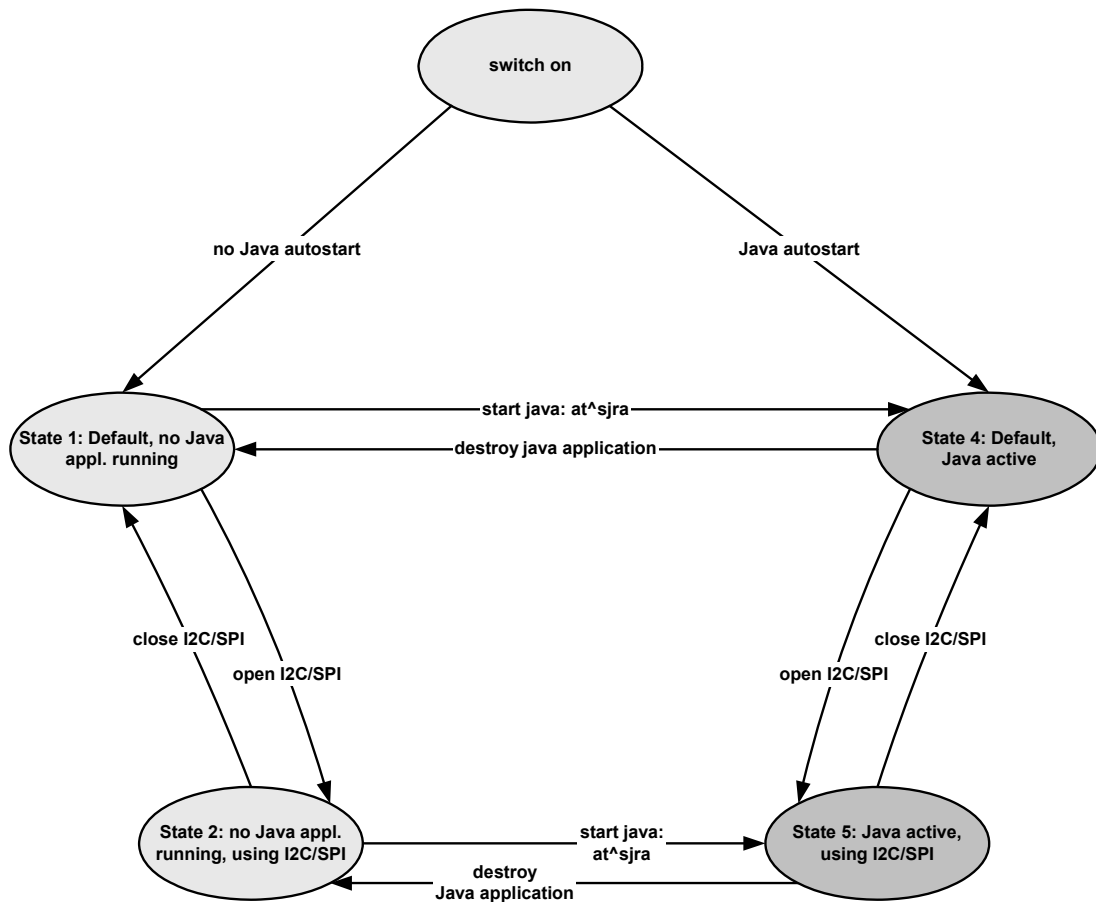


Figure 12: Module State Transition Diagram

Note: No AT parser is available over serial interface ASC0 or ASC1 while a Java application is running on the module.

# 5 Maintenance

The basic maintenance features of the ME are described below. Explicit details of these functions and modes can be found in the AT Command Set [3] and the Hardware Interface Description [4].

## 5.1 IP Service

Apart from the standard Java IP networking interfaces (UDPDatagramConnection, SocketConnection, ...) the ME also supports a set of Internet Services controlled by AT command. There are some correlations between the Java and the AT IP Services.

- The connection profile 0 is also used by Java: when Java starts up a networking connection it tries to set and activate connection profile 0 with the parameters configured by at^sjnet or in the connector.open method.
- Java tries to (re-)use an active Internet Service profile: if using connection profile 0 fails, because e.g. this (or another) connection profile is already used by the Internet Services, Java networking also uses this, already active, profile.
- Deactivation of the connection profile happens when all applications are "finished": Java has its networking idle time. For the Internet Services an inactivity timeout referred to as parameter <inactTO> is available (configurable by at^sics and at^scfg).

So that means that Java networking and AT Internet Services can be used in parallel but care has to be taken about configuring and activation of the connection profile. In the simplest case use connection profile 0 for the Internet Services and set the parameters to the same values as the Java networking parameters. This way it makes no difference whether the connection is activated by the Internet Services or Java.

There are some aspects which have to be kept in mind for all IP Services (Java and AT command):

- When an open TCP connection is cut (e.g. the other side dies/is switched off) it takes around 10 minutes during which retransmissions are send, until the situation is detected as an error (in Java an exception is thrown).
- The number of IP services used in parallel should be kept small. An active IP service uses up resources and may deteriorate the overall performance.
- For information about the bearer state, use the specific IP service command at^sici and, in addition, the general network commands at+cops and/or at+creg.

## 5.2 Remote SIM Access

While Remote SIM Access (RSA) is normally closely coupled with the GSM 07.10 multiplexer there are some things to keep in mind when using it with Java.

- Java must not be started when RSA and/or the GSM 07.10 multiplexer is activated.
- When activating the RSA mode (at^srsa) via a Java AT Command channel while Java is running the parameter <muxChan> of the at^srsa command is ignored and RSA is activated on the channel where the command was issued. The Input- and Outputstream can then be used to transfer RSA protocol data.

The TC65 does not feature RSA mode under Java.

## 5.3 Power Saving

The module supports several power saving modes which can be configured by the AT command *at+cfun* [3]. Power saving affects the Java application in two ways. First, it limits access to the serial interface (RS-232-API) and the GPIO pins. Second, power saving efficiency is directly influenced by the way a Java application is programmed.

Java hardware access limitations:
- In NON-CYCLIC SLEEP mode (cfun=0) the serial interface cannot be accessed. Toggling RTS does end NON-CYCLIC SLEEP mode. In CYCLIC SLEEP mode (CFUN=7 or 9) the serial interface can be used with hardware flow control (CTS/RTS).
- In all SLEEP modes the GPIO polling frequency is reduced so that only signal changes which are less than 0.2Hz can be detected properly. Furthermore, the signal must be constant for at least 2.7s to detect any changes. For further details see at^scpol in [3] or refer to [4].

Java power saving efficiency:
- As long as any Java thread is active, power consumption cannot be reduced, regardless whether any SLEEP mode has been activated or not. A Java application designed to be power efficient should not have any unnecessarily active threads (e.g. no busy loops). Threads waiting in a blocking method (e.g. read) do not hinder power saving.
- When using networking functionality close all connectors and hang-up the bearer manually (using "ath" when CSD is used or "at+cgatt=0" in case of GPRS) every time you intend to reduce power consumption. Disable the network idle timeout (=0).

When going to low power mode there always might be a transition time of around 10s till low power consumption state is reached.

## 5.4 Charging

Please refer to [3] and [4] for general information about charging. Charging can be monitored by the running Java application. The JVM is active in Charge mode and in Charge-Only mode if autostart is activated. Only a limited number of AT commands are available when the module is in Charge-Only mode. A Java application must be able to handle the Charge-Only mode and reset the module to reinstate the normal mode. See [4] for information about the Charge-Only mode.

The Charge-Only mode is indicated by URC "^SYSSTART CHARGE-ONLY MODE".

Note: When a Java application is started in Charge-Only mode only AT Command APIs without CSD are available. The mode-indicating URC is created after issuing the very first AT command on any opened channel. To read the URC it is necessary to register a listener (see [5]) on this AT command API instance before passing the first AT command.

## 5.5 Airplane Mode

The main characteristic of this mode is that the RF is switched off and therefore only a limited set of AT commands is available. The mode can be entered or left using the appropriate at^scfg command. This AT command can also be used to configure the airplane mode as the standard startup mode, see [4]. The JVM is started when autostart is enabled. A Java application must be able to handle this mode. The airplane mode is indicated by URC "SYSSTART AIRPLANE MODE". Since the radio is off all classes related to networking connections, e.g. SocketConnection, UDPDatagramConnection, SocketServerConnection, HTTPConnection, will throw an exception when accessed.

## 5.6 Alarm

The ALARM can be set with the *at+cala* AT command. Please refer to the AT Command Set [3] and Hardware Interface Description [4] for more information. One can set an alarm, switch off the module with *at^smso*, and have the module restart at the time set with *at+cala*. When the alarm triggers the module restarts in a limited functionality mode, the "airplane mode". Only a limited number of AT commands are available in this mode, although the JVM is started when autostart is enabled. A Java application must be able to handle this mode and reset the module to reinstate the normal mode. The mode of a module started by an alarm is indicated by the URC "^SYSSTART AIRPLANE MODE".

Note: For detailed information which functionality is available in this mode see [3] and [4]. The mode indicating URC is created after issuing the very first AT command on any opened channel.

## 5.7 Shutdown

If an unexpected shutdown occurs, data scheduled to be written will get lost due to a buffered write access to the flash file system. The best and safest approach to powering down the module is to issue the AT^SMSO command. This procedure lets the engine log off from the network and allows the software to enter a secure state and save all data. Further details can be found in [4].

### 5.7.1 Automatic Shutdown

The ME is switched off automatically in different situations:
• under- or overtemperature
• under- or overvoltage

Appropriate warning messages transmitted by the ME to the host application are implemented as URCs. To activate the URCs for temperature conditions use the *at^sctm* command. Undervoltage and overvoltage URCs are generated automatically when fault conditions occur.

For further detail refer to the commands at^sctm and at^sbc described in the AT Command Set [3]. In addition, a description of the shutdown procedures can be found in [4].

## 5.7.2    Manual Shutdown

The module can be switched off manually with the AT command, at^smso or when using the TC65 Terminal by pressing the ignition key for a period of time (see [4]). In these cases the midlets destroyApp method is called and the application has 5s time to clean up and call the notifydestroy method. After the 5s the VM is shut down.

## 5.7.3    Restart after Switch Off

When the module is switched off without setting an alarm time (see the AT Command Set [3]), e.g. after a power failure, external hardware must restart the module with the Ignition line (IGT). The Hardware Interface Description [4] explains how to handle a switched off situation.

## 5.8    Special AT Command Set for Java Applications

For the full AT command set refer to [3]. There are differences in the behaviour AT commands issued from a Java application in comparison to AT commands issued over a serial interface.

## 5.8.1    Switching from Data Mode to Command Mode

Cancellation of the data flow with "+++" is not available in Java applications, see [3] for details. To break the data flow use *breakConnection().* Refer to \wtk\doc\index.html [5].

## 5.8.2    Mode Indication after MIDlet Startup

After starting a module without autobauding on, the startup state is indicated over the serial interface.  Similarly, after MIDlet startup the module sends its startup state (^SYSSTART, ^SYSSTART ALARM MODE etc.) to the MIDlet. This is done via a URC to the AT Command API instance which executes the very first AT Command from within Java. To read this URC it is necessary to register a listener (see [5]) on this AT Command API instance before passing the first AT Command.

## 5.8.3    Long Responses

The AT Command API can handle responses of AT commands up to a length of 1024 bytes. Some AT commands have responses longer than 1024 bytes, for these responses the Java application will receive an Exception.

Existing workarounds:
- Instead of listing the whole phone book, read the entries one by one
- Instead of listing the entire short message memory, again list message after message
- Similarly, read the provider list piecewise
- Periods of monitoring commands have to be handled by Java, e.g. *at^moni, at^smong*. These AT commands have to be used without parameters, e.g. for *at^moni* the periods must be implemented in Java.

## 5.8.4 Configuration of Serial Interface

While a Java application is running on the module, only the AT Command API is able to handle AT commands. All AT commands referring to a serial interface are ignored. This includes the commands:

* AT+IPR (sets a fixed local bit rate)
* AT\Q1, AT\Q2 and AT\Q3 (sets type of flow control)

If Java is running, the firmware will ignore any settings from these commands. Responses to the read, write or test commands will be invalid or deliver "ERROR".

Note: When a Java application is running, all settings of the serial interface are done with the class CommConnection. This is fully independent of any AT commands relating to a serial interface.

## 5.8.5 Java Commands

There is a small set of special Java AT commands:

* at^sjra, start a Java application
* at^sjnet, configuration of Java networking connections
* at^sjotap, start and configuration of over the air provisioning
* at^sjsec, security configuration

Refer to the AT command Set [3].

## 5.9 System Out

Any output printed to the System.out stream by a Java application can be redirected to one of the serial interfaces, a file, a "NULL" device (i.e. the output will be discarded) or a UDP socket for using the debugger from an IDE. The configuration can be done at any time using the AT command at^scfg (see [3] for details) and is non-volatile.

## 5.9.1 Serial interfaces

System.out can be written to any of the serial interfaces ASC0, ASC1 or USB.
If System.out is redirected to one of the interfaces used for the Java CommConnection, the interface will be shared between System.out and the CommConnection. This will result in mixed output, if data is written to the CommConnections OutputStream and something is printed via System.out at the same time.
Using System.out and CommConnection on the same serial interface may be done if a device connected to the serial port is only transmitting data to the module. It is recommended to ensure already in the HW design that output from the module cannot be transferred to a connected device.

## 5.9.2    File

The `System.out` print can be redirected into log files within the module's flash file system. The output will be written alternatingly into two files which can be concatenated afterwards to have a single log file.

Writing the output to a file will slow down the virtual machine. To reduce the impact of logging the output may be written first to a buffer before it is written to the file (buffered mode). The buffered output is written either if the buffer is filled or after 200 ms. If the buffer is not used (secure mode) the output is written directly to the file.

## 5.9.3    UDP

Redirection to a UDP socket is used in conjunction with the debugger. UDP is used by default when using on-device-debugging (see chapter 11).  This can be changed by editing the emulators ini-file.

If the output is redirected to an UDP socket, any changes of the `System.out` configuration are ignored while the Virtual Machine is running. The UDP Socket settings will not be stored in the module.

## 5.10     Restrictions

## 5.10.1   Flash File System

The maximum length of a complete path name, including the path and the filename, is limited by the Flash file system on the module to 124 characters. It is recommended that names of classes and files be distinguished by more than case sensitivity.

## 5.10.2   Memory

The CLDC 1.1 HI features a just-in-time compiler. This means that parts of the Java byte code which are frequently executed are translated into machine code to improve efficiency. This feature uses up RAM space. There is always a trade off between code translation to speed up execution and RAM space available for the application.

## 5.11 Performance

The performance study was focused on comparable performance values under various circumstances.

### 5.11.1 Java

This section gives information about the Java command execution throughput ("jPS"= Java statements per second). The scope of this measurement is only the statement execution time, not the execution delay (Java command on AT interface ➔ Java instruction execution ➔ reaction on GPIO).
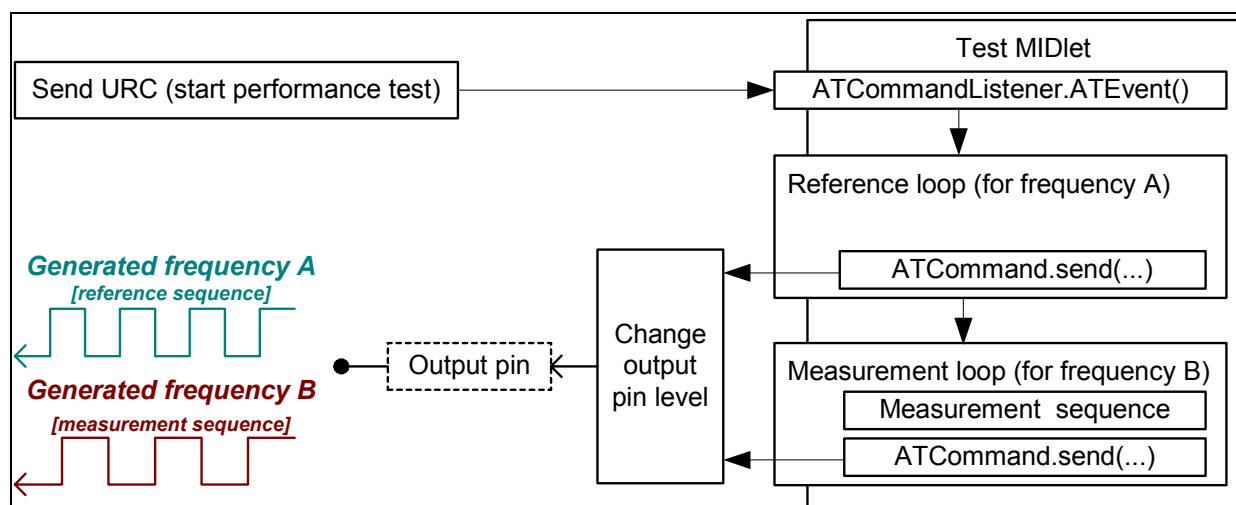


Figure 13: Test case for measuring Java command execution throughput

*The following Java instruction was used for calculation of the typical jPS:*

**value = ( 2 x number of calculation statements ) /**
**( ( 1 / frequencyB ) - ( 1 / frequencyA ) );**

Measurement and calculation were done using:
- **duration of each loop** = 600 s
- **number of calculation statements** = 50 "result=(CONSTANT_VALUE/variable_value);"-Instructions
  (executed twice per pin cycle)
- **frequencyA** as measured with a universal counter
- **frequencyB** as measured with a universal counter

The reference loop has the same structure as the measurement loop except that the measurement sequence is moved.

| State | jPS-Value (mean) |
|---|---|
| ME in IDLE mode / Not connected | ~49000 |
| CSD connection | ~46000 |

Since only a small amount of Java code is executed in this test, it is easily optimized by the CLDC's HI just-in-time compiler. More complex applications might not reach that execution speed.

## 5.11.2 Pin I/O

The pin I/O test was designed to find out how fast a Java MIDlet can process URCs caused by Pin I/O and react to these URCs. The URCs are generated by feeding an input pin with an external frequency. As soon as the Java MIDlet is informed about the URC, it tries to regenerate the feeding frequency by toggling another output pin.
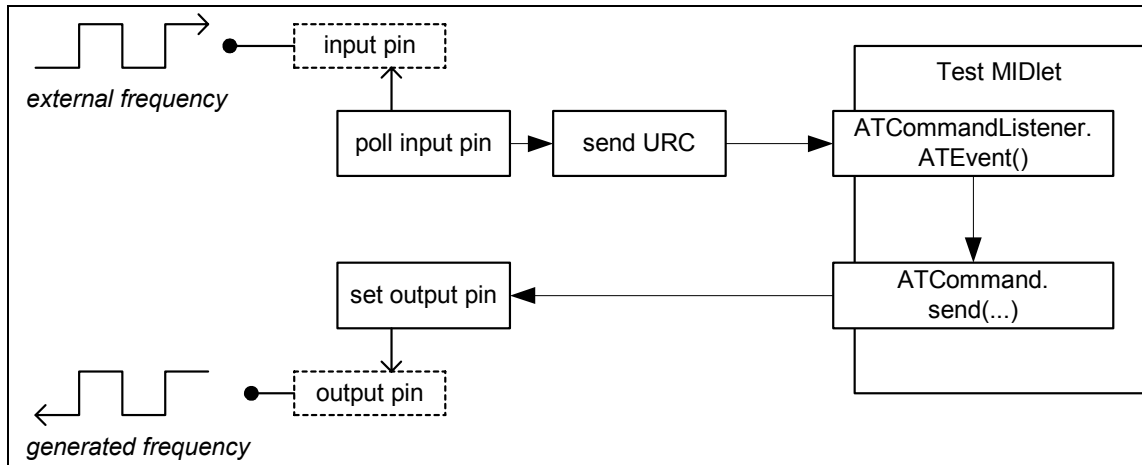
Figure 14: Test case for measuring Java MIDlet performance and handling pin-IO

The results of this test show that the delay from changing the state on the input pin to a state change on the output pin is at least around 50 ms, but that time strongly depends on the amount of garbage to collect and number of threads to be served by the virtual machine. Consequently, pin I/O is not suitable for generating or detecting frequencies.

## 5.11.3 Data Rates on RS-232 API

For details about the software platform and interfaces refer to Chapter 4, "Software Platform". This section summarises limitations and preconditions for performance when using the interface CommConnection from package com.siemens.mp.io (refer to [5]).

The data rate on RS232 depends on the size of the buffer used for reading from and writing to the serial interface. It is recommended that method *read (byte[ ] b)* be used for reading from the serial interface. The recommended buffer size is 2kbyte. To achieve error free data transmission the flow control on CommConnection must be switched on: <autorts> and <autocts>, the same for the connected device.

Different use cases are listed to give an idea of the attainable data rates. All applications for measurement use only one thread and no additional activities other than those described were carried out in parallel.

### 5.11.3.1   Plain Serial Interface

Scenario: A device is connected to ASC0 (refer to 4.2.4). The Java application must handle data input and output streams. A simple Java application (with only one thread) which loops incoming data directly to output, reaches data rates up to 180kbit/s. Test conditions: hardware flow control enabled (<autorts> and <autocts>), 8N1, and baud rate on ASC0 set to 230kbaud (-> theoretical maximum: 184kbit/s net data rate).
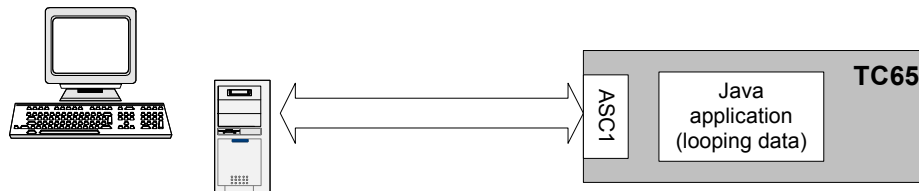
Figure 15: Scenario for testing data rates on ASC1

### 5.11.3.2   Voice Call in Parallel

Same scenario as in section 5.11.3.1, but with a voice call added. The application reflects incoming data directly to output and, additionally, handles an incoming voice call. The data rates are also up to 180kbit/s. Test conditions: same as in Sect.5.11.3.1.
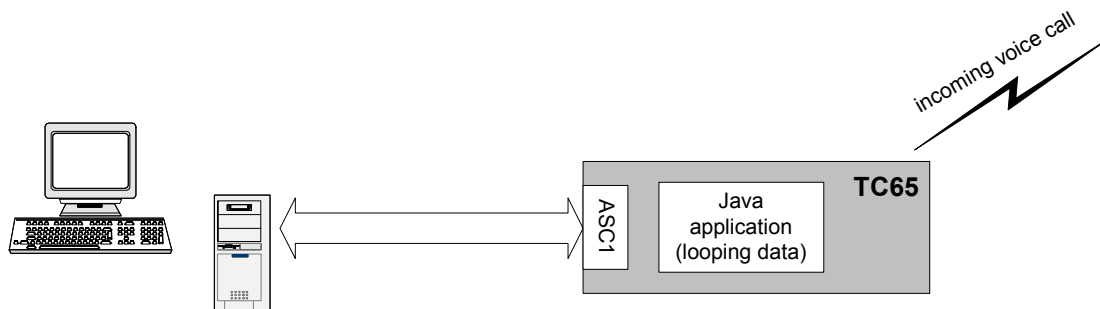
Figure 16: Scenario for testing data rates on ASC1 with a voice call in parallel

### 5.11.3.3 Scenarios with GPRS Connection

The biggest challenges to the module performance are setting up a GPRS connection, receiving data on javax.microedition.io interfaces and sending or receiving the data on the RS232 API with the help of a Java application.

#### 5.11.3.3.1 Upload

Since the ME supports GPRS class 12, up to four timeslots for upload data are available. The Java application receives data over RS232 API and sends them over GPRS to a server.

Table 1: Download data rate with different number of timeslots, CS2

| Upload data rate with x timeslots *Coding scheme 2* [kbit/s] | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 timeslot | theor. Value * | % from theor. Value | 2 time-slots | theor. Value * | % from theor. Value | 3 time-slots | theor. Value * | % from theor. Value | 4 time-slots | theor. Value * | % from theor. Value |
| 9 | 12 | 75% | 15 | 24 | 63% | 20 | 36 | 55% | 16 | 48 | 33% |

* net transmission rates for LLC layer

Table 2: Download data rate with different number of timeslots, CS4

| Upload data rate with x timeslots *Coding scheme 4* [kbit/s] | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 timeslot | theor. Value * | % from theor. value | 2 time-slots | theor. Value * | % from theor. value | 3 time-slots | theor. Value * | % from theor. value | 4 time-slots | theor. Value * | % from theor. value |
| 13 | 20 | 65% | 22 | 40 | 55% | 20 | 60 | 33% | 13 | 80 | 16% |

* net transmission rates for LLC layer
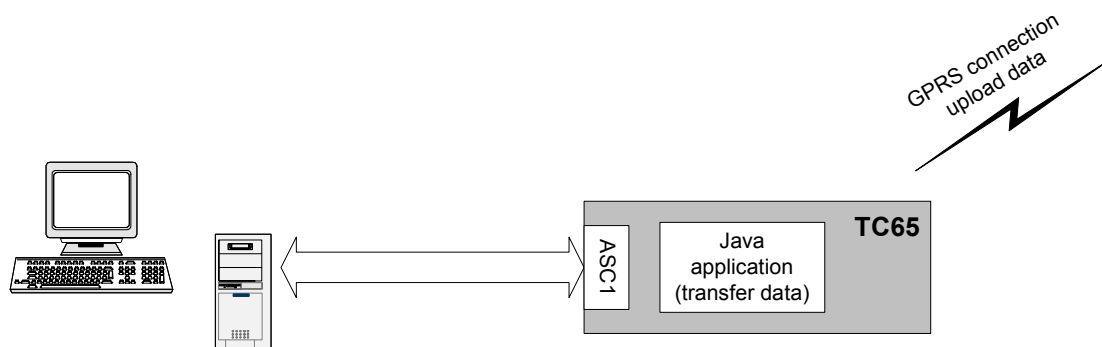


Figure 17: Scenario for testing data rates on ASC1 with GPRS data upload

### 5.11.3.3.2 Download

The data rate for downloading data over GPRS depends on the number of assigned timeslots and the coding schemes given by the net. Since ME supports GPRS class 12, the number of assigned timeslots can be up to 4. For the measurements, the Java application receives data from the server over GPRS and sends them over RS232 to an external device.
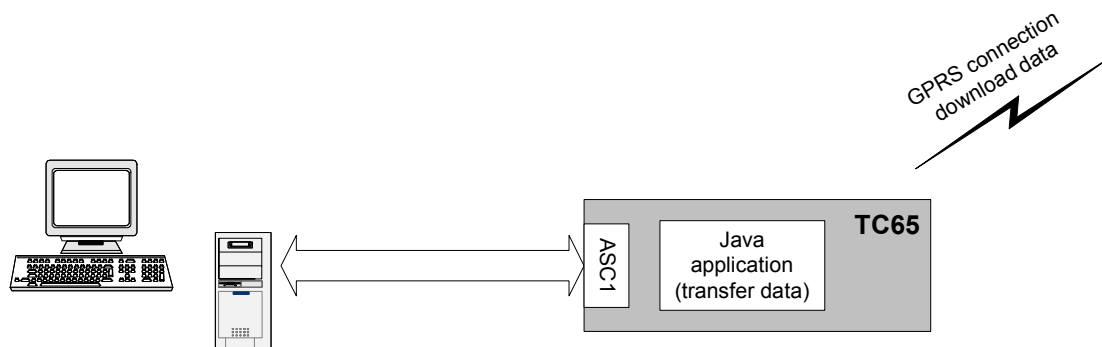


Figure 18: Scenario for testing data rates on ASC1 with GPRS data download

The tables below show the download data rates that can be achieved if hardware flow control is enabled on the CommConnection interface and the port speed is set to 230400.

Table 3: Download data rate with different number of timeslots, CS2

| Download data rate with x timeslots Coding scheme 2 [kbit/s] | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 timeslot | theor. Value * | % from theor. Value | 2 time-slots | theor. Value * | % from theor. Value | 3 time-slots | theor. Value * | % from theor. Value | 4 time-slots | theor. Value * | % from theor. Value |
| 11 | 12 | 91% | 21 | 24 | 87% | 29 | 36 | 81% | 35 | 48 | 73% |

* net transmission rates for LLC layer

Table 4: Download data rate with different number of timeslots, CS4

| Download data rate with x timeslots Coding scheme 4 [kbit/s] | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 timeslot | theor. Value * | % from theor. value | 2 time-slots | theor. Value * | % from theor. value | 3 time-slots | theor. Value * | % from theor. value | 4 time-slots | theor. Value * | % from theor. value |
| 17 | 20 | 85% | 31 | 40 | 78% | 35 | 60 | 58% | 38 | 80 | 48% |

* net transmission rates for LLC layer

# 6 MIDlets

The J2ME™ Mobile Information Device Profile (MIDP) provides a targeted Java API for writing wireless applications. The MIDP runs on top of the Connected Limited Device Configuration (CLDC), which in turn, runs on top of the J2ME™. MIDP applications are referred to as MIDlets. MIDlets are controlled by the mobile device implementation that supports the CLDC and MIDP. Since IMP-NG is a subset of MIDP 2.0, IMP includes MIDlets. The MIDlet code structure is very similar to applet code. There is no main method and MIDlets always extend from the MIDlet class. The MIDlet class in the MIDlet package provides methods to manage a MIDlet's life cycle.

## 6.1 MIDlet Documentation

MIDP and MIDlet documentation can be found at http://wireless.java.sun.com/midp/ and in the html document directory of the wtk,
*…\Siemens\SMTK\<productname>\wtk\doc\index.html*

## 6.2 MIDlet Life Cycle

The MIDlet life cycle defines the protocol between a MIDlet and its environment through a simple well-defined state machine, a concise definition of the MIDlet's states and APIs to signal changes between the states. A MIDlet has three valid states:

- *Paused* – The MIDlet is initialised and is quiescent. It should not be holding or using any shared resources.
- *Active* – The MIDlet is functioning normally.
- *Destroyed* – The MIDlet has released all of its resources and terminated. This state is only entered once.

State changes are controlled by the MIDlet interface, which supports the following methods:

- *pauseApp()* – the MIDlet should release any temporary resources and become passive.
- *startApp()* – the MIDlet starts its execution, needed resources can be acquired here or in the MIDlet constructor.
  Note: Take care that the startApp() method is always properly terminated before calling the destroyApp() method. For example, avoid that threads launched by startApp() enter a closed loop, and be sure that all code was entirely executed. This is especially important for OTAP, which needs to call destroyApp().
- *destroyApp()* – the MIDlet should save any state and release all resources
  Note: To destroy only the Java application without switching off the module, the destroyApp() method can be called explicitly. To destroy the Java application and switch off the module at the same time, it is sufficient to send the AT^SMSO command from somewhere in your code, because this procedure implies calling the destroyApp() method. Likewise, resetting the module with AT+CFUN=x,1 also implies calling the destroyApp() method. Note that AT+CFUN=x,1 will restart the module – to restart Java afterwards either use the autostart mode configured with AT^SCFG or restart Java with AT^SJRA.
  From this you can see that the commands AT^SMSO and AT+CFUN=x,1 should never be sent within the destroyApp() method.
  It is good practice to always call the notifyDestroyed() method at the end of your destroyApp method. And use the destroyApp method as single exit point of your midlet.

- *notifyDestroyed()* – the MIDlet notifies the application management software that it has cleaned up and is done.
  Note: the only way to terminate a MIDlet is to call notifyDestroyed(), but destroyApp() is not automatically called by notifyDestroyed(). You must not terminate your midlet (i.e. having no threads left) and not calling notifyDestroyed() before.
- *notifyPaused()* – the MIDlet notifies the application management software that it has paused
- *resumeRequest()* – the MIDlet asks application management software to be started again.

Table 5: A typical sequence of MIDlet execution

| Application Management Software | MIDlet |
|---|---|
| The application management software creates a new instance of a MIDlet. | The default (no argument) constructor for the MIDlet is called; it is in the Paused state. |
| The application management software has decided that it is an appropriate time for the MIDlet to run, so it calls the MIDlet.startApp method for it to enter the Active state. | The MIDlet acquires any resources it needs and begins to perform its service. |
| The application management software no longer needs the application be active, so it signals it to stop performing its service by calling the *MIDlet.pauseApp* method. | The MIDlet stops performing its service and might choose to release some resources it currently holds. |
| The application management software has determined that the MIDlet is no longer needed, or perhaps needs to make room for a higher priority application in memory, so it signals the MIDlet that it is a candidate to be destroyed by calling the *MIDlet.destroyApp* method. | If it has been designed to do so, the MIDlet saves state or user preferences and performs clean up. |

## 6.3 Hello World MIDlet

Here is a sample HelloWorld program.

```java
/**
 * HelloWorld.java
 */

package example.helloworld;
import javax.microedition.midlet.*;
import java.io.*;

public class HelloWorld extends MIDlet {

  /**
   * HelloWorld - default constructor
   */
  public HelloWorld() {
    System.out.println("HelloWorld: Constructor");
  }

  /**
   * startApp()
   */
  public void startApp() throws MIDletStateChangeException {
    System.out.println("HelloWorld: startApp");
    System.out.println("\nHello World!\n");
    destroyApp();
  }

  /**
   * pauseApp()
   */
  public void pauseApp() {
    System.out.println("HelloWorld: pauseApp()");
  }

  /**
   * destroyApp()
   */
  public void destroyApp(boolean cond) {
    System.out.println("HelloWorld: destroyApp(" + cond + ")");
    notifyDestroyed();
  }
}
```

# 7      File Transfer to Module

## 7.1     Module Exchange Suite

The Module Exchange Suite allows you to view the Flash file system on the module as a directory from Windows Explorer. Make sure that the module is turned on and that one of the module's serial interfaces (ASC0, ASC1 or USB) is connected to the COM port that the Module Exchange Suite is configured to. The configured COM port can be checked or changed under Properties of the Module directory. Please note that the Module Exchange Suite can be used only if the module is in normal mode. While running the module with the Module Exchange Suite, subdirectories and files can be added to the flash file system of module. Keep in mind that a maximum of 200 flash objects (files and subdirectories) per directory in the flash file system of the module is recommended.

## 7.1.1     Windows Based

The directory is called "Module" and can be found at the top level of workspace "MyComputer". To transfer a file to the module, simply copy the file from the source directory to the target directory in the "Module -> Module Disk (A:)".

## 7.1.2     Command Line Based

A suite of command line tools is available for accessing the module's Flash file system. They are installed in the Windows System directory so that the tools are available from any directory. The module's file system is accessed with *mod:*. The tools included in this suite are MESdel, MEScopy, MESxcopy, MESdir, MESmkdir, MESrmdir, MESport, MESclose and MESformat. Entering one of these commands without arguments will describe the command's usage. The tools mimic the standard directory and file commands. A path inside the module's file system is identified by using "mod:" followed by the module disk which is always "A:" (e.g. "MESdir mod:a:" lists the contents of the module's root directory).

## 7.2 Over the Air Provisioning

See Chapter 8 for OTA provisioning.

## 7.3 Security Issues

The developer should be aware of the following security issues. Security aspects in general are discussed in chapter 12.

### 7.3.1 Module Exchange Suite

The serial interface should be mechanically protected.

The copy protection rules for Java applications prevent opening, reading, copying, moving or renaming of JAR files. It is not recommended that the name of a Java application (for example <name>.jar) be used for a directory, since the copy protection will refuse access to open, copy or rename such directories.

### 7.3.2 OTAP

- A password should be used to update with OTA (SMS Authentication)
- Parameters should be set to fixed values *(at^sjotap)* whenever possible so that they cannot be changed over the air.
- The http server should be secure. (e.g. Access control via basic authentication)

# 8 Over The Air Provisioning (OTAP)

## 8.1 Introduction to OTAP

OTA (Over The Air) Provisioning of Java Applications is a common practice in the Java world. OTAP describes mechanisms to install, update and delete Java applications over the air. The ME implements the Over The Air Application Provisioning as specified in the IMP-NG standard (JSR228).

The OTAP mechanism described in this document does not require any physical user interaction with the device; it can be fully controlled over the air interface. Therefore it is suitable for Java devices that are designed not to require any manual interaction such as vending machines or electricity meters.

## 8.2 OTAP Overview

To use OTAP, the developer needs, apart from the device fitted with the Java enabled module, an http server, which is accessible over a TCP/IP connection either over GPRS or CSD, and an SMS sender, which can send Class1, PID $7d short messages. This is the PID reserved for a module's data download.
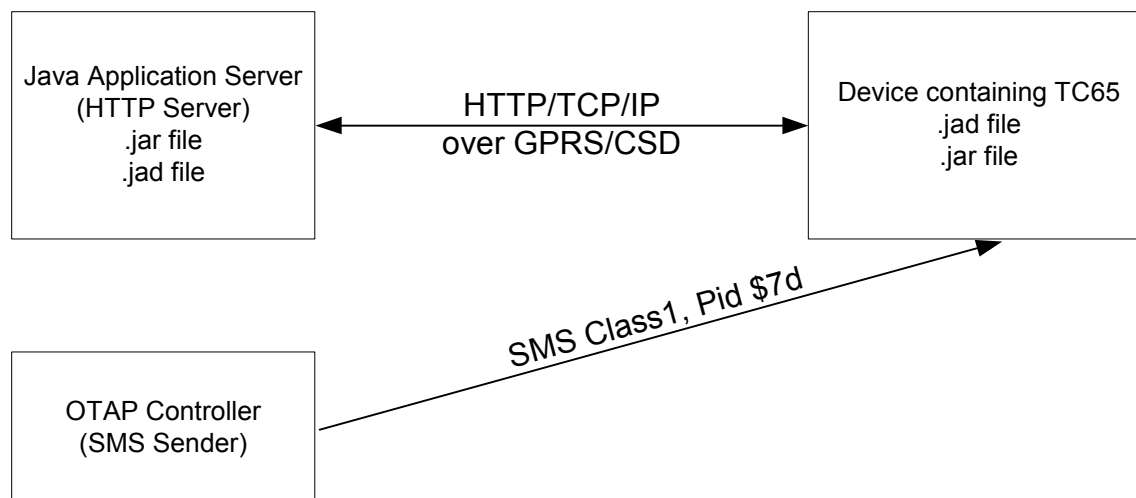


Figure 19: OTAP Overview

The Java Application Server (http Server) contains the .jar and the .jad file to be loaded on the device. Access to these files can be protected by http basic authentication.

The OTAP Controller (SMS Sender) controls the OTAP operations. It sends SMs, with or without additional parameters, to the devices that are to be operated. These devices then try to contact the http server and download new application data from it. The OTAP Controller will not get any response about the result of the operation. Optionally the server might get a result response over http.

There are two types of OTAP operations:
- Install/Update: A new JAR and JAD file are downloaded and installed.
- Delete: A complete application (.jar, .jad, all application data and its directory) is deleted.

## 8.3     OTAP Parameters

There is a set of parameters that control the OTAP procedures. These parameters can either be set by AT command (at^sjotap, refer to AT Command Set [7]) during the production of the device, or by SM (see Section 8.4) during operation of the device in the field. None of the parameters, which are set by AT command, can be overwritten by SM.

* JAD File URL: the location of the JAD file is used to install or update procedures. The JAD file needs to be located on the net (e.g. http://someserver.net/somefile.jad or http://192.168.1.2/somefile.jad ).
* Application Directory: this is the directory where a new application (JAD and JAR file) is installed. The delete operation deletes this directory completely. When entering the application directory with at^sjotap or a short message ensure that the path name is not terminated with a slash. For example, type "a:" or "a:/otap" rather than "a:/" or "a:/otap/". See examples provided in Chapter 8.4.
* http User: a username used for authentication with the http server.
* http Password: a password used for authentication with the http server.
* Bearer: the network bearer used to open the HTTP/TCP/IP connection, either GPRS or CSD.
* APN or Number: depending on the selected network bearer this is either an access point name for GPRS or a telephone number for CSD.
* Net User: a username used for authentication with the network.
* Net Password: a password used for authentication with the network.
* DNS: a Domain Name Server's IP address used to query hostnames.
* NotifyURL: the URL to which results are posted. This parameter is only used when the MIDlet-Install-Notify attribute or MIDlet-Delete-Notify attribute is not present in the descriptor.

There is one additional parameter that can only be set by AT command:
* SM Password: it is used to authenticate incoming OTAP SMs. Setting this password gives an extra level of security.
  Note: If a password set by AT command, all SMs must include this password

Table 6: Parameters and keywords

| Parameters | Max. Length AT | Keyword SM | Install/update | delete |
|---|---|---|---|---|
| JAD File URL | *100* | JADURL | mandatory | unused |
| Application Directory | *50* | APPDIR | mandatory | mandatory |
| HTTP User | *32* | HTTPUSER | optional | unused |
| HTTP Password | *32* | HTTPPWD | optional | unused |
| Bearer | *--* | BEARER | mandatory | optional/P |
| APN or Number | *65* | APNORNUM | mandatory for CSD | optional/P |
| Net User | *32* | NETUSER | optional | optional/P |
| Net Password | *32* | NETPWD | optional | optional/P |
| DNS | *--* | DNS | optional | optional/P |
| Notify URL | *100* | NOTIFYURL | optional | optional/P |
| SM Password | *32* | PWD | optional | optional |

The length of the string parameters in the AT command is limited (see Table 6), the length in the SM is only limited by the maximum SM length.

The minimum set of required parameters depends on the intended operation (see Table 6). "optional/P" indicates that this parameter is only necessary when a POST result is desired.

## 8.4     Short Message Format

An OTAP control SM must use a Submit PDU with Class1, PID $7d and 8 bit encoding. As a fallback for unusual network infrastructures the SM can also be of Class0 and/or PID $00. The content of the SM consists of a set of keywords and parameter values all encoded in ASCII format. These parameters can be distributed over several SMs. There is one single keyword to start the OTAP procedure. For parameters that are repeated in several SMs only the last value sent is valid. For example, an SM could look like this:

**Install operation:**

First SM:       OTAP_IMPNG
                PWD:secret
                JADURL:http://www.greatcompany.com/coolapps/mega.jad
                APPDIR:a:/work/appdir
                HTTPUSER:user
                HTTPPWD:anothersecret


Second SM:   OTAP_IMPNG
                PWD:secret
                BEARER:gprs
                APNORNUM:access.to-thenet.net
                NETUSER:nobody
                NETPWD:nothing
                DNS:192.168.1.2
                START:install


**Delete operation:**

OTAP_IMPNG
PWD:secret
APPDIR:a:/work/appdir
START:delete


The first line is required: it is used to identify an OTAP SM. All other lines are optional and their order is insignificant, each line is terminated with an LF: '\n' including the last one. The keywords, in capital letters, are case sensitive. A colon separates the keywords from their values.

The values of APPDIR, BEARER and START are used internally and must be lower case. The password (PWD) is case sensitive. The case sensitivity of the other parameter values depends on the server application or the network. It is likely that not all parameters can be sent in one SM. They can be distributed over several SMs. Every SM needs to contain the identifying first line (OTAP_IMPNG) and the PWD parameter if a mandatory password has been enabled. OTAP is started when the keyword START, possibly with a parameter, is contained in the SM and the parameter set is valid for the requested operation. It always ends with a reboot, either when the operation is completed, an error occurred, or the safety timer expired. This also means that all parameters previously set by SM are gone.

Apart from the first and the last line in this example, these are the parameters described in the previous section. Possible parameters for the START keyword are: "install", "delete" or nothing. In the last case, an install operation is done by default.

The network does not guarantee the order of SMs. So when using multiple SMs to start an OTAP operation their order on the receiving side might be different from the order in which they were sent. This could lead to trouble because the OTAP operation might start before all parameters are received. If you discover such problems, try waiting a few seconds between each SM.

# 8.5 Java File Format

In general, all Java files have to comply with the IMP-NG and ME specifications. There are certain components of the JAD file that the developer must pay attention to when using OTAP:

- MIDlet-Jar-URL: make sure that this parameter points to a location on the network where your latest JAR files will be located, e.g. http://192.168.1.3/datafiles/mytest.jar, not in the filesystem like file://a:/java/mytest/mytest.jar. Otherwise this JAD file is useless for OTAP.
- MIDlet-Install-Notify: this is an optional entry specifying a URL to which the result of an update/install operation is posted. That is the only way to get any feedback about the outcome of an install/update operation. The format of the posted URL complies with the IMP-NG OTA Provisioning specification. In contrast to the jar and jad file this URL must not be protected by basic authentication.
- MIDlet-Delete-Notify: this is an optional entry specifying a URL to which the result of a delete operation is posted. That is the only way to get any feedback about the outcome of a delete operation. The format of the posted URL complies with the IMP-NG OTA Provisioning specification. In contrast to the jar and jad file this URL must not be protected by basic authentication.
- MIDlet-Name, MIDlet-Version, MIDlet-Vendor: are mandatory entries in the JAD and Manifest file. Both files must contain equal values, otherwise result 905 (see 8.7) is returned.
- MIDlet-Jar-Size must contain the correct size of the jar file, otherwise result 904 (see 8.7) is returned.

Example:
MIDlet-Name: MyTest
MIDlet-Version: 1.0.1
MIDlet-Vendor: TLR Inc.
MIDlet-Jar-URL: http://192.168.1.3/datafiles/MyTest.jar
MIDlet-Description: My very important test
MIDlet-1: MyTest, , example.mytest.MyTest
MIDlet-Jar-Size: 1442
MicroEdition-Profile: IMP-NG
MicroEdition-Configuration: CLDC-1.1

A suitable Manifest file for the JAD file above might look like:
Manifest-Version: 1.0
MIDlet-Name: MyTest
MIDlet-Version: 1.0.1
MIDlet-Vendor: TLR Inc.
MIDlet-1: MyTest, , example.mytest.MyTest
MicroEdition-Profile: IMP-NG
MicroEdition-Configuration: CLDC-1.1

## 8.6     Procedures

## 8.6.1     Install/Update



Figure 20: OTAP: Install/Update Information Flow (messages in brackets are optional)

When an SM with keyword START:install is received and there is a valid parameter set for the operation, the module always reboots either when the operation completed, an error occurred or the safety timer expired. If there is any error during an update operation the old application is kept untouched, with one exception. If there is not enough space in the file system to keep the old and the new application at the same time, the old application is deleted before the download of the new one, therefore it is lost when an error occurs.
If install/update was successful, autostart is set to the new application.

## 8.6.2 Delete

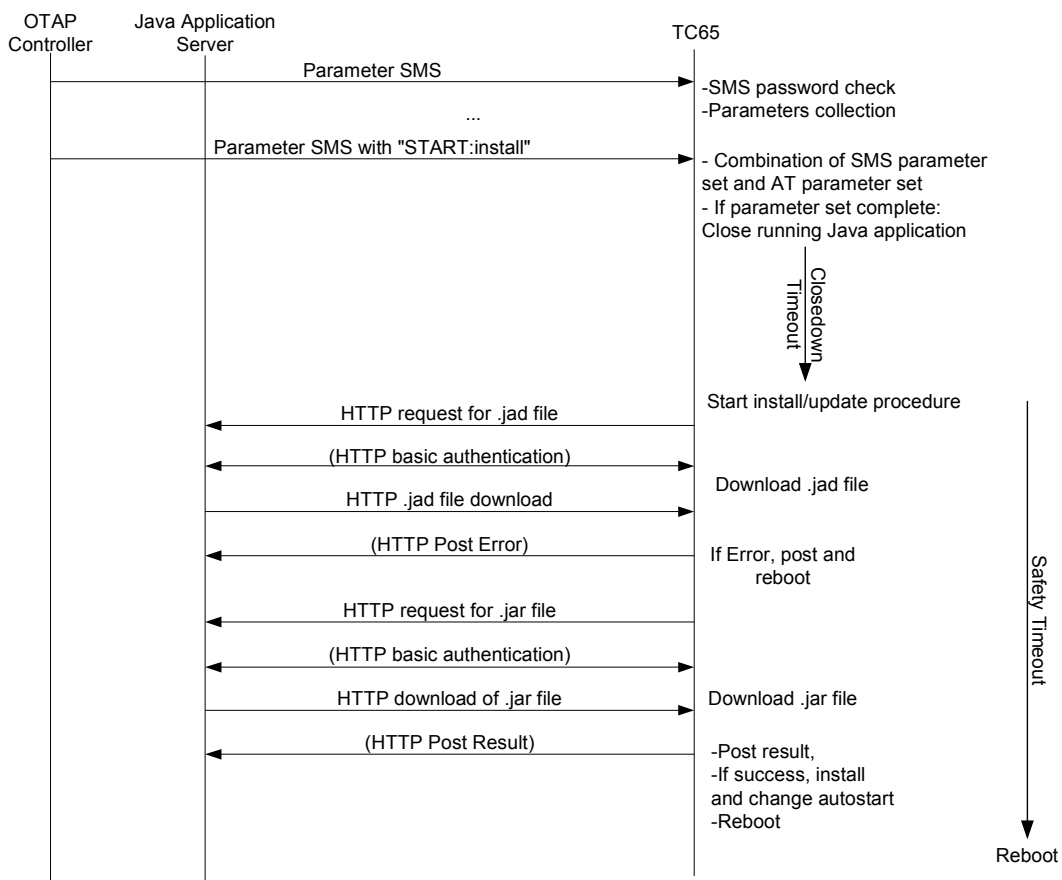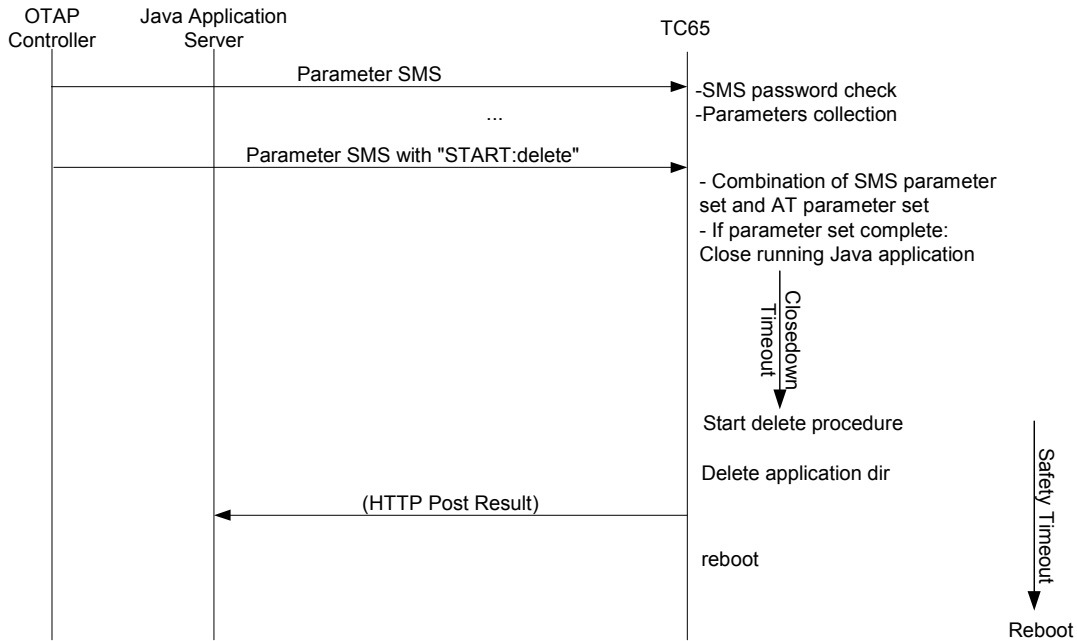Figure 21: OTAP: Delete Information Flow (messages in brackets are optional)

When an SM with keyword START:delete is received and there is a valid parameter set for this operation, the module reboots either when the operation completed, an error occurred or the safety timer expired. If there is any error the application is kept untouched. Autostart is not changed.

## 8.7 Time Out Values and Result Codes

Timeouts:
- Closedown Timeout: 10 seconds
- Safety Timeout: 10 minutes

Result Codes: Supported status codes in body of the http POST request:
- 900         Success
- 901         Insufficient memory in filesystem
- 902         -not supported-
- 903         -not supported-
- 904         JAR size mismatch, given size in JAD file does not match real size of jar file
- 905         Attribute mismatch, one of the mandatory attributes MIDlet-name, MIDlet-version, MIDlet-Vendor in the JAD file does not match those given in the JAR manifest
- 906         invalid descriptor, something is wrong with the format of the .jad file
- 907 invalid JAR, the JAR file was not available under MIDlet-Jar-URL, files could not be extracted from JAR archive, or something else is wrong with the format of the file.
- 908 incompatible configuration or profile
- 909 application authentication failure, signature did not match certificate
- 910 application authorization failure, tried to replace signed with unsigned version
- 911 -not supported-
- 912 Delete Notification

All HTTP packets (GET, POST) sent by the module contain the IMEI number in the User-Agent field, e.g.

```
User-Agent: <productname>/000012345678903 Profile/IMP-NG
Configuration/CLDC-1.1
```

This eases device identification at the HTTP server.

## 8.8 Tips and Tricks for OTAP

- For security reasons it is recommended that an SMS password be used. Otherwise the "delete" operation can remove entire directories without any authentication.
- For extra security, set up a private CSD/PPP Server and set its phone number as a fixed parameter. This way, applications can only be downloaded from one specific server.
- As a side effect, OTAP can be used to simply reboot the module. Simply start an OTAP procedure with a parameter set which will not really do anything, such as a delete operation on a nonexistent directory.
- If you do not want to start OTAP by SMS let your Java application do it by issuing the at^sjotap command. This triggers an install/update operation as described in chapter 8.6.1 but without the SMS part.
  Note: If a malfunctioning Java application is loaded the SM method will still be needed for another update.
- The OTAP procedure cannot be tested in the debug environment
- Be aware that the module needs to be logged into the network to do OTAP. That means that either the Java application must enter the PIN, the PIN needs to be disabled or Autopin (see AT Command Set [3]) needs be used.
- The OTAP procedure might fail due to call collision, e.g. an incoming call when OTAP tries to start a CSD connection.

## 8.9 OTAP Tracer

For easy debugging of the OTAP scenario, the OTAP procedure can be traced over the serial interface. The trace output shows details of the OTAP procedure and the used parameters. To enable the OTAP trace output use the at command at^scfg, e.g.

AT^SCFG=Trace/Syslog/OTAP,1

The serial interface on which you issue this command is then exclusively used for the OTAP tracer. All other functionality which is normally present (AT commands or CommConnection and System.out in Java) is not available when the tracer is on.

This feature is intended to be used during development phase and not in deployed devices.

## 8.10 Security

Java Security as described in chapter 12 also has consequences for OTAP. If the module is in secured mode the MIDlet signature is also relevant to the OTAP procedure. This means:

- If the application is an unsigned version of an installed signed version of the same application then status code 910 is returned.
- If the applications signature does not match the module's certificate then status code 909 is returned.

## 8.11    How To

This chapter is a step-by-step guide for using OTAP.

1.  Do you need OTAP? Is there any chance that it might be necessary to update the Java application, install a new one or delete it? It could be that device is in the field and you cannot or do not want to update it over the serial line. If the answer is "yes" then read through the following steps, if the answer is "no" then consider simply setting the OTAP SMS password to protect your system. Then you are finished with OTAP.
2.  Take a look at the parameters (chapter 8.3), which control OTAP. You need to decide which of them you want to allow to be changed over the air (by SMS) and which you do not. This is mainly a question of security and what can fit into a short message. Then set the "unchangeable" parameters with the AT command (*at^sjotap*).
3.  Prepare the http server. The server must be accessible from your device over TCP/IP. That means there is a route from your device over the air interface to the http server and back. When in doubt, write a small Java application using the httpConnection Interface to test it.
4.  Prepare the JAR and JAD files which are to be loaded over the air. Make sure that these files conform to the requirements listed in chapter 8.5 and that they represent a valid application which can be started by *at^sjra***.**
5.  Put the files (JAR and JAD) on the http Server. The files can either be publicly available or protected through basic authentication. When in doubt try to download the files from the server by using a common web browser on a PC, which can reach your http server over TCP/IP.
6.  Prepare the SM sender. The sender must be able to send SMs, which conform to chapter 8.4, to your device. When in doubt try to send "normal" SMs to your device which can than be read out from the AT command interface.
7.  Test with a local device. Send a suitable short message to your device, which completes the necessary parameter, sets and starts the operation. The operation is finished when the device reboots. You can now check the content of the file system and if the correct jar and jad files were loaded into the correct location.
8.  Analyze errors. If the above test failed, looking at your device's behavior and your http servers access log can give you a hint as to what went wrong:
    *   If the device did not terminate the running Java application and did not reboot, not even after the safety timeout, either your SM was not understood (probably in the wrong format) or it did not properly authenticate (probably used the wrong password) or your parameter set is incomplete for the requested operation.
    *   If the device terminated the running Java application, but did not access your http server, and rebooted after the safety timeout, there were most likely some problems when opening the network connection. Check your network parameters.
    *   If the device downloaded the jad and possibly even the jar file but then rebooted without saving them in the file system, most likely one of the errors outlined in chapter 8.5 occurred. These are also the only errors which will return a response. They are posted to the http server if the jad file contains the required URL.
9.  Start update of remote devices. If you were able to successfully update your local device, which is hopefully a mirror of all your remote devices, you can start the update of all other devices.

# 9 Compile and Run a Program without a Java IDE

This chapter explains how to compile and run a Java application without a Java IDE.

## 9.1 Build Results

A JAR file must be created by compiling an SMTK project. A JAR file will contain the class files and auxiliary resources associated with an application. A JAD file contains information (file name, size, version, etc.) on the actual content of the associated JAR file. It must be written by the user. The JAR file has the ".jar" extension and the JAD file has the ".jad" extension. A JAD file is always required no matter whether the module is provisioned with the Module Exchange Suite, as described in Section 7.1, or with OTA provisioning. OTA provisioning is described in Chapter 8.

In addition to class and resource files, a JAR file contains a manifest file, which describes the contents of the JAR. The manifest has the name manifest.mf and is automatically stored in the JAR file itself. An IMP manifest file for:

```
package example.mytest;
public class MyTest extends MIDlet
```

includes at least:

```
Manifest-Version: 1.0
MIDlet-Name: MyTest
MIDlet-Version: 1.0.1
MIDlet-Vendor: Siemens
MIDlet-1: MyTest, example.mytest.MyTest
MicroEdition-Profile: IMP-NG
MicroEdition-Configuration: CLDC-1.1
```

A JAD file must be written by the developer and must include at least:

```
MIDlet-Name: MyTest
MIDlet-Version: 1.0.1
MIDlet-Vendor: Siemens
MIDlet-1: MyTest, example.mytest.MyTest
MIDlet-Jar-URL: http://192.168.1.3/datafiles/MyTest.jar
MIDlet-Jar-Size: 1408
MicroEdition-Profile: IMP-NG
MicroEdition-Configuration: CLDC-1.1
```

A detailed description of these attributes and others can be found in the Java/MIDlet documentation http://java.sun.com/j2me/docs/alt-html/WTK104_UG/Ap_Attributes.html

## 9.2 Compile

- Launch a *Command Prompt*. This can be done from the *Programs* menu or by typing "cmd" at the *Run…* prompt in the *Start* menu.
- Change to the directory where the code to be compiled is kept.
- Compile the program with the SDK. Examples of build batch files can be found in each sample program found in the examples directory, *\Siemens\SMTK\<productname> \wtk\src\example.*
- If the compile was successful the program can be run from the command line. Examples of run batch files can be found in the examples directories listed above as well.

The batch files for compiling and running the samples refer to master batch files in the *…\Siemens\SMTK\<productname>\wtk\bin* directory and use the system environment variables IMPNG_JDK_DIR and IMPNG_DIR. IMPNG_JDK_DIR points to the root directory of the installed JDK and IMPNG_DIR points to the root directory of the Siemens-SMTK-<productname>-IMPNG installation. The installation process sets these environment variables. A modification is usually not necessary. They may be modified as requested (e.g. when switching to a different JDK) via the advanced system properties.

## 9.3 Run on the Module with Manual Start

- Compile the application at the prompt as discussed in Section 9.2 or in an IDE.
- Transfer the .jar and .jad file from the development platform to the desired directory on the module using the Module Exchange Suite or OTA provisioning. Chapter 7 explains how to download your application to the module.
- Start a terminal program and connect to ASC0.
- The command *at^sjra* is used to start the application and is sent to the module via your terminal program. Either the application can be started by .jar or by .jad file.

Example:
In your terminal program enter: *at^sjra=a:/java/jam/example/helloworld/helloworld.jar*
If you prefer to start with .jad file: *at^sjra=a:/java/jam/example/helloworld/helloworld.jad*

The Flash file system on the module is referenced by "a:".
Depending on which file you specify the java application manager tries to find the corresponding file in the same directory. This search is not done by name, but by comparing the contained attributes. The first file which contains the same values for MIDlet-Name, MIDlet-Version and MIDlet-Vendor is used.

## 9.4 Run on the Module with Autostart

- Compile the application at the prompt as discussed in Section 9.2 or in an SMTK integrated IDE.
- Transfer the .jar and .jad file from the development platform to the desired directory on the module using the Module Exchange Suite or OTA provisioning. See Chapter 7.

## 9.4.1    Switch on Autostart

- There is an AT command, *at^scfg*, for configuring the autostart functionality. Please refer to the AT Command Set [3].
- Restart the module.

## 9.4.2    Switch off Autostart

There are three methods for switching off the autostart feature:
- the *at^scfg* AT command, or
- the graphical "autoexec_off.exe" tool (included in the Installation CD software under wtk/bin), or
- the command line "cmd_autoexec_off.exe" tool (included in the Installation CD software under wtk/bin)

To disable the automatic start of a user application in a module these steps must be carried out:

***Using the graphical "autoexec_off.exe" tool:***

1. Connect the module to the PC
2. Make sure, that the module is switched off
3. Start the "Autoexec_Off" program
4. Select the COM-Port
5. Press the "AutoExec Off" button

***Using the command line tool  "cmd_autoexec_off.exe":***

1. Connect the module to the PC
2. Make sure, that the module is switched off
3. Start the command line tool "cmd_autoexec_off" with option <com-port> (com1 up to com99 for com port selection)

# 10 Compile and Run a Program with a Java IDE

## 10.1 Eclipse 3.1.2 (with ME Plugin 1.2.3)

The following description assumes that Eclipse 3.1.2 is installed from SMTK CD as described in chapter 3.3.

### 10.1.1 Setup a New Project

***Step 1:*** Create new Java Project
- Select Eclipse menu *File->New->Other…*
- Select J2ME Midlet Suite:
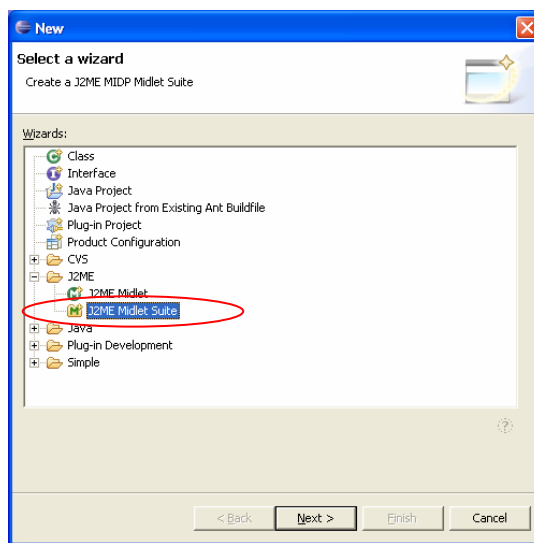


Figure 22: Create new Eclipse project: Create a J2ME MIDP Midlet Suite

- Type in the used name (e.g. Project01) and path for the new project:
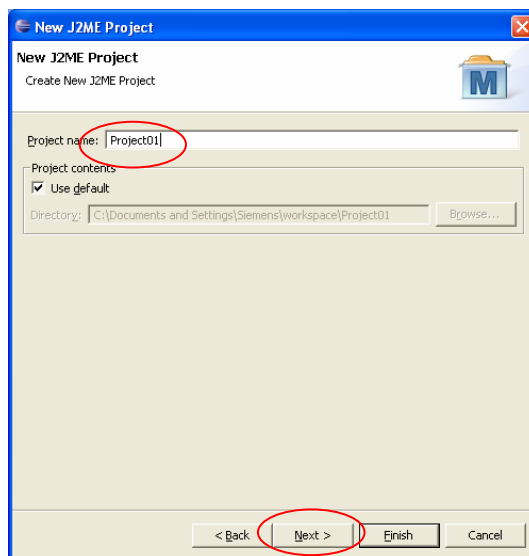


Figure 23: Create new Eclipse project: New J2ME Project

- Select e.g. TC65 Release 2 emulator:
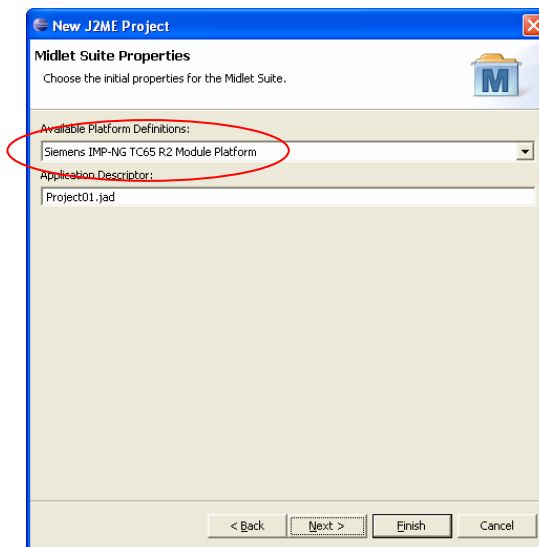


Figure 24: Create new Eclipse project: Midlet Suite Properties

- A new Java Project has been created:
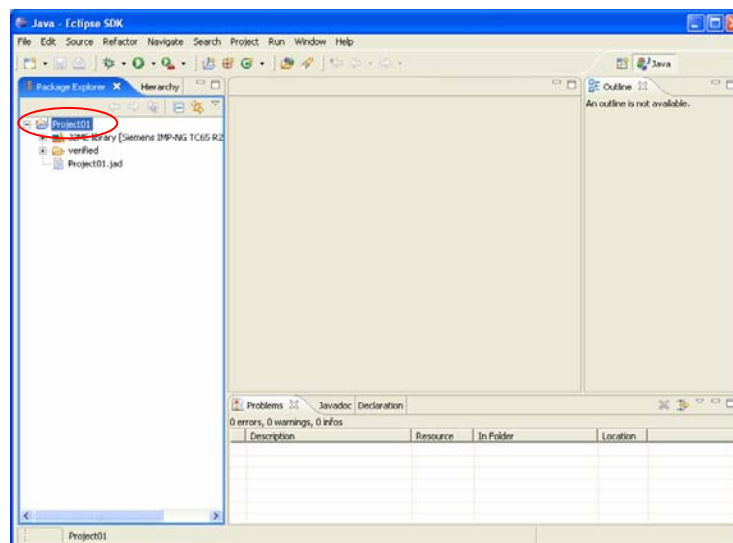


Figure 25: Create new Eclipse project: work area with new created project

- Select "Project01" and press <Ctrl+N> and select "J2ME Midlet":
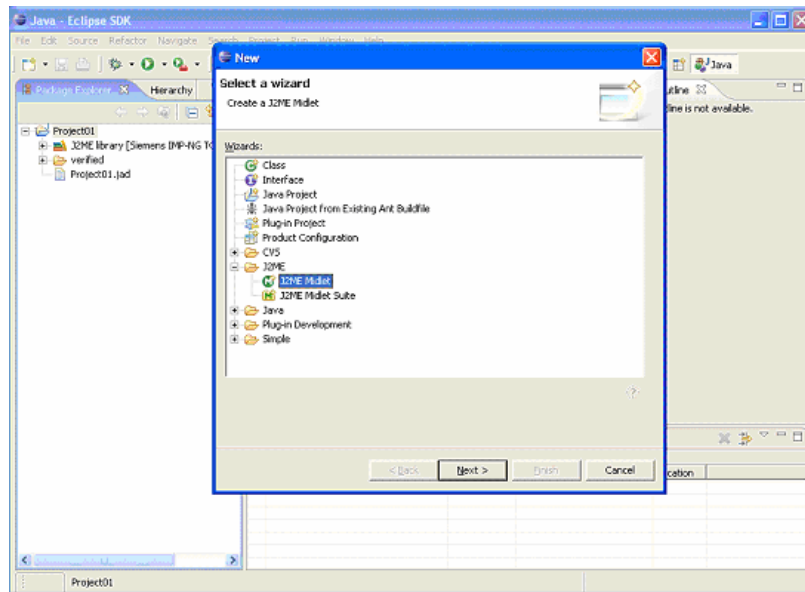


Figure 26: Create new Eclipse project: Create a J2ME Midlet

- Fill in Midlet parameters (package directory, name of Java source file) and press <Finish>:
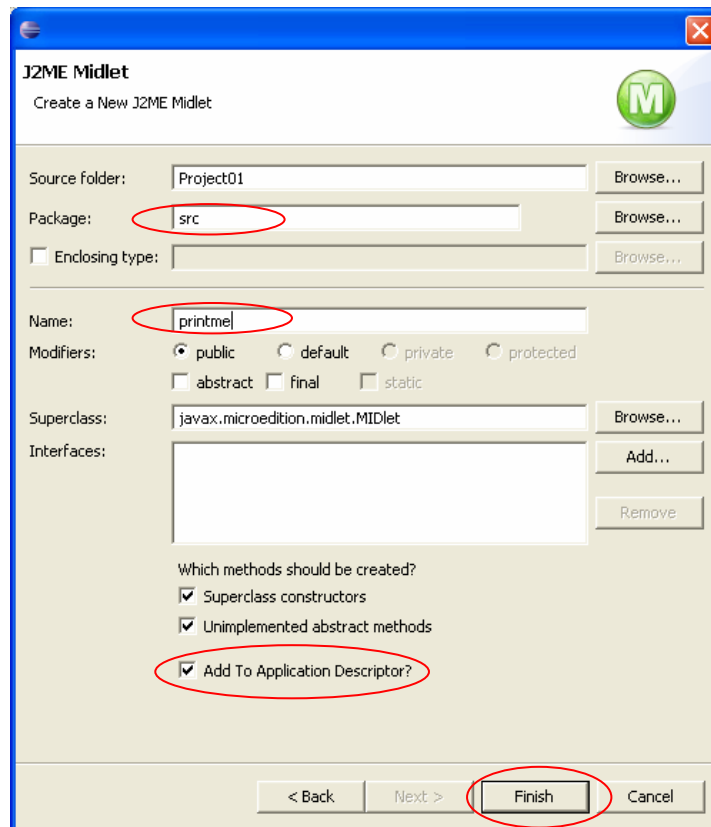


Figure 27: Create new Eclipse project: Create a New J2ME Midlet

***Step 2:***      Modify Java source file "printme.java" template:
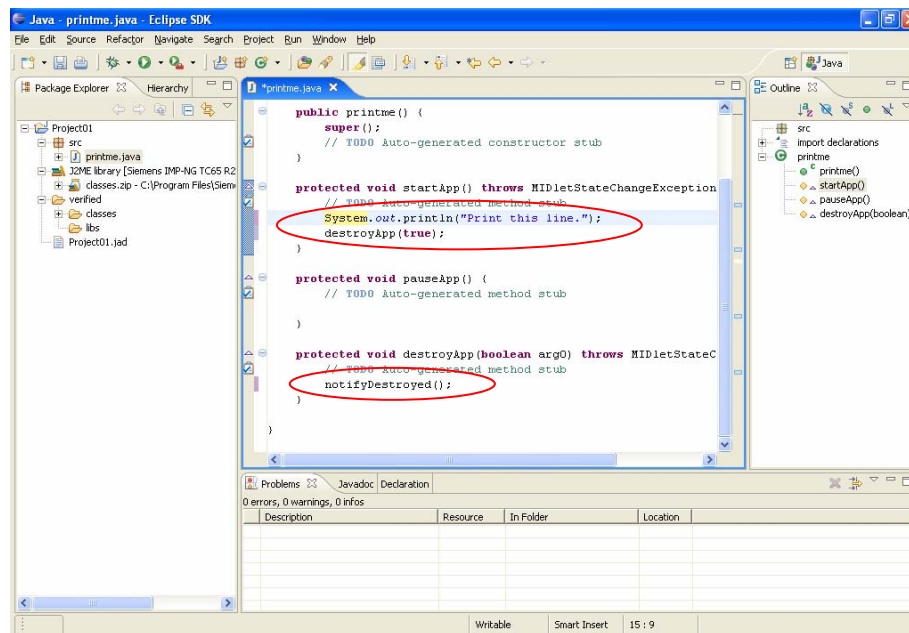


Figure 28: Create new Eclipse project: Edit some Java commands

This application prints the given line via ASC1 (default setting)

Please keep in mind to add "notifyDestroyed();" in destroyApp() method.
Please refer to chapter "6 MIDlets" for more details about Midlet life cycles.

***Step 3:***      Configure Jad file.

- Double click on "Project01.jad" to open the editor and add "deployed\" path
  in Midlet Jar URL edit line.
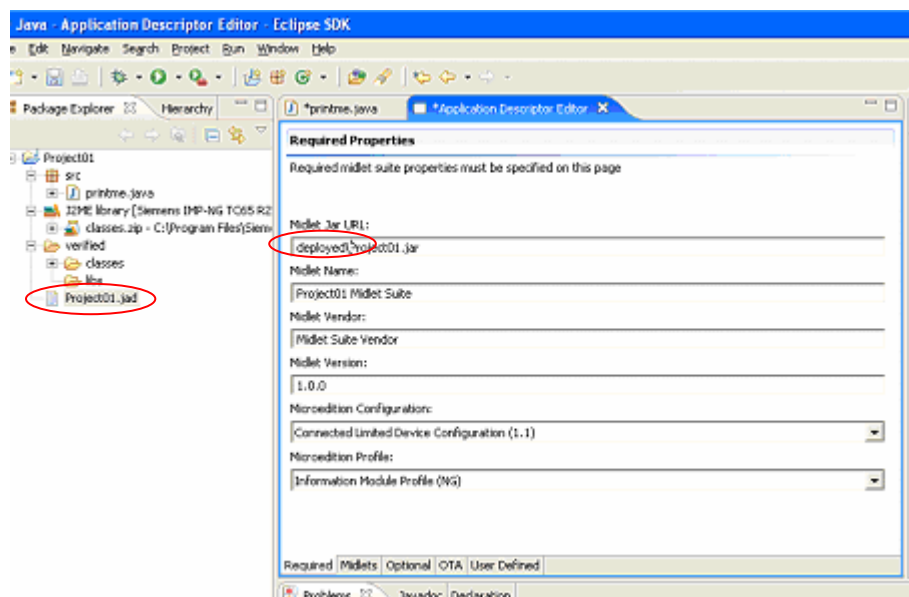


Figure 29: Create new Eclipse project: Edit "deployed" path to Jar file

***Step 4:***        Create Jad and Jar file.

         - Select "Project01"
         - Open context menu of "Project01" and select *Create Package* from item
           *J2ME*
         - Jad and Jar file are be created under the "deployed" directory (default
           setting of Eclipse)

***Step 5:***        Download Jad and Jar file  into the Flash File System of the module and start
         the Java application.

         a) Manual download and start of Java application:
         - Copy Jad and Jar files manually using MES (e.g MES Windows Explorer)
           into the module
         - Start the Java application by using at command "at^sjra=a:\Project01.jar"
           (e.g. using ASC1)
         - The Java "System.out" is printed via ASC1 (default setting)  and is displayed
           on terminal programs (e.g. "Print this line.").
         b) Automatic download and start of Java application using IDE:
         - Open context menu of "printme.java" and select *Emulated J2ME Midlet* from
         item *Run As*
         - The SMTK emulator downloads Jar and Jad file into the Flash File of the
           module (transmission via MES port that was selected during installation) and
           automatically starts the Java application.
         - The Java "System.out" is printed via ASC1 (default setting)  and is displayed
           on terminal programs (e.g.  "Print this line.").

Further information about importing an existing Java project into Eclipse based on the "Hello World" example is described in chapter "11.3.2.5".

# 11 Debug Environment

Please note that this section is not intended as a tutorial in debugging or how to use Netbeans IDE 5.0, Borland JBuilder or Eclipse. Documents for these IDEs can be found on their respective homepages. Once the proper emulator has been selected (as described in the relevant IDE sections below), your Java application can be built, debugged and executed.

## 11.1 Data Flow of a Java Application in the Debug Environment
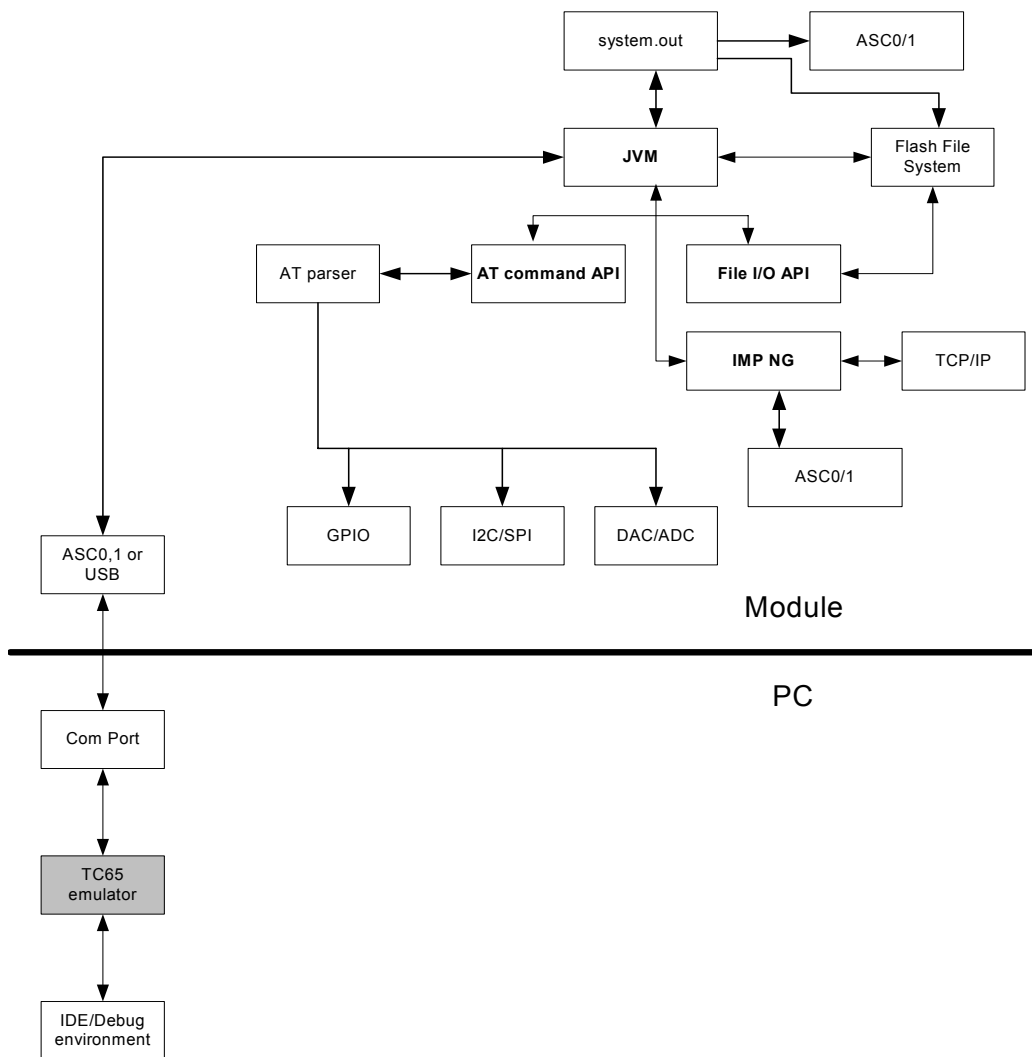


Figure 30: Data flow of a Java application in the debug environment

In the debug environment the module is connected to a PC via a serial interface. This can be a USB or an RS232 line. The application can then be edited, built, debugged or run within an IDE on the PC. When running or debugging the MIDlet under IDE control it is executed on the module (on-device execution) not on the PC. This can be either debugging mode, where the midlet execution can still be controlled from the IDE (on-device debugging) or normal mode, where the midlet is copied to the module and started normally. This ensures that all interfaces behave the same whether debugging mode is used or not.

## 11.2　Emulator

The ME emulator is part of the SMTK and is used as the controlling entity for on-device debugging. Some values can be configured in the file wtk/bin/WM_Debug_config.ini. The emulator is running fine without changes in WM_Debug_config.ini file, because the settings for the serial interface (COM port and baudrate) are configured during installation of SMTK.

Debugging information between the Debugger (IDE) and the JVM is transferred over an IP connection. In order to establish this IP connection between the PC and the module the emulator needs a special Dial-Up-Network (DUN):

- ISP name: "IP connection for remote debugging"
- Modem: either "Standard 19200 bps Modem" or "<productname> 14400 bps Modem"
- Phone number: *88#
- Disable the "Redial if line dropped" option.
- Enable "Connect automatically"

This Dial-Up Network (DUN) connection is installed automatically together with a standard modem during installation of SMTK. The emulator use always the COM port settings configured for this Dial-Up Network connection.

You can use any of the three serial interfaces (ASC0, ASC1, USB) to connect with module, but you will lose the functionality which is normally present on the interface. Because of this loss and because of its speed it is recommended that the USB interface be used.

Please keep in mind, that if you like to use an USB modem connection, you have to install and configure that modem connection after SMTK installation manually. There are some points to notice in case of using the USB interface in a debugging session. If you are starting a debugging session very quickly once again after end of a previous debugging session, the emulator is not able to select the USB interface and is selecting a standard modem connection. Please wait some seconds to avoid this operation system dependent problem, because the Windows operation system needs some seconds to enable the USB port once again after the "IP connection for remote debugging" is closed.

If necessary, the IP addresses used for the debug connection can also be changed. This is done in the file wtk/bin/WM_Debug_config.ini. For details, see also the at^scfg command and its "Userware/DebugInterface" parameters described in [3]. Please keep in mind, that the IP address range 10.x.x.x is not supported for the configuration of debugging!

During installation of SMTK some new programs are installed for handling the debugging session in conjunction with the IDE. The installation routine of the SMTK doesn't change any configuration of an existing firewall on your PC.
In the case, that a firewall is installed on your PC and the local configured and used IP connection (Dial-Up Network connection for debugging) is blocked or disturbed by this firewall, please configure the firewall or the Dial-Up Network connection manually to accept the new installed programs and the port or to use another port or contact your local PC administrator for help.

## 11.3    Java IDE

The SMTK is integrated into your Java IDE during installation. Please note that the IDE integration is intended to create MIDlets suitable for the Java enabled module and for debugging using the emulator. JAR files used in the module must be configured according to the batch file examples given. If the SMTK install succeeded, you can easily switch between the Siemens environment and Standard-JDK environment, the special libraries/APIs and emulators are available, and AT commands can be sent to the module. Regular function of the IDE for non-Siemens projects is unchanged.

Using the debugger please keep in mind that the MIDlet-URL, included in the Jad file, has to indicate and store the location where the emulator will find the corresponding Jar file. Generally the location will only be the file name for the Jar file.

If you are using Eclipse IDE the location of the Jar file is "deployed\<filename>.jar". Please check this path name inside the Jad file before starting a debugging session with Eclipse IDE and change it manually, if the "deployed" subdirectory is missing. Please keep in mind, that the subdirectory "deployed" is used as a default setting in the Eclipse IDE and can be changed by the user within the Eclipse menu.

While using "on-device debugging" the ME is restarted after the end of each debugging session. This is independent of the used IDE (Eclipse 3.0.1, Eclipse 3.0.2, Eclipse 3.1.0, Eclipse 3.1.1, Eclipse 3.1.2, NetBeans IDE 5.0, JBuilder X, JBuilder 2005, JBuilder 2006 Enterprise/Developer).

Please keep in mind, that it is not possible to use obfuscated files for a debugging session.

When writing and debugging a midlet be aware of the midlet life cycle (see chapter "6.2 MIDlet Life Cycle").

## 11.3.1   NetBeans IDE 5.0

This section indicates the changes to your IDE you will see after integrating the SMTK and describes how to exploit these features to build and debug your applications.

All installed emulators can be found using the NetBeans IDE 5.0 menu *Tools -> Java Platform Manager.* See Figure 31.
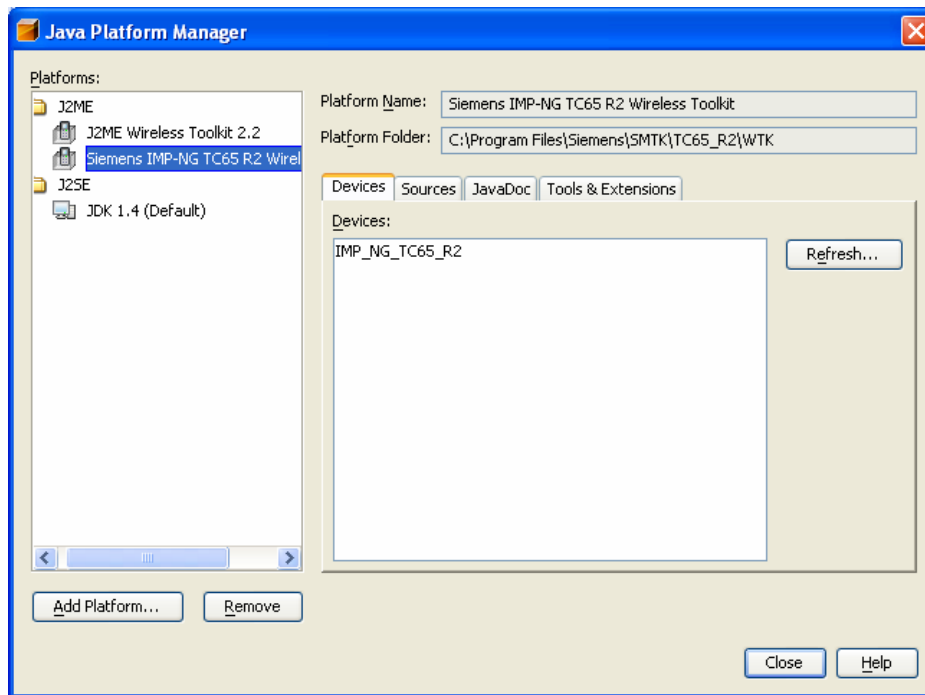


Figure 31: NetBeans IDE 5.0 - installed emulators

## 11.3.1.1 Switching emulator to IMP-NG emulator

You can switch to the SMTK emulator project dependent using the context menu for project properties (e.g. "HelloSample" <properties>) and select the emulator platform and the device e.g. "*Siemens IMP-NG TC65 R2 Wireless Toolkit*" and "*IMP_NG_TC65_R2*" for TC65 Release 2. See Figure 32.
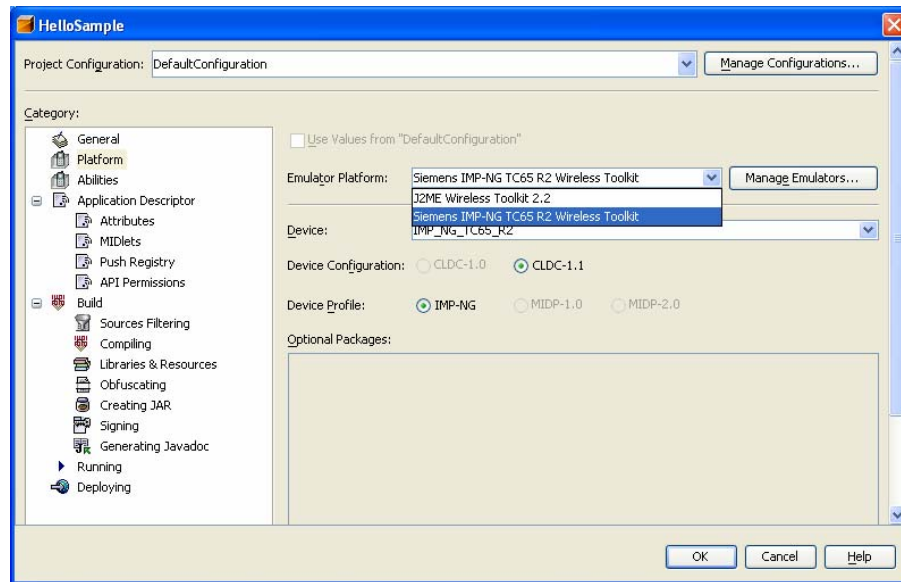


Figure 32: NetBeans IDE 5.0 - Switching to IMP-NG emulator

### 11.3.1.2 Templates

Templates for a Siemens MIDlet can be found in the file explorer and under NetBeans IDE 5.0 menu *File-> New File...* The MIDlet template provides the skeleton of an IMP-NG MIDlet application.

A wizard "New File" will be opened. Please select category "IMP-NG" and File Type "IMP-NG MIDlet" and press <Next> button for working with this skeleton.
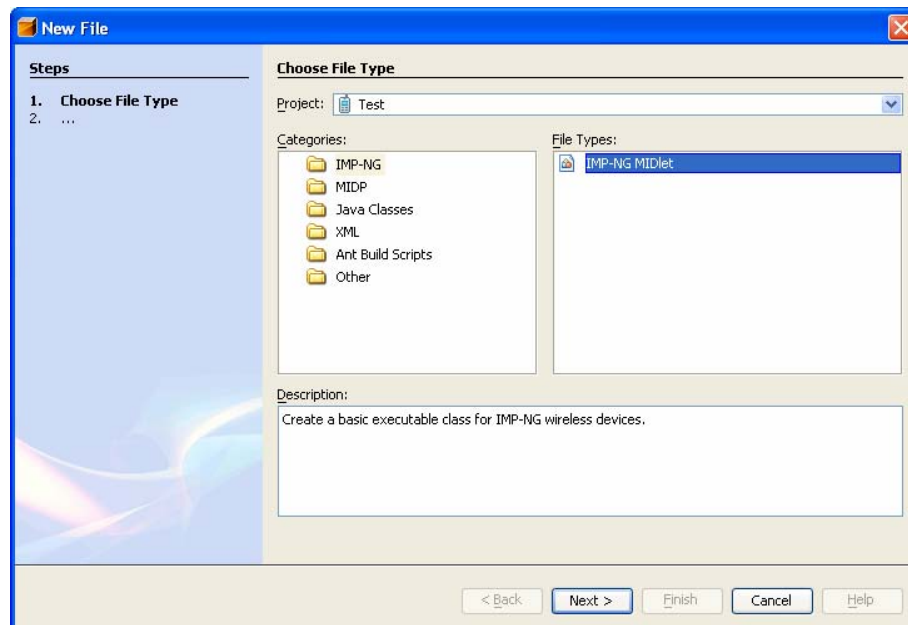


Figure 33: NetBeans IDE 5.0 - Selecting an IMP-NG MIDlet template (e.g. project "Test")

### 11.3.1.3   Examples

There are three IMP-NG MIDlet sample projects ("HelloSample", "NetDemo" and "RS232Sample") included. Each sample project can be selected directly in NetBeans IDE 5.0.

You can open the sample project (e.g. "HelloSample") using the NetBeans IDE 5.0 menu *File -> New Project...* A "New Project" wizard will be opened. Please select Category *Samples -> Mobile -> IMP-NG* and select (e.g. "Hello World Example").
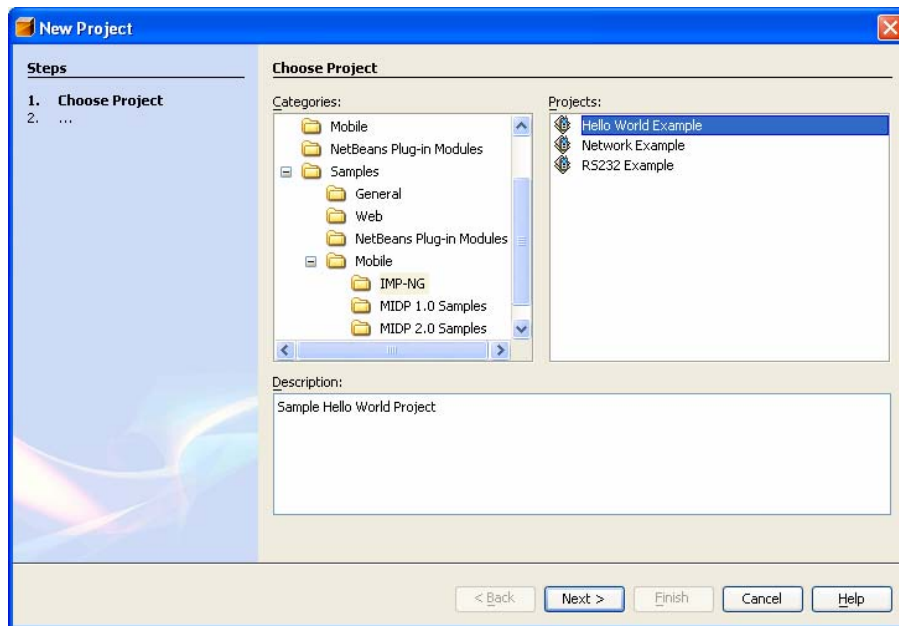


Figure 34: NetBeans IDE 5.0 - selecting sample project "Hello World Sample"

Please press <Next> button.
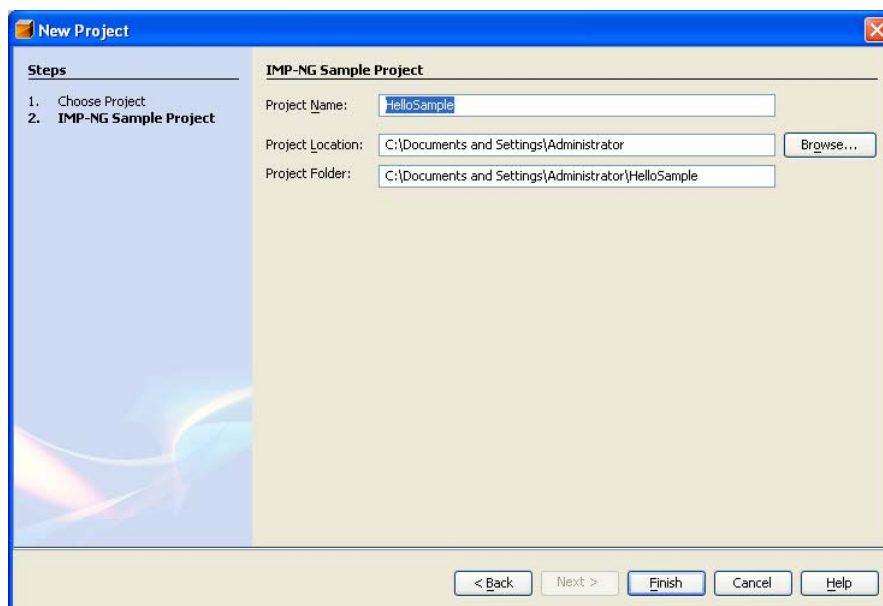


Figure 35: NetBeans IDE 5.0 - confirmation of sample project "Hello World Sample"

Please press <Finish> button to end selection of the sample project.

## 11.3.1.4   Compile and run

Ensure that the proper emulator will be used and compile the project as normal. Any output will be shown in the output window in the IDE together with some debugging information of the IDE. The html help files of the SMTK can be accessed directly by pressing Alt+F1.

## 11.3.1.5   Starting debugging session without download of Java files

It is possible to start a debugging session without download of the Java files (*.jad and *.jar) again. The emulator has to be started with an additional parameter "-noload" and downloading of the Java files is suppressed.

If you like to suppress downloading of *.jar and *.jad file, please use the project context menu *Properties*,  select category "Running" and type in "-noload" as extra parameter for the emulator as shown in the following figure:
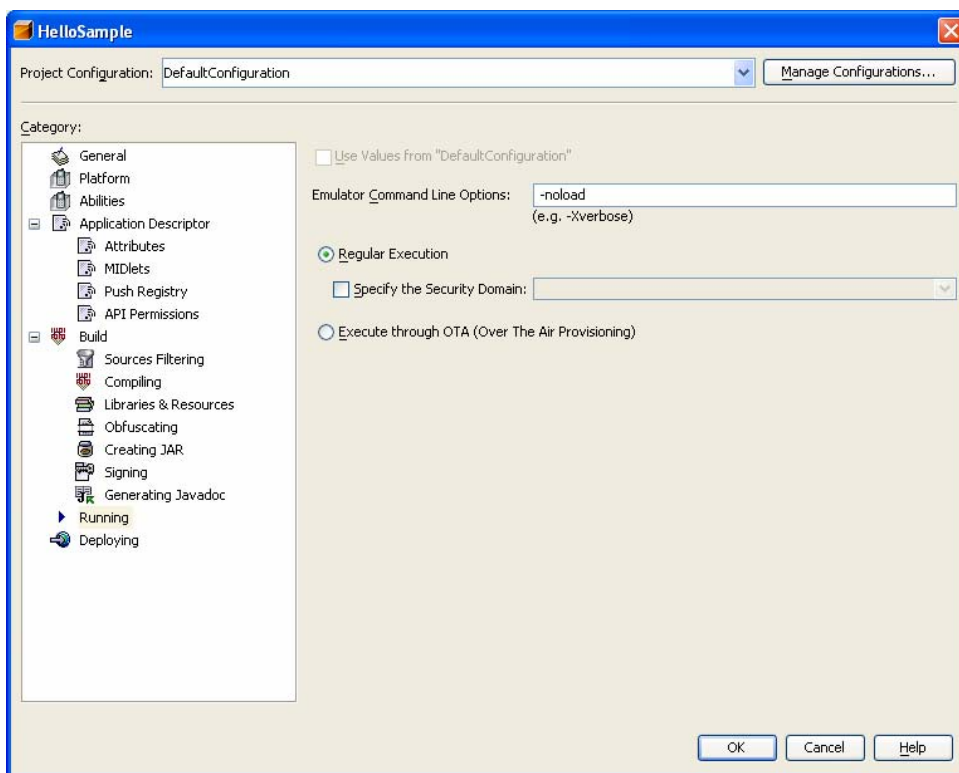


Figure 36: NetBeans IDE 5.0 - add emulator option "-noload" (e.g. project "HelloSample")

Please keep in mind that you have to remove this emulator option before starting debugging session with download of Java files. The Java files are not updated in the Flash Files System of the module while option "-noload" is set!

Remark:
This feature is not working in current NetBeans IDE 5.0 version (included on the SMTK CD).

## 11.3.2 Eclipse 3.0 and Eclipse 3.1

We recommend that Eclipse 3.0.1 or higher is to be used.

### 11.3.2.1 Eclipse 3.0

Please use the customized Eclipse ME Plugin version that is distributed with the SMTK CD under the directory "EclipseMEplugin_WM" only for Eclipse 3.0.1 and 3.0.2 and install Eclipse ME as described in chapter "3.3". However, instead of selecting "eclipseme.feature_1.2.3_site.zip" choose "*\EclipseMEpluging_WM\plugin.zip*" from SMTK CD after clicking the *New Archived Site…* button.

### 11.3.2.2 Eclipse 3.1

If Eclipse is not installed, please follow the steps as described in chapter "3.3".

Please use and install EclipseME plugin 1.2.1 or 1.2.3 for Eclipse 3.1.0, 3.1.1 or 3.1.2.

### 11.3.2.3 Additional Configuration of integrated emulator

The following configuration is displayed after SMTK integration into Eclipse by using the Eclipse menu *Window→Preferences→J2ME→Platform Components*:
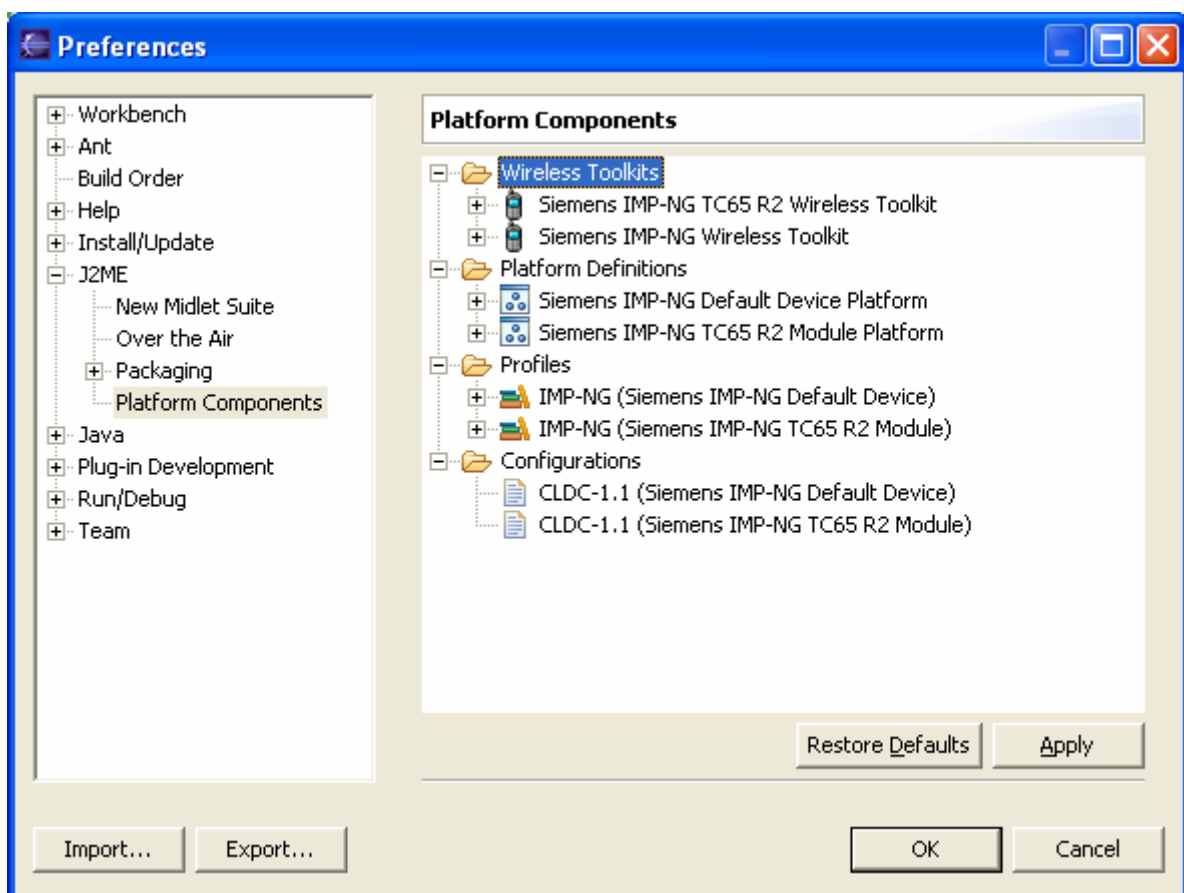


Figure 37: Eclipse – Display of different integrated emulators

To use the debugger it is necessary to increase the debug server delay timeout of Eclipse IDE. Please set the timeout under *Windows→ Preferences→J2ME → Debug Server Delay* to 15000. If you develop an extremely large application you may have to increase this timeout.

### 11.3.2.4 Switching Emulators

You can easily switch between different installed emulators by using the preferences of a project and choosing e.g. *Siemens IMP-NG TC65 R2 Module*, see figure below.
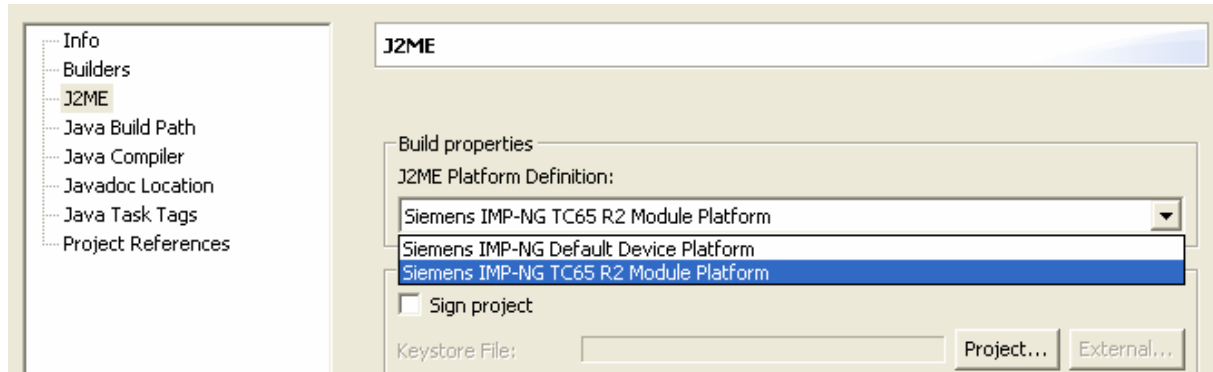


Figure 38: Eclipse – J2ME platform

### 11.3.2.5 Import Example

You can import an existing Eclipse project using the following steps:
  - Open menu: *File→Import→Existing Project into Workspace*
  - choose the root directory of the Eclipse project (e.g. "Hello World" example of the SMTK CD) (see figure below)
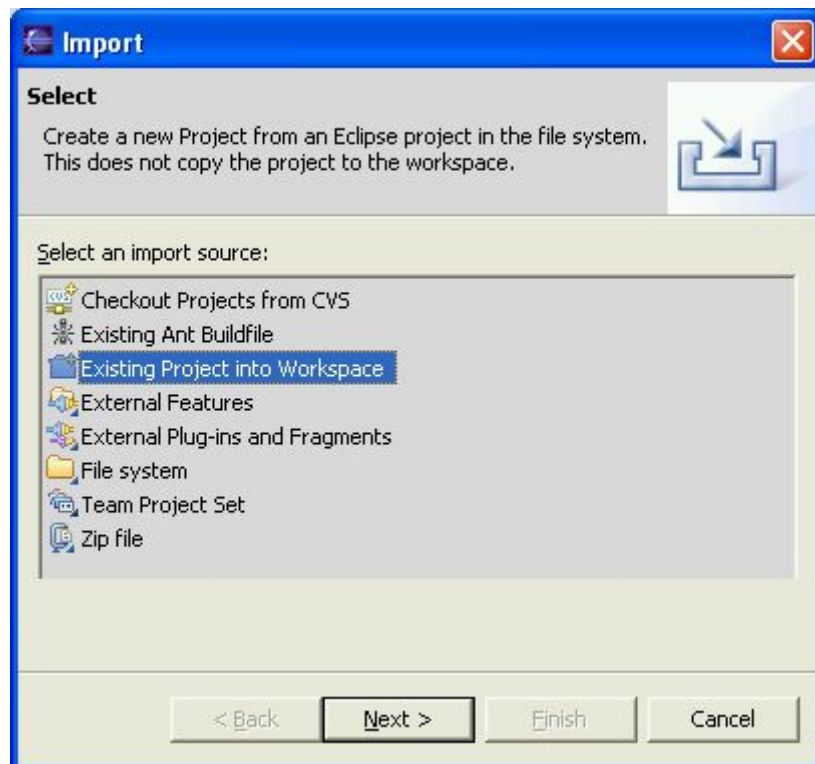


Figure 39: Eclipse – Project import

The following figure shows the "Hello World" example in the IDE.
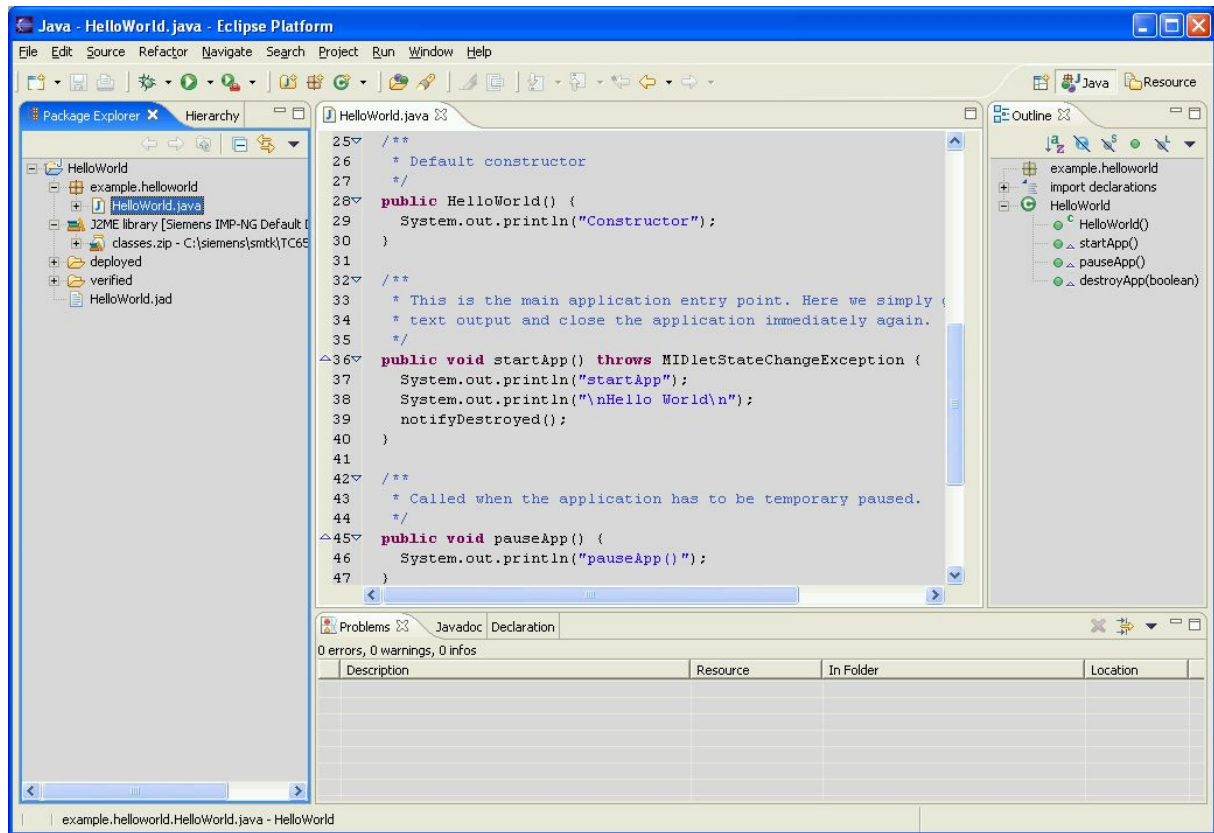


Figure 40: Eclipse - Example

The html help files of the SMTK can be accessed directly by pressing Shift+F2 while the cursor points to a Java expression in a Java source file.

## 11.3.2.6   Compile and debug

To build the jar and jad files you have rebuild the project with the "create package" function. Open the context menu of the project and go to *J2ME→Create Package*
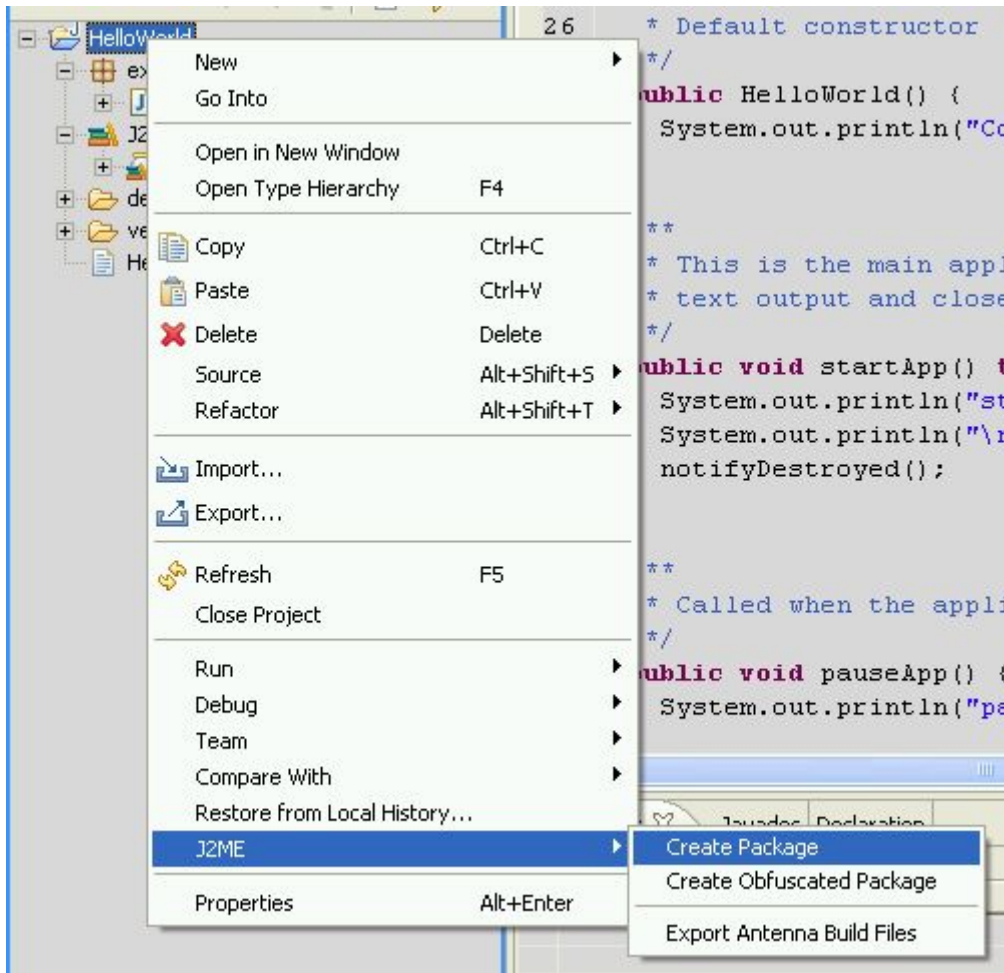


Figure 41: Eclipse – Create package

Now you are ready to debug your project. *Run→Debug…*

An emulator launcher with pre-settings is provided for starting the debugging session. Please ensure that you have selected the right project and executable midlet.
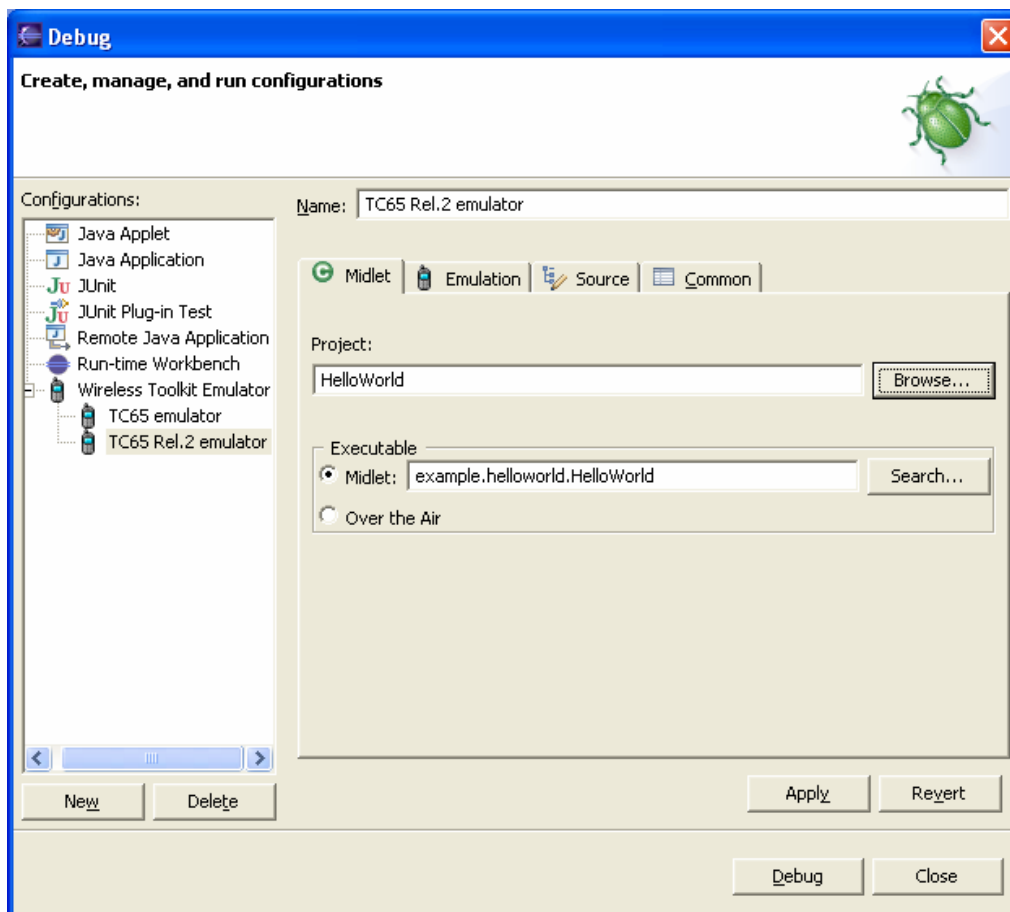
Figure 42: Eclipse - Configuration

Restriction:
The integration of the SMTK using the SMTK installer depends selecting the offered default workspace "\workspace" of Eclipse (e.g. Eclipse 3.0.1 and 3.0.2 "..\eclipse\workspace\.." or Eclipse 3.1.0, 3.1.1 and 3.1.2 "..\Documents and Settings\<user name>\workspace\..").

If you like to use the SMTK integration inside other workspaces, please integrate SMTK manually in these workspaces using Eclipse menu *Window→Preferences→J2ME→Platform Components* (as shown in Figure 37: Eclipse – Display of different integrated emulators). Please keep in mind that all manually integrated SMTKs have to be removed manually as well in case of removing SMTK.

### 11.3.2.7  Starting debugging session without download of Java files

It is possible to start a debugging session without download of the Java files (*.jad and *.jar) again. The emulator has to be started with an additional parameter "-noload" and downloading of the Java files is suppressed. Please add the parameter in the following emulator configuration line as described in the figure above:
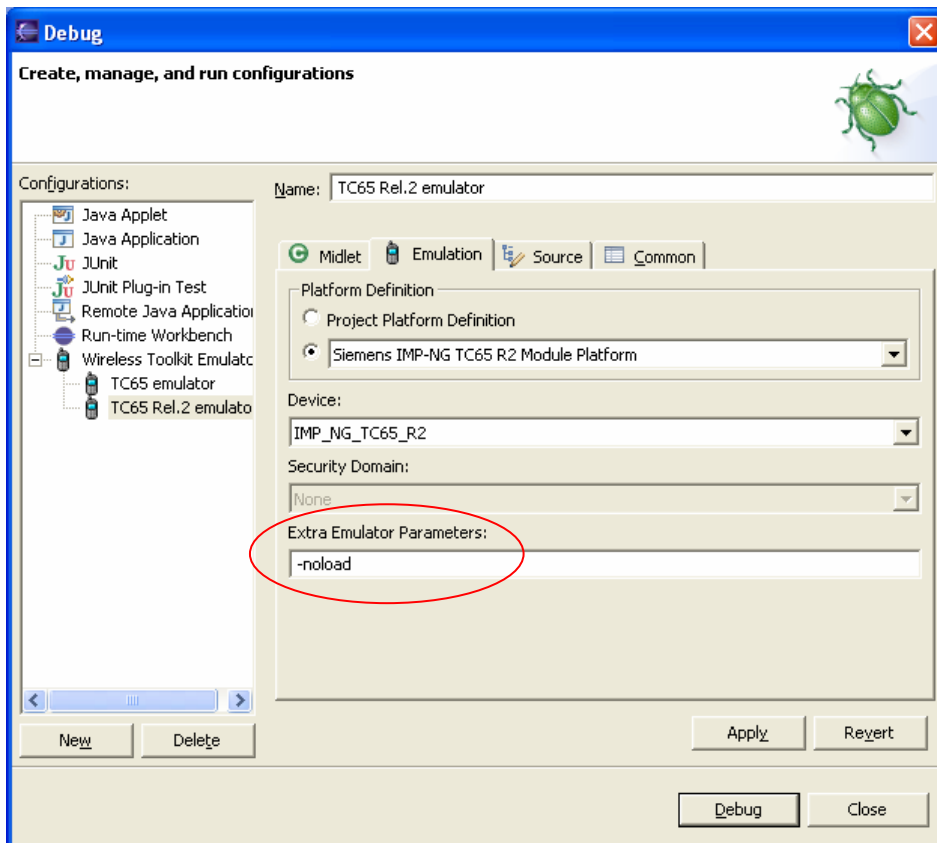


Figure 43: Eclipse - Configuration

Please keep in mind that you have to remove this emulator option before starting debugging session with download of Java files. The Java files are not updated in the Flash files system of the module while option "-noload" is set!

## 11.3.3    Borland JBuilder X

If you want to use JBuilder X and it is not installed, first install JBuilder X and follow the installation wizard instructions. Run the SMTK installation program in maintenance mode. The SMTK will find the installed JBuilder X IDE. Select JBuilder X to integrate the SMTK into JBuilder X.

After integration of SMTK into JBuilder X you can examine the integration by opening the menu *Tools -> Configure JDKs…* (see Figure below)
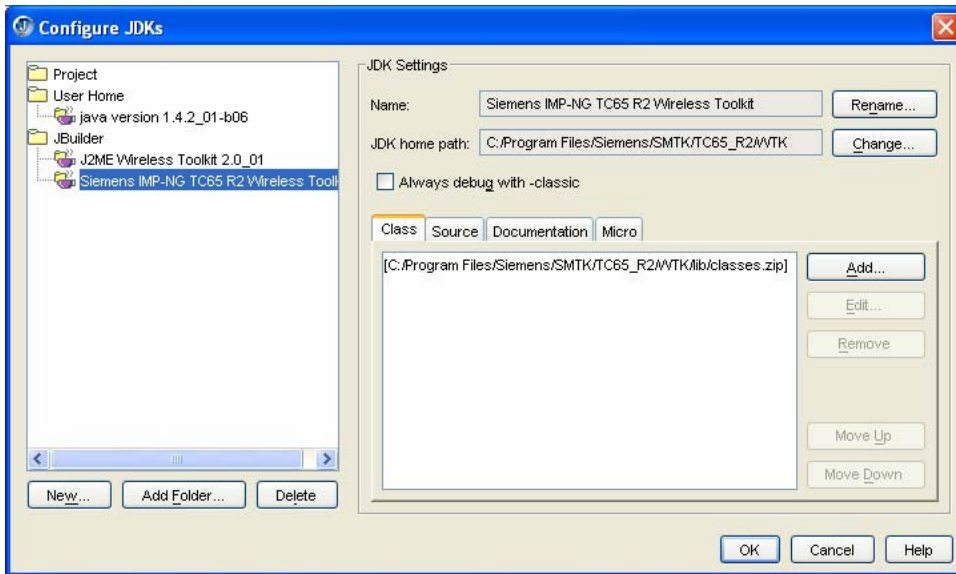


Figure 44: JBuilder X – JDK settings

The libraries included with the SMTK can be examined by opening the menu *Tools →  Configure Libraries…* (see Figure below)
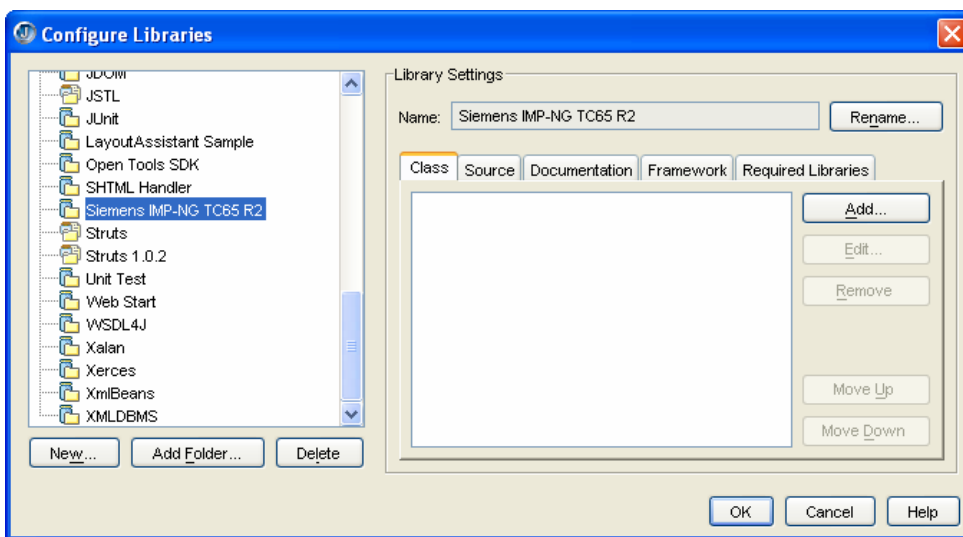


Figure 45: JBuilder X – Siemens Library

### 11.3.3.1 Examples

There are sample projects provided with the SMTK. These projects can be found in the JBuilderSamples directory of the SMTK installation directory. This directory is accessed by opening a project using the menu *File → Open Project…* (see Figure below)
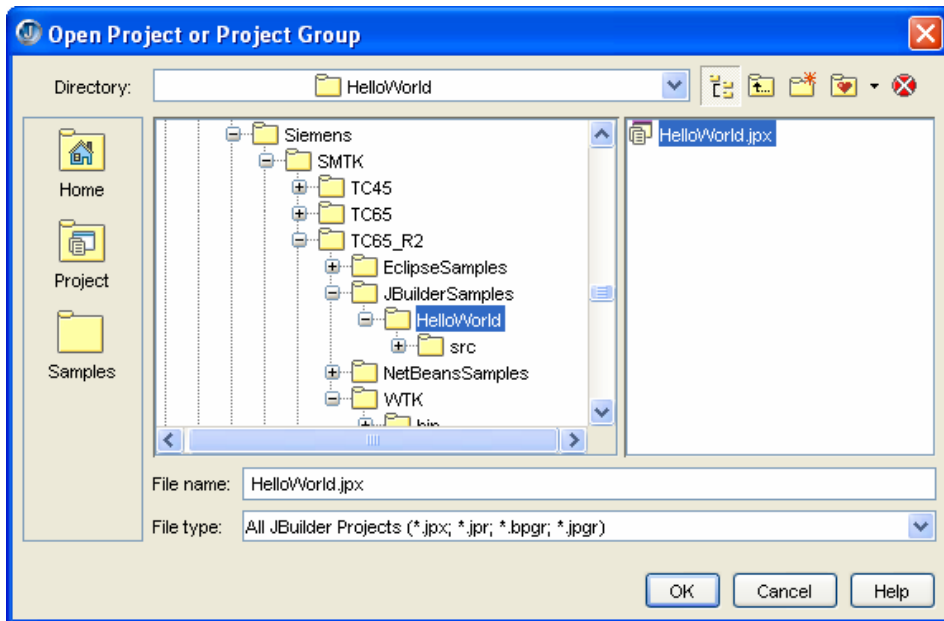


Figure 46: JBuilder X – Sample Projects

Open the Project (e.g. "HelloWorld.jpx"), rebuild the sources and start the debugger using the micro edition (context menu *"HelloWorld.jad"* → *Micro Debug using "HelloWorld"*).
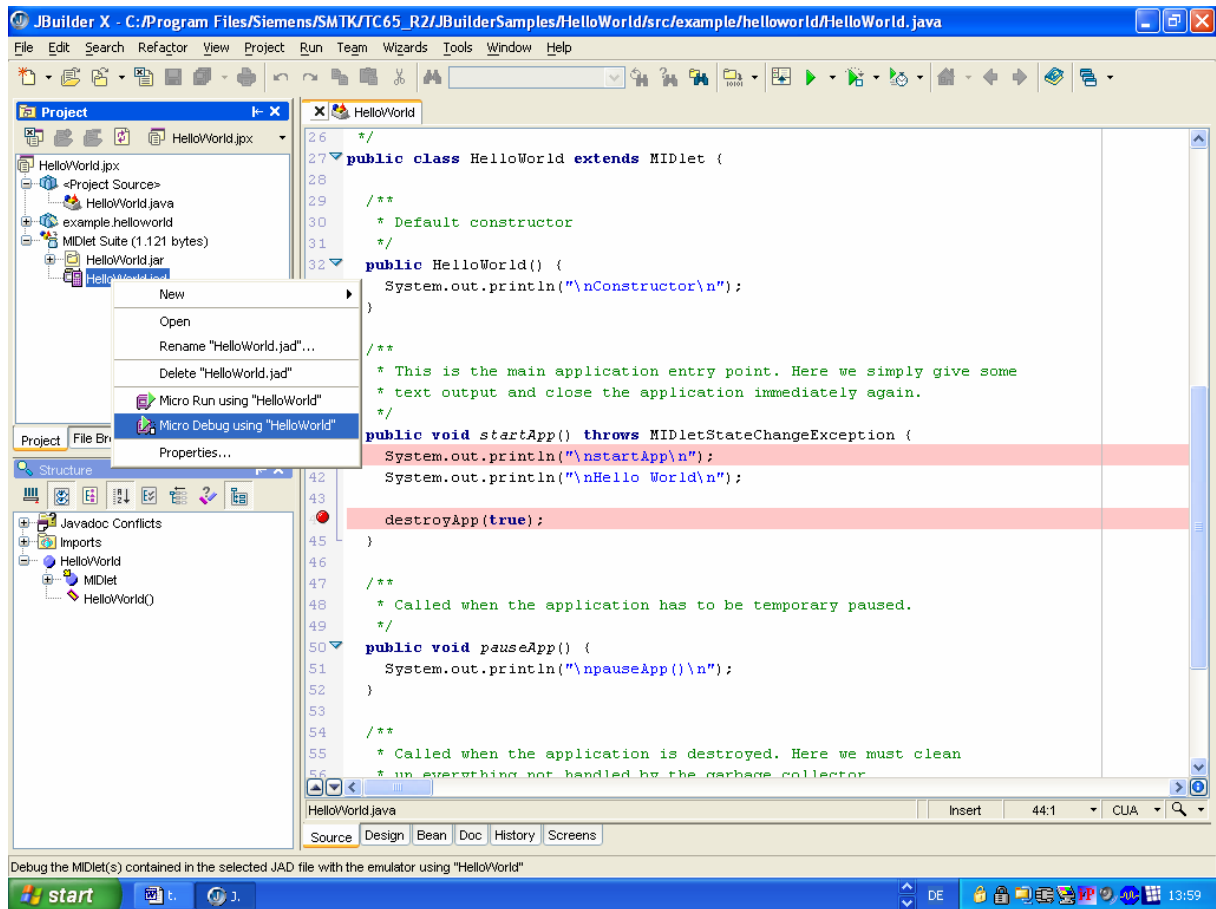


Figure 47: JBuilder X – Starting the debugging session

## 11.3.3.2 Starting debugging session without download of Java files

It is possible to start a debugging session without download of the Java files (*.jad and *.jar) again. The emulator has to be started with an additional parameter "-noload" and downloading of the Java files is suppressed. Please add the parameter in the following emulator running configuration line as described in the figures above.

Select the project properties (e.g. "HelloWorld.jpx") and edit the emulator configuration.
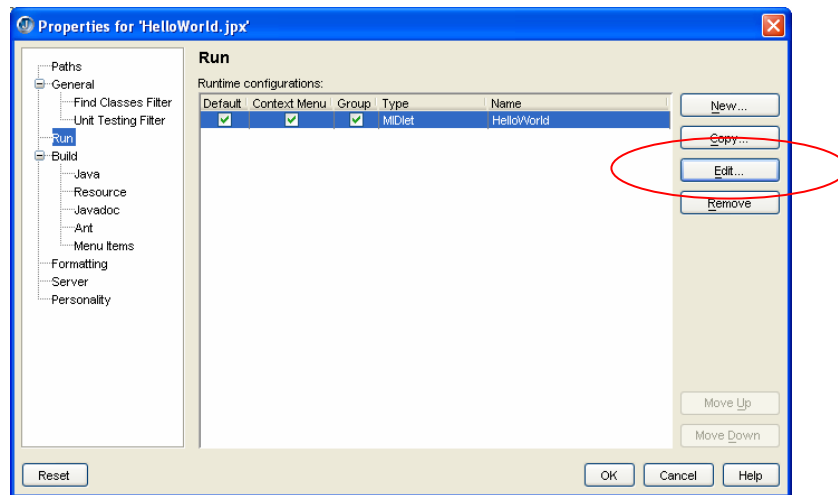


Figure 48: JBuilder X – Edit project properties for starting the emulator

Please add additional emulator parameter "-noload" and press "OK".
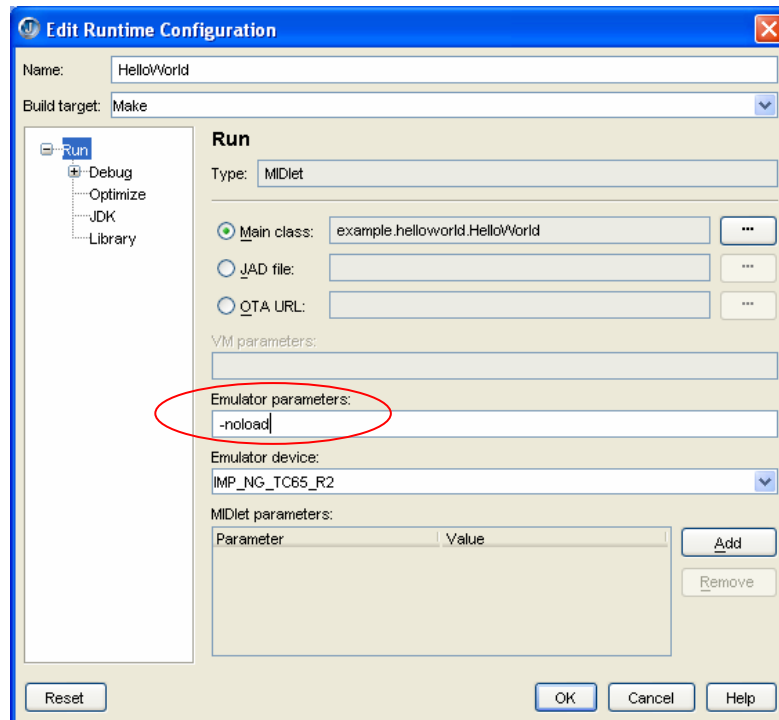


Figure 49: JBuilder - Runtime Configuration

Please keep in mind, that you have to remove this emulator option before starting debugging session with download of Java files. The Java files are not updated in the Flash files system of the module while option "-noload" is set!

The same procedure is used for all different JBuilder IDE versions.

## 11.3.4 Borland JBuilder 2005 and JBuilder 2006 Enterprise/Developer

If JBuilder 2005 or JBuilder 2006 Enterprise/Developer is not installed, run the JBuilder 2005 or JBuilder 2006 Enterprise/Developer install program and follow the installation wizard instructions. Use the SMTK installation routine in maintenance mode. The SMTK will find the installed JBuilder 2005 or JBuilder 2006 Enterprise/Developer IDE. Select JBuilder 2005 or JBuilder 2006 Enterprise/Developer to integrate the SMTK into JBuilder 2005 or JBuilder 2006 Enterprise/Developer.

After the integration of the SMTK into JBuilder 2005 or JBuilder 2006 Enterprise/Developer you can examine the integration by opening the menu *Tools -> Configure → JDKs…* (see Figure below)
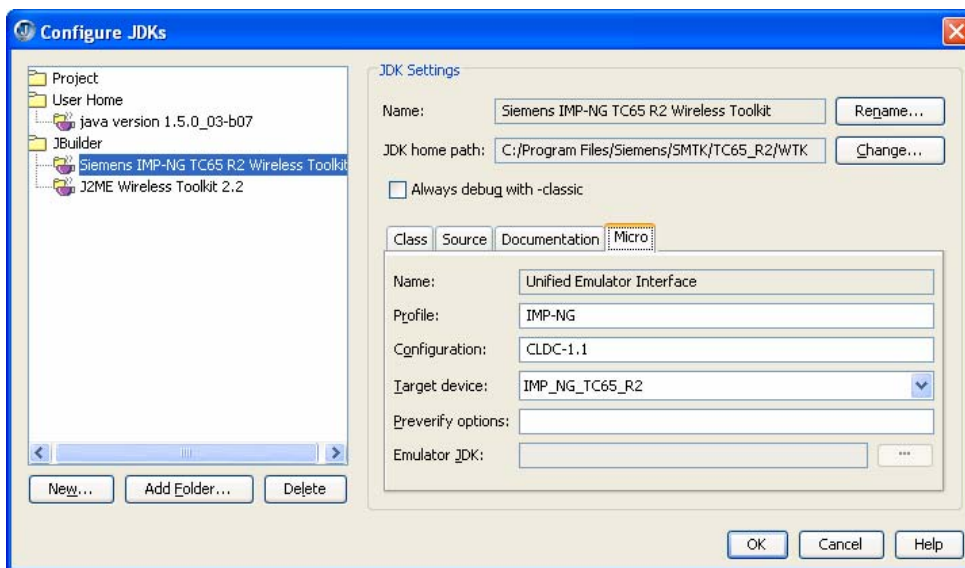


Figure 50: JBuilder 2006 Enterprise/Developer – JDK settings

The libraries included with the SMTK can be examined by opening the menu *Tools →
Configure → Libraries…* (see Figure below)
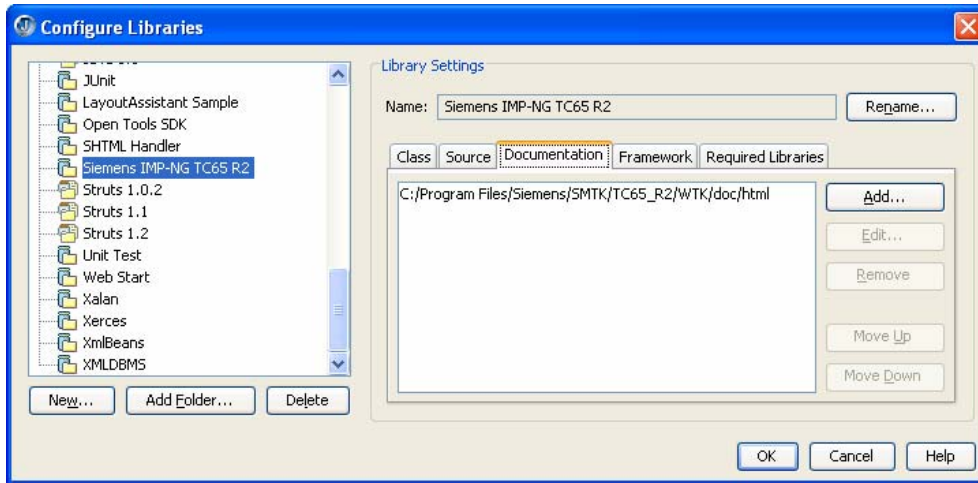


Figure 51: JBuilder 2006 Enterprise/Developer – Siemens Library

### 11.3.4.1 Examples

There are sample projects provided with the SMTK. These projects can be found in the
JBuilderSamples directory of the SMTK installation directory. This directory is accessed by
opening a project using the menu *File → Open Project…* (see Figure below)
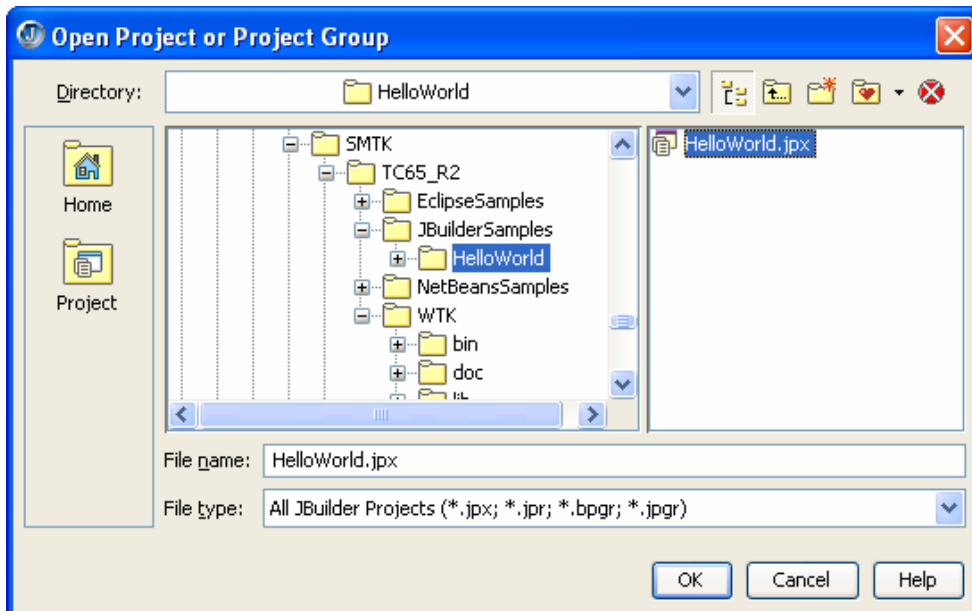


Figure 52: JBuilder 2006 Enterprise/Developer – Sample Projects

Starting the debugging session is done in the same way as for JBuilder X (see chapter for
JBuilder X above)

## 11.4 Breakpoints

Breakpoints can be set as usual within the IDE. The debugger cannot step through methods or functions whose source code is not available.

## 11.5 Switching Java "System.out" to IDE debug window

The Java "System.out" can be redirected during a debugging session using a UDP socket connection as described in chapter "5.9 System Out" and "5.9.3 UDP". Switching between Java "System.out" using serial port and Java "System.out" using UDP socket is done by a setting inside of the emulator configuration file "WM_Debug_config.ini" (e.g. TC65 Release 2: located under "TC65_R2\WTK\bin" ). Switching Java "System.out" to serial port is used as default setting.
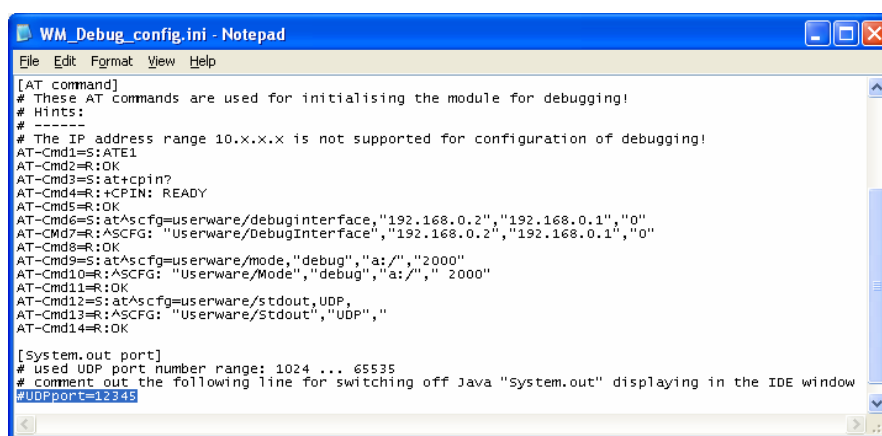


Figure 53: Emulator configuration file "switching Java System.out to serial port"

The following line of the "WM_Debug_config.ini" file is used for switching the Java "System.out" direction:

    #UDPport=12345

To switch the Java "System.out" direction to the UDP socket and displaying it on a IDE window please remove "#". The configuration line looks as follows:

    UDPport=12345

To switch the Java "System.out" direction to serial port output please add "#" (default setting). The configuration line looks as follows:

    #UDPport=12345

Remarks:
Please keep in mind that the handling of Java "System.out" is done asynchronous. It is possible, that not all Java "System.out" data are written into the IDE window. To avoid this problem please set a breakpoint in front of the Java function call "notifyDestroyed()" inside your Java source.

Java "System.out" using IDE window is only supported during debugging session.

# 12      Java Security

The Java Security Model follows the specification of MIDP 2.0 and is IMP-NG conforming. It integrates only a simple protection domain concept since protection domains are not needed for module use cases.

Java Security is divided into two main areas:
- secure MIDlet data links (HTTPS, Secure Connection) (see chapter 12.1)
- execution of signed/unsigned MIDlets (see chapter 12.2)

The interface of Java Security offers the following functionality.
- insert/delete X.509 certificate (default is no certificate, see 12.2.1)
- switch between trusted and untrusted mode for the execution of MIDlet (default is trusted after inserting the certificate, see 12.2.1)
- enable/disable untrusted domain in trusted mode (default is disabled)
- switch MES (default is ON see 12.3)
- switch https certificate verification (default is OFF, see 12.1)

## 12.1    Secure Data Transfer

This feature makes it possible for MIDlets to use safe data links to external communications partners. The specification IMP-NG defines two java classes with this characteristic - HTTPSConnection and SecureConnection.

The Siemens implementation follows the recommendations in IMP-NG:

HTTPSConnection
- HTTP over TLS as documented in <u>RFC 2818</u> and TLS Protocol Version 1.0 as specified in <u>RFC 2246</u>.

SecureConnection
- TLS Protocol Version 1.0 as specified in <u>RFC 2246</u>

Two modes exist for safe data links.
*Mode 1:*
- No examination of the server certificate takes place when setting up the connection.  The authenticity of the server certificate is not verified. (Figure 54: Mode 1 – Customer Root Certificate does not exist)

*Mode 2 (see* 12.2.1 Change to Secured Mode*, 1. Step):*
- Customer Root Certificate is inside of the module
- Command: "Switch on Certificate Verification for HTTPS Connections" was sent
- The server certificate is examined when setting up a connection. Two configurations are valid. The server certificate is identical to the certificate in the module (both certificates are self signed root certificates) or the server certificate forms a chain with the certificate of the module. Thus the authenticity of the server certificate can be examined with the help of the certificate of the module. (Figure 55: Mode 2 - Server Certificate and Certificate into module are identical and Figure 56: Mode 2 - Server Certificate and self signed root Certificate in module form a chain)
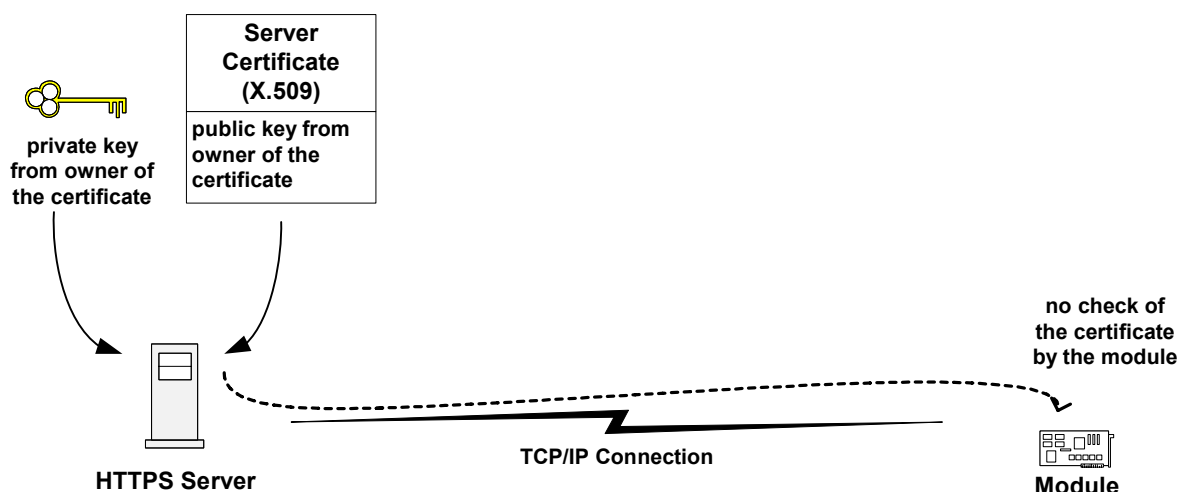


Figure 54: Mode 1 – Customer Root Certificate does not exist
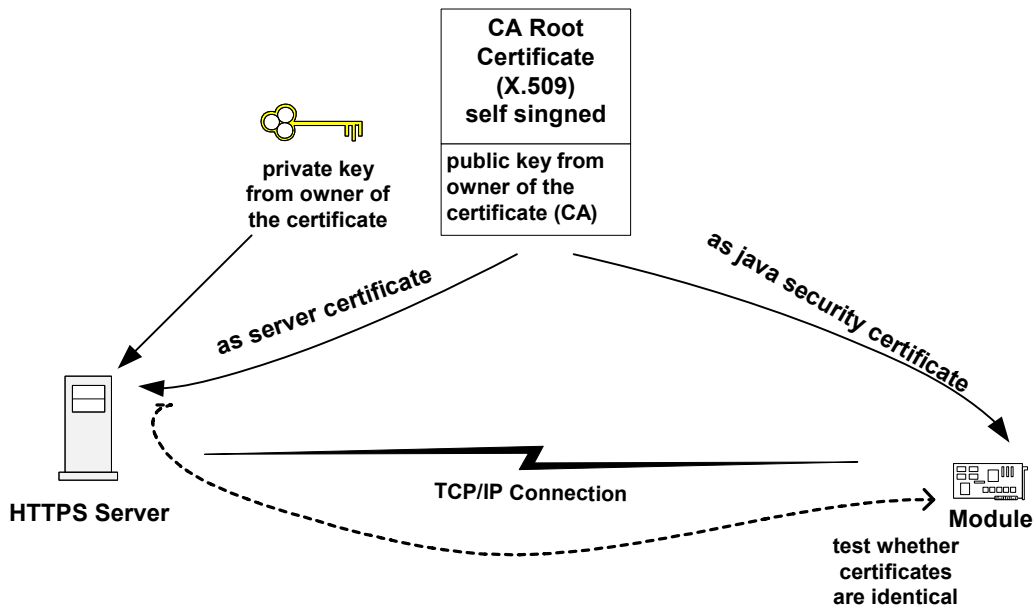
Figure 55: Mode 2 - Server Certificate and Certificate into module are identical
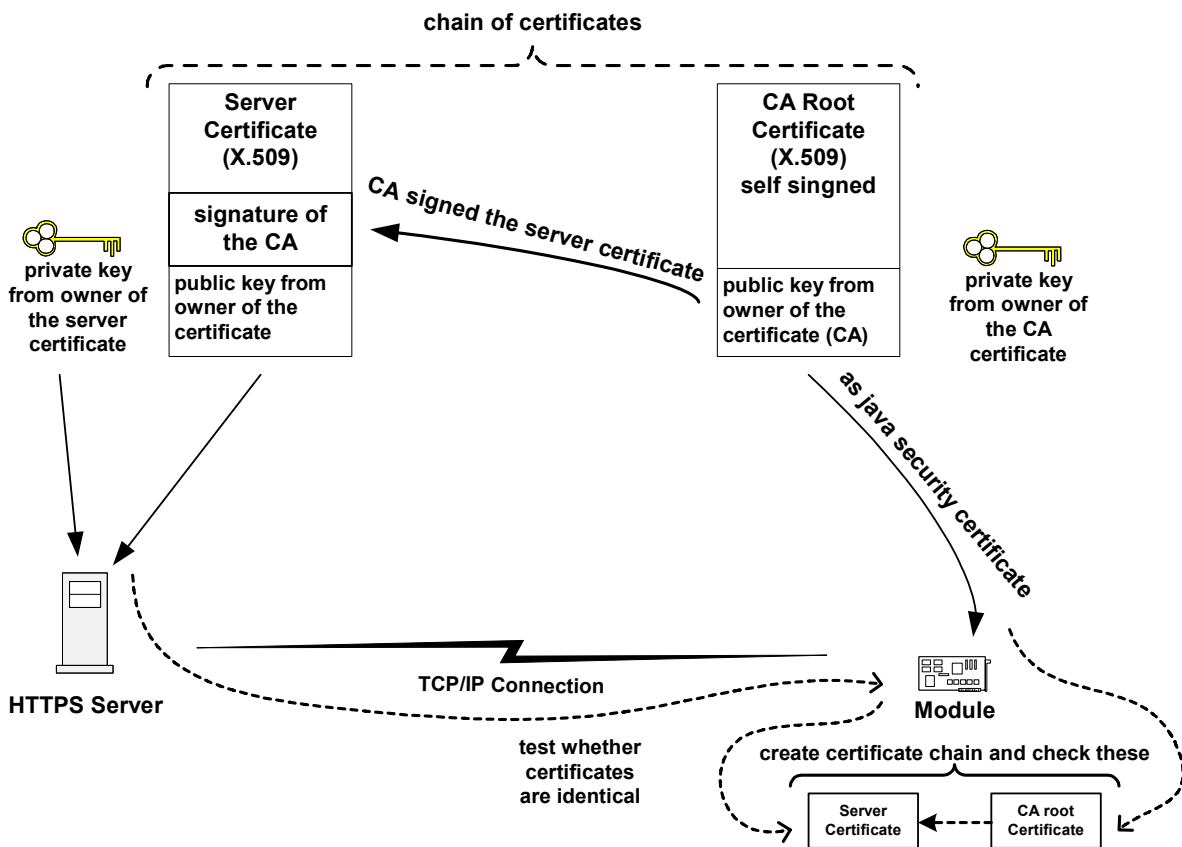


Figure 56: Mode 2 - Server Certificate and self signed root Certificate in module form a chain

## 12.1.1 Create a Secure Data Transfer Environment Step by Step

The following steps describe the creation of the configuration:
- The certificate exists within the module (*see* 12.2.1 Change to Secured Mode, Step 1)
- Certificate verification is activated for a data connection (HTTPS or SecureConnection)

The steps described below use the cygwin + openssl environment (for installation see http://www.cygwin.com/, the openssl documentation can be found here http://www.openssl.org/docs/apps/openssl.html)

A CA Root certificate is generated. This certificate can be placed on the HTTPS-Server. Another possibility is to use the private key of the certificate in order to sign thereby a server certificate. Both certificates form then a chain, which can be examined by the ME.

1. *Create CA and generate CA Root Certificate*
   - We need certificates with sha1 signature. Java Security supports a sha1 signature of the certificate only.
     Add the parameter "-sha1" to the command "Making CA certificate ..." in the section of file CA.pl  (cygwin location *"\cygwin\usr\ssl\misc"*)
   - create a shell (use location *\cygwin\usr\ssl\misc*)
   - execute commands
     ```
     >perl CA.pl –newca
     ```
   - convert file format from PEM to DER
     CA certificate cacert.pem
     ```
     >openssl x509 -in ./demoCA/cacert.pem -inform PEM
                 -out ./demoCA/cacert.der -outform DER
     ```
     CA private key file  cakey.pem
     ```
     >openssl pkcs8 -in ./demoCA/private/cakey.pem
                   -inform PEM
                   -out ./demoCA/private/cakey.der
                   -outform DER -nocrypt -topk8
     ```

2. *Create server certificate and java keystore*
   - execute command
     ```
     >keytool -genkey –alias server -keypass keypass
             -keystore customer.ks -storepass keystorepass
             -sigalg SHA1withRSA -keyalg RSA
     ```
     the field "name" of the certificate is the domain name or the IP address of the server

3. *Create certificate request for server certificate*
   - execute command
     ```
     >keytool -certreq -alias server -file server.csr
             -keypass keypass
             -keystore customer.ks -storepass keystorepass
     ```

4. *Sign certificate request by CA*
   - execute command
     ```
     >openssl ca -in server.csr -out server.pem
     ```
   - convert file format from PEM to DER
     ```
     >openssl x509 -in server.pem -inform PEM
                 –out server.der -outform DER
     ```

5.  *Import CA root certificate and CA private key into java  keystore*
    -   Use the CA Root Certificate for the creation of Java Security Command (see chapter 12.5.3)
    -   execute command
        ```
        >java -jar setprivatekey.jar -alias dummyca
                -storepass keystorepass -keystore customer.ks
                -keypass cakeypass
                -keyfile ./democa/private/cakey.der
                -certfile ./democa/cacert.der
        ```

6.  *Export private key from server certificate*
    -   The private key is needed for the (HTTPS or Secure Connection)server configuration.
    -   execute command
        ```
        >java -jar getprivatekey.jar -alias server
                -keystore customer.ks -storepass keystorepass
                -keypass keypass -keyfile server_privkey.der
        ```

7.  *Send CA Root Certificate to the module*
    -   Create Java Security Command "**SetRootCert**",  see chapter 12.5.3
    -   Send this command with AT^SJSEC (12.4.3) to the module

8.  *Send Java Security Command "Switch on Certificate Verification for HTTPS Connections" to the module*
    -

*Result:*
    -   You have a keystore for the configuration of the Java Security of the module
    -   You have a signed server certificate (files "server.pem" or "server.der")
    -   You have a private key file for your server configuration
    -   the module contains the CA Root Certificate
    -   HTTPS certificate verification is activated

## 12.2 Execution Control

The Java environment of the ME supports two modes.

**Unsecured mode:**
- The device starts all Java applications (MIDlets).

**Secured mode**
- A condition for the secured mode of the device is the existence of a certificate inside of the module.
- The customer can activate the secured mode of the device. To do so, the customer sends a root certificate (x.509 certificate) and the command "Switch ON Security Mode" to the device (over an AT Interface). The device changes from "unsecured mode" to the "secured mode". From this time the module will only start Java applications with a valid signature. In addition, the device will only accept special commands from the customer if they are marked with a signature. The device examines each command with the public key of the customer root certificate.
- The secured mode supports a simple protection domain concept, providing a domain for unsigned MIDlets. If this domain (domain for untrusted MIDlets) is active, then an unsigned MIDlet is assigned to this domain and has only limited access to the Java-API. The untrusted domain is activated by use of Java Security Command "Switch ON Untrusted Domain" (see chapter 12.5.3).
    *untrusted domain:*
    - HTTP-Connection is permitted
    - TCP/IP-Socket Connection is permitted

**Standard behavior of the module**
Siemens supplies modules with unsecured mode as the default configuration.
Insert the certificate:
    - The module changes into the mode "trusted" for MIDlet execution. "Untrusted Domain" is OFF.
    - HTTPS certificate verification is OFF.
    - MES is ON.
Remove the certificate:
    - The module changes into the mode "untrusted" for MIDlet execution.
    - HTTPS certificate verification is OFF
    - MES is ON.

## 12.2.1    Change to Secured Mode Concept

Create and insert a customer root certificate:
A condition for the change to the secured mode is the existence of a customer root
certificate inside of the module



**generate key pair**

**private key**          **public key**

**generate customer root
certificate**

Customer Root Certificate
(X.509)

public key from issuer of
the certificate

Customer Environment

**binary  data of certificate**

**build binary data for AT
command AT^SJSEC**

| AT^SJSEC | cmd len | cmd id | O | O | O | binary data of certificate | signature |

Computer          send to module          Module
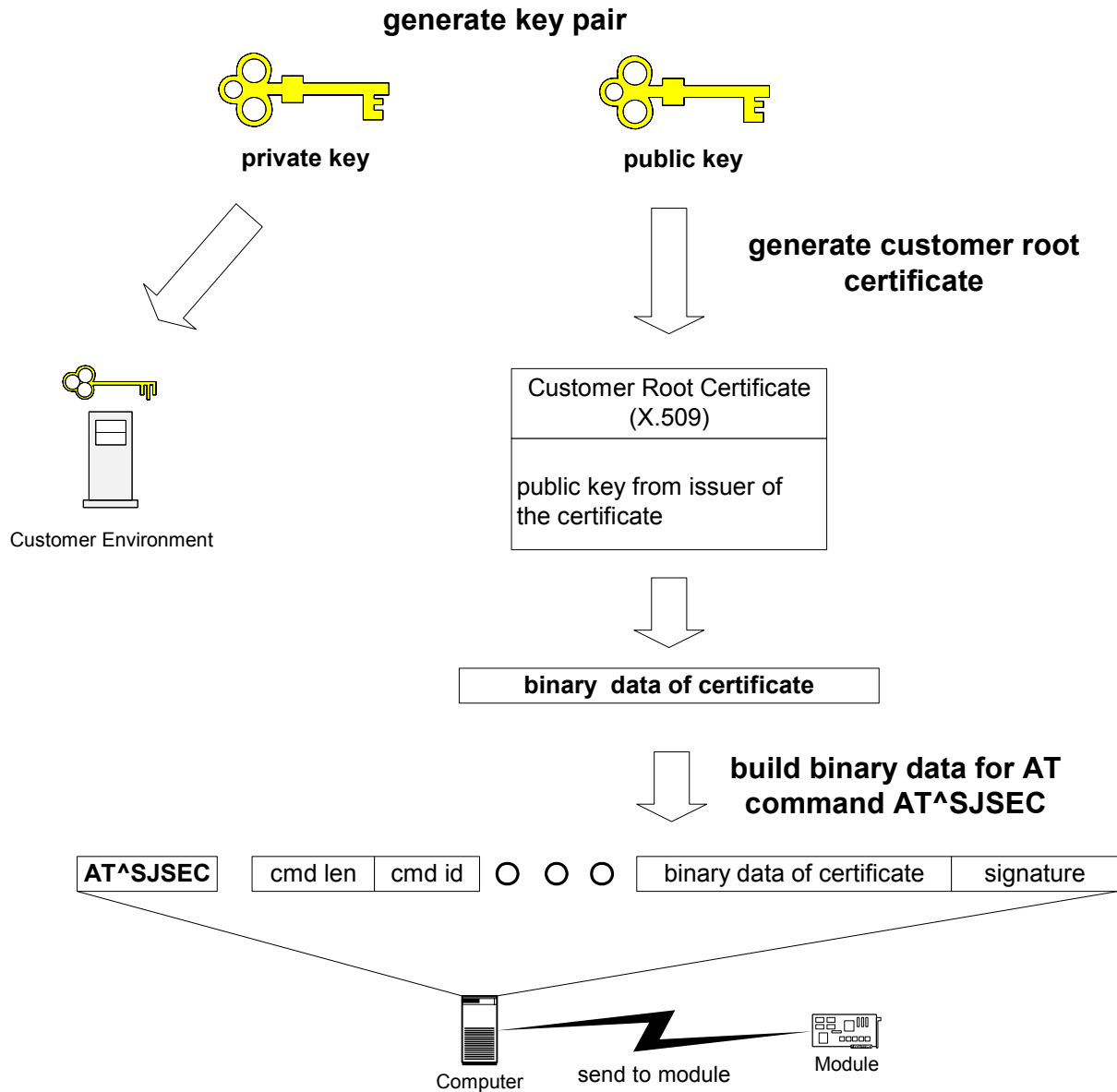
Figure 57: Insert Customer Root Certificate

After this action the module is in the following conditions
- The module changes into the mode "trusted" for MIDlet execution. "Untrusted
  Domain" is OFF.
- HTTPS certificate verification is OFF.
- MES is ON.

## 12.2.2   Concept for the Signing the Java MIDlet

Java-MIDlet
JAR-File

generate HASH

**JAR-File HASH (SHA1)**

encoding with private key

**private key**

Customer Environment

**encoded JAR-File HASH**

encoding base64

add the signature (is base64
encoded) into the JAD file as the
value of the MIDlet-Jar-RSA-SHA1
attribute

Java-MIDlet
JAD-File
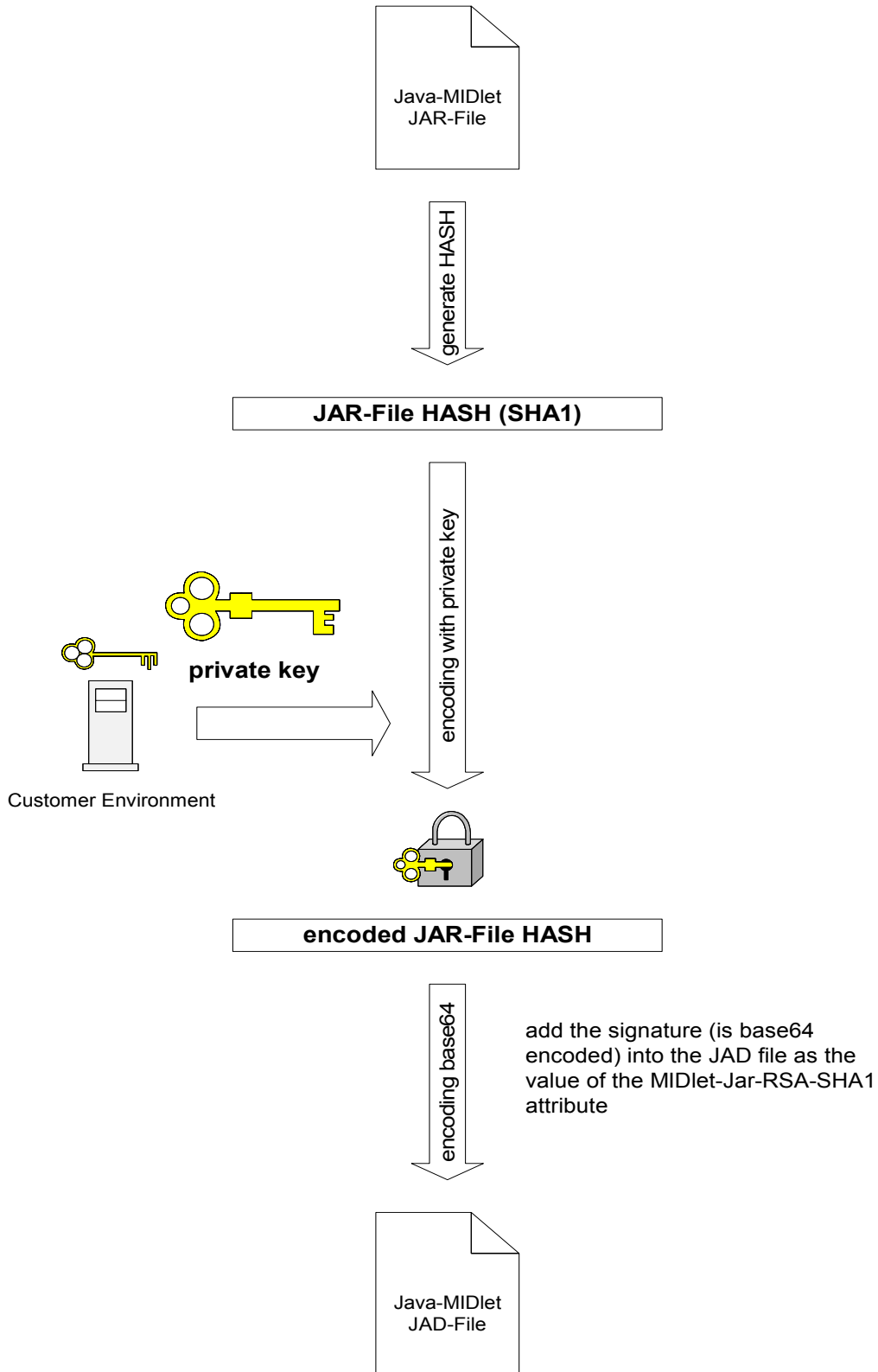
Figure 58: Prepare MIDlet for Secured Mode

## 12.3    Application and Data Protection

In addition to the Java secured mode it is possible to prevent the activation of the Module Exchange Suite. When Module Exchange Suite access is deactivated with *at^sjsec*, it is no longer possible to access to the Flash file system on the module. A condition for the deactivation of the access to the Flash file system is the existence of a customer root certificate inside of the module (see 12.2.1).
The default state of MES is ON.

## 12.4    Structure and Description of the Java Security Commands

Special commands are used in the Java security environment. These commands are transferred to the module with the help of the special AT command *at^sjec*.  This command makes it possible to send binary data to the module.  After *at^sjsec* is issued, the module changes into a block transfer mode. Now binary data in a fixed format can be sent. These binary data contain the actual Java security commands.

## 12.4.1 Structure of the Java Security Commands

**General structure**

| total len | cmd | param num | parameter unit | … | signature unit |
|---|---|---|---|---|---|

| UINT16 | UINT16 | UINT16 | parameter unit | ○ ○ ○ | UINT16 | UINT16 | signature |
|---|---|---|---|---|---|---|---|

| param len | param id | param data |
|---|---|---|

| UINT16 | UINT16 | … |
|---|---|---|

part of the cmd structure for the creation of the signature

total len = all bytes of the command structure (including size of "total len")

param len = all bytes of the parameter structure (including size of "param len")

**List of parameters**

| param id | param len | param data | description |
|---|---|---|---|
| 0x0001 | D | certificate data | content of *.der file |
| 0x0002 | 0x0005 | 0x00 or 0x01 | on/off switch, 0x00 = off, 0x01 = on |
| 0x0003 | 0x0014 | IMEI | numeric numbers in ASCII format (zero terminated string) |
| 0x0004 | D | signature data | SHA-1signature the of command, base64 coded (zero terminated string) |

D - depend from the length of parameter

**List of commands**

| cmd id | description |
|---|---|
| 0x0001 | Set Customer Root Certificate |
| 0x0002 | Del Customer Root Certificate |
| 0x0003 | Switch on/off Certificate Verification for HTTPS Connections |
| 0x0004 | Switch on/off OBEX Functionality |
| 0x0005 | Switch on/off Security Mode of the module |
| 0x0006 | Switch on/off Untrusted Domain inside of the Security Mode |

**Set Customer Root Certificate**

| total len | 0x0001 | 0x0003 | param unit certificate | param unit IMEI | param unit signature |
|---|---|---|---|---|---|

**Del Customer Root Certificate**

| total len | 0x0002 | 0x0002 | param unit IMEI | param unit signature |
|---|---|---|---|---|

**Switch on/off Certificate Verification for HTTPS Connections**

| total len | 0x0003 | 0x0003 | param unit switch | param unit IMEI | param unit signature |
|---|---|---|---|---|---|

**Switch on/off OBEX Functionality**

| total len | 0x0004 | 0x0003 | param unit switch | param unit IMEI | param unit signature |
|---|---|---|---|---|---|

**Switch on/off Security Mode**

| total len | 0x0005 | 0x0003 | param unit switch | param unit IMEI | param unit signature |
|---|---|---|---|---|---|

**Switch on/off Untrusted Domain**

| total len | 0x0006 | 0x0003 | param unit switch | param unit IMEI | param unit signature |
|---|---|---|---|---|---|

## 12.4.2    Build Java Security Command



Figure 59: Build Java Security Command

## 12.4.3    Send Java Security Command to the Module

Use a terminal program.
enter:

```
at^sjsec
```

wait for the answer:

```
CONNECT

JSEC READY: SEND COMMAND ...
```

Now you can send the binary data of the command (for example: from a file with the binary data of the command).
The module's answer depends on the result of the operation.

The read command, *at^sjsec?*, can be used to request the current Java security status.

| Read command AT^SJSEC? | The read command can be used to request the current status of java security. |
|---|---|
| | Response |
| | ^SJSEC:<state>,<HTTPS state>,<OBEX state>,<untrusted domain> [<certificate content>] |
| | |
| | <state>    Java security mode |
| |     0    Unsecured mode, i.e. security mode not active (default) |
| |     1    Secured mode, i.e. security mode active |
| | <HTTPS state> |
| |     0    The HTTPS connection or Secure Connection is possible if the server certificate (or certificate chain) is valid (default) |
| |     1    The HTTPS Connection or Secure Connection is possible only if the server certificate is signed by the customer (owner of root certificate in device) |
| | <OBEX state> |
| |     0    Start of Module Exchange Suite is not permitted |
| |     1    Start of Module Exchange Suite is permitted (default) |
| | <untrusted domain> |
| |     0    Untrusted domain does not exist, MIDlets must be signed (default) |
| |     1    Untrusted domain does exist, unsigned MIDlets have limited rights of access to the Java-API |
| | |
| | Certificate Information: (default no certificate) |
| |     Issuer: |
| |     SerialNumber: |
| |     Subject: |
| |     Signature algorithm: |
| |     Thumbprint algoritm: |
| |     Thumbprint: |

## 12.5 Create a Java Security Environment Step by Step

### 12.5.1 Create Key store

The key store contains the key pairs for signing data. For producing the key store with keys the tool "keytool.exe" can be used.

The program is in the Java SDK. (for a description see
http://java.sun.com/j2se/1.3/docs/tooldocs/solaris/keytool.html)

Example:
```
keytool -genkey -alias keyname -keypass keypassword -keystore customer.ks
        -storepass keystorepassword -sigalg SHA1withRSA -keyalg RSA
```

### 12.5.2 Export X.509 Root Certificate

For exporting the x.509 root certificate use "keytool.exe". The program is in the Java SDK. (for description see http://java.sun.com/j2se/1.3/docs/tooldocs/solaris/keytool.html)

```
keytool -export -v -keystore customer.ks -storepass keystorepassword
        -alias keyname > certificate.der
```

### 12.5.3 Create Java Security Commands

For producing the java security commands the tool "jseccmd.jar" can be used.
This program is in the folder "wkt\bin".

Command for inserting the Customer Root Certificate
- In consequence of the command
  - java security mode             ON (untrusted domain OFF)
  - HTTPS certificate verification     OFF
  - MES state                        ON

```
java -jar jseccmd.jar -cmd SetRootCert -certfile customer.der
                 -imei 012345678901234 -alias keyname
                 -storepass keystorepassword -keypass keypassword
                 -keystore customer.ks > SetRootCert.bin
```

Command for removing the Customer Root Certificate
- In consequence of the command all switches are reset
  - java security mode             OFF (untrusted domain OFF)
  - HTTPS certificate verification     OFF
  - MES state                        ON

```
java -jar jseccmd.jar -cmd DelRootCert
                 -imei 012345678901234 -alias keyname
                 -storepass keystorepassword -keypass keypassword
                 -keystore customer.ks > DelRootCert.bin
```

Command: switch on Java Security Mode

```
java -jar jseccmd.jar -cmd TrustedModeOn
                –imei 012345678901234 -alias keyname
                -storepass keystorepassword -keypass keypassword
                -keystore customer.ks > TrustedModeOn.bin
```

Command: switch off Java Security Mode

```
java -jar jseccmd.jar -cmd TrustedModeOff
                –imei 012345678901234 -alias keyname
                -storepass keystorepassword -keypass keypassword
                -keystore customer.ks > TrustedModeOff.bin
```

Command: switch on Untrusted Domain (it is possible only inside of the Java Security Mode)

```
java -jar jseccmd.jar -cmd UntrustedDomainOn
                –imei 012345678901234 -alias keyname
                -storepass keystorepassword -keypass keypassword
                -keystore customer.ks > UntrustedDomainOn.bin
```

Command: switch off Untrusted Domain

```
java -jar jseccmd.jar -cmd UntrustedDomainOff
                –imei 012345678901234 -alias keyname
                -storepass keystorepassword -keypass keypassword
                -keystore customer.ks > UntrustedDomainOff.bin
```

Command: switch on Certificate Verification for HTTPS Connections

```
java -jar jseccmd.jar -cmd HttpsVerifyOn
                –imei 012345678901234 -alias keyname
                -storepass keystorepassword -keypass keypassword
                -keystore customer.ks > HttpsVerifyOn.bin
```

Command: switch off Certificate Verification for HTTPS Connections

```
java -jar jseccmd.jar -cmd HttpsVerifyOff
                –imei 012345678901234 -alias keyname
                -storepass keystorepassword -keypass keypassword
                -keystore customer.ks > HttpsVerifyOff.bin
```

Command: switch on module exchange functionality

```
java -jar jseccmd.jar -cmd ObexActivationOn
                      -imei 012345678901234 -alias keyname
                      -storepass keystorepassword -keypass keypassword
                      -keystore customer.ks > ObexActivationOn.bin
```

Command: switch off module exchange functionality

```
java -jar jseccmd.jar -cmd ObexActivationOff
                      -imei 012345678901234 -alias keyname
                      -storepass keystorepassword -keypass keypassword
                      -keystore customer.ks > ObexActivationOff.bin
```

## 12.5.4   Sign a MIDlet

Use the tool "jadtool.jar"  to sign a Java MIDlet.  This program is in the folder "wkt\bin".

```
java -jar jadtool.jar -addjarsig -jarfile helloworld.jar
                                 -inputjad helloworld.jad
                                 -outputjad helloworld.jad
                                 -alias keyname -storepass keystorepassword
                                 -keypass keypassword -keystore customer.ks
                                 -encoding UTF-8
```

## 12.6     Attention

The central element of Java Security is the **private key**. If Java Security is activated and you lose the private key, then the **module is destroyed**. You have no chance of deactivating Java Security, downloading of a new Midlet or starting any other operation concerning Java Security.
To prevent problems you are strongly advised to **secure the private key**.

# 13  Java Tutorial

This small tutorial includes explanations on how to use the AT Command API and suggestions for programming MIDlets. The developer should read about MIDlets, Threads and AT commands as a complement to this tutorial.

## 13.1  Using the AT Command API

Perhaps the most important API for the developer is the AT command API. This is the API that lets the developer issue commands to control the module. This API consists of the *ATCommand* class and the *ATCommandListener* and *ATCommandResponseListener* interfaces. Their javadocs can be found in *…\wtk\doc\html\index.html*, [5].

## 13.1.1  Class ATCommand

The *ATCommand* class supports the execution of AT commands in much the same way as they would be executed over a serial interface. It provides a simple way to send strings directly to the device's AT parsers.

### 13.1.1.1  Instantiation with or without CSD Support

There can be only exactly as many *ATCommand* instances as there are parsers on the device. If there are no more parsers available, the *ATCommand* constructor will throw *ATCommandFailedException*. All AT parser instances support CSD. However from a Java application point of view it may make sense to have one dedicated instance for CSD call handling. Therefore, and also for historical reasons, only one parser with CSD support may be requested through the constructor. If more then one parser with CSD support is requested, the constructor will throw *ATCommandFailedException*.

```
try {
  ATCommand atc = new ATCommand(false);
  /* An instance of ATCommand is created. CSD is not explicitly
   * requested. */
} catch (ATCommandFailedException e) {
  System.out.println(e);
}
```

The csdSupported() method returns the CSD capability of the connected instance of the device's AT parser. The method checks as well, if the current mode of the module supports CSD. Please notice that this check has not been done when opening the *ATCommand* instance.

```
boolean csd_support = atc.csdSupported();
```

*release()* releases the resources held by the instance of the ATCommand class. After calling this function the class instance cannot be used any more but the resources are free to be used by a new instance

### 13.1.1.2   Sending an AT Command to the Device, the send() Method

An AT command is sent to the device by using the send() method. The AT command is sent as a string which must include the finalizing line feed "\r" or the corresponding line end character.

```
String response = atc.send("at+cpin?\r");
/* method returns when the module returns a response */
System.out.println(response);
```

Possible response printed to System.out:
```
+CPIN: READY OK
```

This send function is a blocking call, which means that the calling thread will be interrupted until the module returns a response. The function returns the response, the result code of the AT command, as a string.

Occasionally it may be infeasible to wait for an AT command that requires some time to be processed, such as at+cops. There is a second, non-blocking, send function which takes a second parameter in addition to the AT command. This second parameter is a callback instance, *ATCommandResponseListener*.  Any response to the AT command is delivered to the callback instance when it becomes available. The method itself returns immediately. The *ATCommandResponseListener* and the non-blocking send method are described in Section 13.1.2.

Note: Using the send methods with strings with incorrect AT command syntax will cause errors.

### 13.1.1.3   Data Connections

If a data connection is created with the *ATCommand* class, for instance with '*atd*', an input stream is opened to receive the data from the connection. Similarly, an output stream can be opened to send data on the connection.

```
/* Please note that this example would not work unless the module had
 * been initialized and logged into a network. */

System.out.println("Dialing: ATD" + CALLED_NO);
response = atc.send("ATD" + CALLED_NO + "\r");
System.out.println("received: " + response);

if (response.indexOf("CONNECT") >= 0) {
  try {
    // We have a data connection, now we do some streaming...
    // IOException will be thrown if any of the Stream methods fail
    OutputStream dataOut = ATCmd.getDataOutputStream();
    InputStream dataIn = ATCmd.getDataInputStream();

    // out streaming...
    dataOut.write(new String("\n\rHello world\n\r").getBytes());
    dataOut.write(new String("\n\rThis data was sent by a Java " +
                        "MIDlet!\n\r").getBytes());
    dataOut.write(new String("Press 'Q' to close the " +
                        "connection\n\r").getBytes());

    // ...and in streaming
    System.out.println("Waiting for incoming data, currently " +
                    dataIn.available() + " bytes in buffer.");
    rcv = 0;
```

```
    while(((char)rcv != 'q') && ((char)rcv != 'Q') && (rcv != -1)){
      rcv = dataIn.read();
      if (rcv >= 0) {
        System.out.print((char)rcv);
      }
    }

    /* The example continues after the next block of text */
```

In **…/Siemens/SMTK/<productname>/wtk/src/example** (e.g. TC65 Release 2 …/Siemens/SMTK/TC65_R2/wtk/src/example) a complete data connection example, DataConnectionDemo.java, can be found.

These streams behave slightly differently than regular data streams. The streams are not closed by using the close() method. A stream remains open until the *release()* method is called. A module can be switched from the data mode to the AT command mode by calling the *breakConnection()* method.

```
    /* continue example */

    if (rcv != -1) {
      // Now break the data connection
      System.out.println("\n\n\rBreaking connection");
      try {
        strRcv = ATCmd.breakConnection();
      } catch(Exception e) {
        System.out.println(e);
      }
      System.out.println("received: " + strRcv);
    } else {
      // Received EOF, somebody else broke the connection
      System.out.println("\n\n\rSomebody else switched to " +
                          "command mode!");
    }
    System.out.println("Hanging up");
    strRcv = ATCmd.send("ATH\r");
    System.out.println("received: " + strRcv);
  } catch(IOException e) {
    System.out.println(e);
  }
} else {
  System.out.println("No data connection established,");
}
```

An IOException is thrown if any function of the I/O streams are called when the module is in AT command mode, except for read functions. The read function case is explained in more detail in chapter 13.1.3.

The ATCommand class does not report the result codes returned after data connection release.

Data Connections are not only used for data transfer over the air but also to access external hardware. Here is a list of at commands which open a data connection:
• atd, for data calls
• at^sspi, for access to I2C/SPI
• at^sis commands for Internet services

For data connection signaling see also chapter 13.1.3

### 13.1.1.4  Synchronization

For performance reasons no synchronization is done in the ATCommand class. If an instance of this class has to be accessed from different threads ensure that the *send()* functions, the *release()* function, the *cancelCommand()* function and the *breakConnection()* function are synchronized in the user implementation.

## 13.1.2   ATCommandResponseListener Interface

The *ATCommandResponseListener* interface defines the capabilities for receiving the response to an AT command sent to one of the module's AT parsers. When the user wants to use the non blocking version of the *ATCommand.send* function of an implementation class of the *ATCommandResponseListener* interface must be created first. The single method of this class, *ATResponse()*, must contain the processing code for the possible response to the AT command sent.

```
class MyListener implements ATCommandResponseListener {

  String listen_for;

  public MyListener(String awaited_response) {
    listen_for = awaited_response;
  }

  void ATResponse(String Response) {
    if (Response.indexOf(listen_for) >= 0) {
      System.out.println("received: " + strRcv);
    }
  }
}
```

### 13.1.2.1   Non-blocking ATCommand.send() Method

After creating an instance of the *ATCommandResponseListener* class, the class instance can be passed as the second parameter of the non-blocking ATCommand.send() method. After the AT command has been passed to the AT parser, the function returns immediately and the response to the AT command is passed to this callback class later when it becomes available

Somewhere in the application:

```
MyListener connect_list = new MyListener("CONNECT");
atc.send("ATD" + CALLED_NO + "\r", connect_list);

/* Application continues while the AT command is processed*/
/* When the module delivers its response to the AT command the callback
 * method ATResponse is called. If the response is "CONNECT", we will see
 * the printed message from ATResponse in MyListener. */
```

A running AT command sent with the non-blocking send function can be cancelled with *ATCommand.cancelCommand()*. Any possible responses to the cancellation are sent to the waiting callback instance.

Note: Using the send methods with incorrect AT command syntax in the strings will cause errors.

## 13.1.3 ATCommandListener Interface

The *ATCommandListener* interface implements callback functions for:

- URCs
- Changes of the serial interface signals RING, DCD and DSR
- Opening and closing of data connections

The user must create an implementation class for *ATCommandListener* to receive AT events. The *ATEvent* method of this class must contain the processing code for the different AT-Events (URCs). The *RINGChanged, DCDChanged, DSRChanged* and *CONNChanged* methods should contain the processing code for possible state changes. This code shall leave the listener context as soon as possible (i.e. running a new thread). While the callback method does not return it cannot be called again if changes occur.<br>

The *CONNChanged* method indicates the start and the end of data connections. During a data connection it is possible to transfer data with the I/O stream methods (see chapter 12.1.1.3). Some data services (i.e. FTP) transfer the data so quickly, that the *CONNChanged* start and close events are received even parallel to the response of the AT command which originated the data connection. The user's application must realize, that the data connection had taken place and read the data with the I/O stream read methods afterwards.

### 13.1.3.1 ATEvents

An ATEvent or a URC is a report message sent from the module to the application. An unsolicited result code is either delivered automatically when an event occurs or as a result of a query the module previously received. However, a URC is not issued as a *direct* response to an executed AT command. Some URCs must be activated with an AT command.

Typical URCs may be information about incoming calls, a received SM, temperature changes, the status of the battery, etc. A summary of URCs is listed in the AT Command Set document [3].

### 13.1.3.2 Implementation

```
class ATListenerA implements ATCommandListener {

public void ATEvent(String Event) {
  if (Event.indexOf("+CALA: Reminder 1") >= 0) {
    /* take desired action after receiving the reminder */
  } else if (Event.indexOf("+CALA: Reminder 2") >= 0) {
    /* take desired action after receiving the reminder */
  } else if (Event.indexOf("+CALA: Reminder 3") >= 0) {
    /* take desired action after receiving the reminder */
  }

  /* No action taken for these events */
  public void RINGChanged(boolean SignalState) {}
  public void DCDChanged(boolean SignalState) {}
  public void DSRChanged(boolean SignalState) {}
}

class ATListenerB implements ATCommandListener {

  public void ATEvent(String Event) {
    if (Event.indexOf("+SCKS: 0") >= 0) {
      System.out.println("SIM Card is not inserted.");
```

```
        /* perform other actions */
    } else if (Event.indexOf("+SCKS: 1") >= 0) {
      System.out.println("SIM Card is inserted.");
     /* perform other actions */
    }
  }

  public void RINGChanged(boolean SignalState) {
    /* take some action when the RING signal changes if you want to */
  }

  public void DCDChanged(boolean SignalState) {
    /* take some action when the DCD signal changes if you want to */
  }

  public void DSRChanged(boolean SignalState {}
    /* take some action when the DSR signal changes if you want to */
  }

  public void CONNChanged(boolean SignalState {}
    /* take some action when the state of a connection changes if you want
    to */
  }

}
```

### 13.1.3.3  Registering a Listener with an ATCommand Instance

After creating an instance of the *ATCommandListener* class, it must be passed as a parameter to the *ATCommand.addListener()* method. The callback methods of the instance will be called by the runtime system each time the corresponding events (URCs or signal state changes) occur on the corresponding device AT parser.

```
/* we have two ATCommands instances, atc1 and atc2 */
ATListenerA reminder_listener = new ATListenerA();
ATListenerB card_listener = new ATListenerB();

atc1.addListener(reminder_listener);
atc2.addListener(card_listener);
```

The *ATCommand.removeListener()* method removes a listener object that has been previously added to the internal list table of listener objects. After it has been removed from the list it will not be called when URCs occur. If it was not previously registered the list remains unchanged.

The same *ATCommandListener* may be added to several *ATCommand* instances and several *ATCommandListeners* may be added to the same *ATCommand*.

## 13.2    Programming the MIDlet

The life cycle and structure of MIDlets are described in Chapter 6. Since the MIDlets will run on J2ME™, all of J2ME™'s features, including threads, are available. Small applications, such as those without any timer functions or those used only for tests and simple examples, can be written without using threads. Longer applications should be implemented with threads.

### 13.2.1    Threads

Although small applications can be written without using threads longer applications should use them. The Java programming language is naturally multi-threaded which can make a substantial difference in the performance of your application. Therefore we recommend referring to Java descriptions on threads before making any choices about threading models. Threads can be created in two ways. A class can be a subclass of *Thread* or it can implement *Runnable*.

For example, threads can be launched in *startApp()* and destroyed in *destroyApp()*. Note that destroying Java threads can be tricky. It is recommended that the developer read the Java documentation on threads. It may be necessary to poll a variable within the thread to see if it is still alive.

### 13.2.2    Example

```
/* This example derives a class from Thread and creates two instances
 * of the subclass. One thread instance finishes itself, the other one
 * is stopped by the main application. */
```

```
package example.threaddemo;

import javax.microedition.midlet.*;

public class ThreadDemo extends MIDlet {

  /* Member variables */
  boolean    runThreads = true; // Flag for stopping threads
  DemoThread thread1;           // First instance of DemoThread
  DemoThread thread2;           // Second instance of DemoThread

  /* Private class implementing the thread to be started by the
   * main application */
  private class DemoThread extends Thread {
    int loops;

    public DemoThread(int waitTime) {
      /* Store number of loops to execute */
      loops = waitTime;
      System.out.println("Thread(" + loops + "): Created");
    }

    public void run() {
      System.out.println("Thread(" + loops + "): Started");
      for (int i = 1; i <= loops; i++) {
        /* Check if main application asked thread to die */
        if (runThreads != true) {
          System.out.println("Thread(" + loops + "): Stopped from outside");
          /* Leave thread */
          return;
        }
         /* Print loop counter and wait 1 second,
          * do something useful here instead */
```

```java
          System.out.println("Thread(" + loops + "): Loop " + i);
          try {
            Thread.sleep(1000);
          } catch(InterruptedException e) {
            System.out.println(e);
          }
        }
        System.out.println("Thread(" + loops + "): Finished naturally");
      }
  }

  /**
   * ThreadDemo - constructor
   */
  public ThreadDemo() {
    System.out.println("ThreadDemo: Constructor, creating threads");
    thread1 = new DemoThread(2);
    thread2 = new DemoThread(6);
  }

  /**
   * startApp()
   */
  public void startApp() throws MIDletStateChangeException {
    System.out.println("ThreadDemo: startApp, starting threads");
    thread1.start();
    thread2.start();
    System.out.println("ThreadDemo: Waiting 4 seconds before stopping threads");
    try {
      Thread.sleep(4000);
    } catch(InterruptedException e) {
      System.out.println(e);
    }
    destroyApp(true);
    System.out.println("ThreadDemo: Closing application");

  }

  /**
   * pauseApp()
   */
  public void pauseApp() {
    System.out.println("ThreadDemo: pauseApp()");
  }

  /**
   * destroyApp()
   */
  public void destroyApp(boolean cond) {
    System.out.println("ThreadDemo: destroyApp(" + cond + ")");
    System.out.println("ThreadDemo: Stopping threads from outsdide");
    runThreads = false;
    try {
      System.out.println("ThreadDemo: Waiting for threads to die");
      thread1.join();
      thread2.join();
    } catch(InterruptedException e) {
      System.out.println(e);
    }
    System.out.println("ThreadDemo: All threads died");

    notifyDestroyed();
  }
}
```

# 14 Differences to the TC45

For those who are familiar with the Siemens WM IMP1.0 product (TC45) this is an overview of the main differences between this and the IMP-NG product line (e.g. TC65).

- "real" TCP and UDP access interfaces: SocketConnection, ServerSocketConnection, UDPDatagramConnection.
  Use of StreamConnection, StreamConnectionNotifier, DatagramConnection is now discouraged.
- CommConnection can access both RS232 interfaces ASC0 and ASC1
- Standard.out can be directed to any of the serial interfaces ASC0, ASC1, USB or written to the flash file system.
- No IO pin multiplexing: GPIOs, DAI and the serial interface for CommConnection do not share any pins, so the selection mechanism no longer exists.
- The CommConnection interface which used to be proprietary (com.siemens.icm.io) is now part of the standard package (javax.microedition.io).
- No interface emulation on the PC: When running a MIDlet under the emulator, it is completely executed in the connected module and therefore uses the modules "real" interfaces. The emulation of interfaces such as networking, file system or serial interface on the PC side no longer exists.
- .jad files required: A suitable descriptor file is now not only required for OTAP but in all cases. An absent or invalid .jad file causes an error when starting an application (at^sjra or autostart).
- Mandatory attributes: the attributes MicroEdition-Profile and MicroEdition-Configuration are now mandatory attributes in the manifest and .jad file.
- Flexible echo: When using the ATCommand class the "echo" is can be switched on and off as in non-Java mode. The default is echo on. In the TC45 the echo was always off.
- activeCount result: The method activeCount of class Thread returns the number of threads currently present in the VM. In the TC45 only the non sleeping or waiting threads were counted.