Java User’s Guide

Users Guide

Version: 01
DocId: bgs5_java_usersguide_v01

Products: Cinterion® BGS5

Document Name:	**Java User's Guide**
Version:	**01**
Date:	**2013-11-19**
DocId:	**bgs5_java_usersguide_v01**
Status	**Confidential / Released**
Supported Products:	**Cinterion® BGS5**

GENERAL NOTE
THE USE OF THE PRODUCT INCLUDING THE SOFTWARE AND DOCUMENTATION (THE "PRODUCT") IS SUBJECT TO THE RELEASE NOTE PROVIDED TOGETHER WITH PRODUCT. IN ANY EVENT THE PROVISIONS OF THE RELEASE NOTE SHALL PREVAIL. THIS DOCUMENT CONTAINS INFORMATION ON GEMALTO M2M PRODUCTS. THE SPECIFICATIONS IN THIS DOCUMENT ARE SUBJECT TO CHANGE AT GEMALTO M2M'S DISCRETION. GEMALTO M2M GMBH GRANTS A NON-EXCLUSIVE RIGHT TO USE THE PRODUCT. THE RECIPIENT SHALL NOT TRANSFER, COPY, MODIFY, TRANSLATE, REVERSE ENGINEER, CREATE DERIVATIVE WORKS; DISASSEMBLE OR DECOMPILE THE PRODUCT OR OTHERWISE USE THE PRODUCT EXCEPT AS SPECIFICALLY AUTHORIZED. THE PRODUCT AND THIS DOCUMENT ARE PROVIDED ON AN "AS IS" BASIS ONLY AND MAY CONTAIN DEFICIENCIES OR INADEQUACIES. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, GEMALTO M2M GMBH DISCLAIMS ALL WARRANTIES AND LIABILITIES. THE RECIPIENT UNDERTAKES FOR AN UNLIMITED PERIOD OF TIME TO OBSERVE SECRECY REGARDING ANY INFORMATION AND DATA PROVIDED TO HIM IN THE CONTEXT OF THE DELIVERY OF THE PRODUCT. THIS GENERAL NOTE SHALL BE GOVERNED AND CONSTRUED ACCORDING TO GERMAN 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 © 2013, Gemalto M2M GmbH, a Gemalto Company

Trademark Notice
Gemalto, the Gemalto logo, are trademarks and service marks of Gemalto and are registered in certain countries. Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other registered trademarks or trademarks mentioned in this document are property of their respective owners.

Content

1 Preface 10
1.1 Document History 10

2 Overview 11
2.1 Related Documents 11
2.2 Terms and Abbreviations 11

3 Installation 13
3.1 System Requirements 13
3.2 Installation CD Content 13
3.3 Cinterion Mobility Toolkit Installation 14
3.3.1 Installation Prerequisites 14
3.3.2 Installing CMTK 15
3.4 CMTK Uninstall 19
3.5 Upgrades 19

4 Software Platform 20
4.1 Software Architecture 20
4.2 Hardware Interfaces 21
4.2.1 ASC0 - Serial Device 21
4.2.2 ASC1 - Serial Device 21
4.2.3 General Purpose I/O 21
4.2.4 ADC 21
4.2.5 PWM/DAC 21
4.2.6 Digital Audio Interface (DAI) 21
4.2.7 I2C 22
4.2.8 SPI 22
4.2.9 USB 22
4.3 Other Interfaces 22
4.3.1 IP Networking 22
4.3.2 Media 22
4.3.3 Others 22
4.4 Data Flow of a Java Application Running on the Module 23
4.5 Handling Interfaces and Data Service Resources 24
4.5.1 Module States 24
4.5.1.1 State 1: Default – Module after Power-Up 24
4.5.1.2 State 2: Default - System.out is Configured 25
4.5.1.3 State 3: User Specific Java Application Started 25
4.5.1.4 State 4: Typical Deployment 25

5 Maintenance 26
5.1 IP Service 26
5.2 Power Saving 27
5.3 Airplane Mode 28

5.4 Alarm ... 28
5.5 Shutdown ... 29
5.5.1 Automatic Shutdown ... 29
5.5.2 Manual Shutdown ... 29
5.5.3 Restart after Switch Off ... 29
5.5.4 Watchdog ... 29
5.6 Special AT Command Set for Java Applications ... 30
5.6.1 Switching from Data Mode to Command Mode ... 30
5.6.2 Mode Indication after MIDlet Startup ... 30
5.6.3 Long Responses ... 30
5.6.4 Configuration of Serial Interface (ASC0, ASC1) ... 30
5.6.5 Java Commands ... 31
5.7 System Out ... 31
5.7.1 Serial interfaces ... 31
5.7.2 File ... 31
5.8 Restrictions ... 32
5.8.1 Flash File System ... 32
5.8.2 Memory ... 32
5.8.3 JAD File Size ... 32
5.8.4 AT Command API ... 32
5.9 System Time ... 32

6 MIDlets ... 33
6.1 MIDlet Documentation ... 33
6.2 MIDlet Life Cycle ... 33
6.3 Multiple MIDlets ... 34
6.4 Hello World MIDlet ... 35

7 File Transfer to Module ... 36
7.1 Module Exchange Suite ... 36
7.1.1 Windows Based ... 36
7.1.2 Command Line Based ... 36
7.2 Over the Air Provisioning ... 36
7.3 Security Issues ... 37
7.3.1 Module Exchange Suite ... 37
7.3.2 OTAP ... 37

8 Over The Air Provisioning (OTAP) ... 38
8.1 Introduction to OTAP ... 38
8.2 OTAP Overview ... 38
8.3 OTAP Parameters ... 39
8.4 Short Message Format ... 40
8.5 Java File Format ... 41
8.6 Procedures ... 42
8.6.1 Install/Update ... 42
8.6.2 Delete ... 43

8.7 Time Out Values and Result Codes 44
8.8 Tips and Tricks for OTAP 44
8.9 OTAP Tracer 45
8.10 Security 45
8.11 How To 45

9 Compile and Run a Program without a Java IDE 47
9.1 Build Results 47
9.2 Compile 48
9.3 Run on the Module with Manual Start 48
9.4 Run on the Module with Autostart 49
9.4.1 Switch on Autostart 49
9.4.2 Switch off Autostart 49
9.4.3 Autostart Fail-Safe 50

10 Compile and Run a Program with a Java IDE 51
10.1 Debug Environment 51
10.1.1 Data Flow of a Java Application in the Debug Environment 51
10.1.2 Emulator 52
10.1.3 Change Baud Rate 53
10.2 Using Eclipse for Java Development 55
10.2.1 Installing the "Mobile Tools for Java" Plugin 56
10.2.2 Integrating Cinterion WTK Manually 57
10.2.3 Import the provided WTK Samples 61
10.2.4 Creating a new MIDlet 63
10.2.5 Using Eclipse Workspaces 72
10.3 Using NetBeans for Java Development 73
10.3.1 Installing the "Mobility" Plugin 73
10.3.2 Integrating Cinterion WTK Manually 74
10.3.3 Opening the Provided WTK Samples 77
10.3.4 Creating a New MIDlet 78

11 Java Security 81
11.1 Secure Data Transfer 82
11.1.1 HTTPS - Client Authentication 85
11.2 Execution Control 86
11.2.1 Change to Secured Mode Concept 87
11.2.2 Concept for the Signing the Java MIDlet 88
11.3 Application and Data Protection 89
11.4 Structure and Description of the Java Security Commands 89
11.4.1 Structure of the Java Security Commands 89
11.4.2 Build Java Security Commands 92
11.4.3 Send Java Security Command to the Module 93
11.4.3.1 AT^SJMSEC Command Syntax 94

11.5 Create a Java Security Environment Step by Step 97
11.5.1 Create SE Keysstore 97
11.5.2 Create ME Keystore 97
11.5.3 Create Java Security Commands 97
11.5.4 Sign a MIDlet 101
11.6 Attention 101

12 Differences to EGS5/TC65i 102

Tables

Table 1: A typical sequence of MIDlet execution 34
Table 2: Parameters and keywords 39
Table 3: List of commands 90
Table 4: List of parameters 90
Table 5: Command structure 91

Figures

Figure 1: Overview 11
Figure 2: CMTK - InstallShield Wizard 15
Figure 3: Module Exchange Suite - InstallShield Wizard 16
Figure 4: IMP Debug Connection - InstallShield Wizard 17
Figure 5: Device software: Modem driver installation 17
Figure 6: Scan COM ports for available JAVA module 18
Figure 7: Scan for supported JAVA IDEs 18
Figure 8: Query to install Eclipse IDE as part of CMTK installation 19
Figure 9: System architecture 20
Figure 10: Data flow of a Java application running on the module 23
Figure 11: Module State 1 24
Figure 12: Module State 2 25
Figure 13: Module State 3 25
Figure 14: Module State 4 25
Figure 15: OTAP Overview 38
Figure 16: OTAP: Install/Update Information Flow (messages in brackets are optional) 42
Figure 17: OTAP: Delete Information Flow (messages in brackets are optional) 43
Figure 18: Data flow of a Java application in the debug environment 51
Figure 19: Specify maximum port speed for modem device 53
Figure 20: Configure debug connection 54
Figure 21: Specify baud rate for debug connection 54
Figure 22: Installing the "Mobile Tools for Java" plugin 56
Figure 23: Integrating Cinterion WTK manually - Select preference 57
Figure 24: Integrating Cinterion WTK manually - Browse 58
Figure 25: Integrating Cinterion WTK manually - Edit 59
Figure 26: Integrating Cinterion WTK manually - Library 60
Figure 27: Integrating Cinterion WTK manually - Add WTK documentation 60
Figure 28: Import the provided WTK Samples - Select 61
Figure 29: Import the provided WTK Samples - Copy 62
Figure 30: Creating a new MIDlet - Select wizard 63
Figure 31: Creating a new MIDlet - Create project 64
Figure 32: Creating a new MIDlet - Configure project 65
Figure 33: Creating a new MIDlet - Project overview 66
Figure 34: Creating a new MIDlet - Configure compliance level 67
Figure 35: Add cwmlib to project 68
Figure 36: Include cwmlib into project package 69
Figure 37: Creating a new MIDlet - Program name 70
Figure 38: Creating a new MIDlet - HelloWorld.java 71
Figure 39: Using Eclipse workspaces 72
Figure 40: Installing "Mobility" plugin 73
Figure 41: Integrating Cinterion WTK manually - Select platform 74
Figure 42: Integrating Cinterion WTK manually - Select type 75
Figure 43: Integrating Cinterion WTK manually - Platform folder 76
Figure 44: Integrating Cinterion WTK manually - Finish 77
Figure 45: Creating a new MIDlet - Choose project 78
Figure 46: Creating a new MIDlet - Enter name and location 78
Figure 47: Creating a new MIDlet - Configure platform 79
Figure 48: Creating a new MIDlet - Program name 79
Figure 49: Creating a new MIDlet - HelloWorld.java 80

Figure 50: Mode 1 - Customer Root certificate does not exist 83
Figure 51: Mode 2 - Server certificate and certificate into module are identical..... 84
Figure 52: Mode 2 - Server certificate and self signed root certificate in module form a chain 85
Figure 53: Insert customer ME keystore 87
Figure 54: Prepare MIDlet for secured mode..... 88
Figure 55: Structure of Java Security commands 89
Figure 56: Build Java Security command..... 92

1 Preface

This document covers the following IMP-NG Java products from Gemalto M2M:
1. Cinterion® BGS5 Module

Where applicable 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 BGS5. Screenshots are provided as examples and, unless otherwise stated, apply to all supported products.

1.1 Document History

New document "Java User's Guide" Version **01**

Chapter	What is new?
--	Initial document setup.

2 Overview

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.

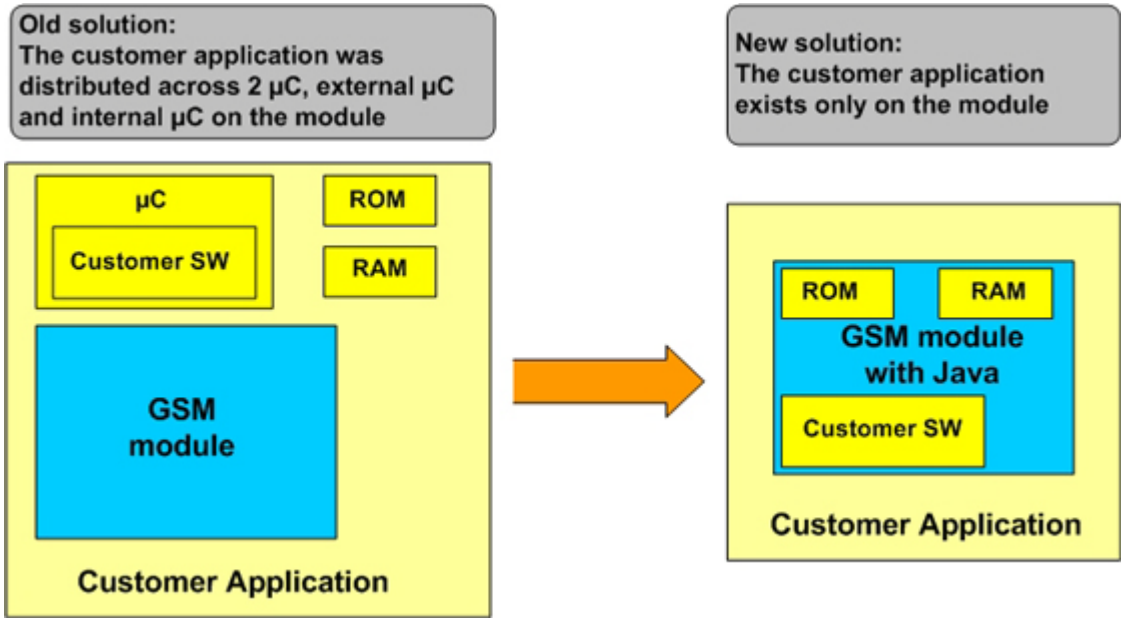

Figure 1: Overview

2.1 Related Documents

List of documents referenced throughout this manual:
[1] AT Command Set of your Gemalto M2M product
[2] Hardware Interface Description of your Gemalto M2M product
[3] Java doc \wtk\doc\html\index.html
[4] IMP-NG, JSR228, Standard

2.2 Terms and Abbreviations

Abbreviation	Description
API	Application Program Interface
ASC	Asynchronous Serial Controller
CLDC	Connected Limited Device Configuration
CMTK	Cinterion Mobility Toolkit
CSD	Circuit-Switched Data
DAI	Digital Audio Interface

Abbreviation	Description
DCD	Data Carrier Detect
DSR	Data Set Ready
FFS	Flash File System
GPIO	General Purpose I/O
GPRS	General Packet Radio Service
HTTP	Hypertext Transfer Protocol
I/O	Input/Output
IDE	Integrated Development Environment
IMC	Inter-MIDlet Communication (com.sun.midp.io.pipe.PipeConnection)
IMP-NG	Information Module Profile - Next Generation
IP	Internet Protocol
Java ME™	Java Micro Edition (also known as J2ME)
Java SE™	Java Standard Edition
JAD	Java Application Description
JAR	Java Archive
JDK	Java Development Kit
JSR	Java Specification Request
JRC	Java Remote Control
JVM	Java Virtual Machine
LED	Light Emitting Diode
ME	Mobile Equipment
MES	Module Exchange Suite
MIDP	Mobile Information Device Protocol
ODD	On Device Debugging
OTA	Over The Air
OTAP	Over The Air Provisioning of Java Applications
PDP	Packet Data Protocol
PDU	Protocol Data Unit
SDK	Software Development Kit
SMS	Short Message Service
TCP	Transfer Control Protocol
TLS	Transport Layer Security, formerly SSL (Secure Socket Layer)
URC	Unsolicited Result Code
URL	Universal Resource Locator
VBS	Visual Basic Script
WTK	Wireless Toolkit

3 Installation

3.1 System Requirements

The Cinterion Mobility Toolkit (CMTK) requires that you have:
- Windows XP, Windows Vista or Windows 7 installed
- 110 Mbytes free disk space for the CMTK (without JDK and IDE)
- Administration privileges
- A recent Java SE Development Kit. To install the JDK on CD, follow the instructions in Section 3.3.1.

If a Java IDE such as NetBeans (as of 6.7 or higher) or Eclipse (as of 4.2.0 - Juno or higher) is installed, the CMTK environment can be integrated into it during installation of the CMTK. To install one of the IDEs follow the installation instructions in Section 3.3.1.

If you wish to access the module via USB ensure that the USB cable is plugged between the module's USB interface and the PC.

3.2 Installation CD Content

The Cinterion Mobility Toolkit Installation CD includes:
- Wireless Toolkit. The WTK is the directory where all the necessary components for product specific Java application creation and debugging are stored. The WTK version is stored in a text file under "Program Files\Cinterion\CMTK\<product name>\WTK\VersionWTK.txt". The WTK is distributed on the CD under "program files\Cinterion\CMTK\BGS5\WTK" with
  - bin
    Various tools
  - doc
    HTML documentation of Java API
  - lib
    Libraries containing the WTK classes
  - resources
    Archive containing part of the Java API
  - runtimes, toolkit-lib
    Runtime environment required for using the WTK
- WTK samples are distributed on the CD under "All Users\Cinterion\BGS5 WTK Examples"
- Module Exchange Suite (MES). MES setup is distributed on CD under "Installer\MES-Setup.exe". MES provides tools to access the Flash file system on the module from the development environment over a serial interface. The MES may be installed separately, but can also be installed as part of CMTK - see Section 3.3.
- IMP Debug Connection. The setup is found under "Installer\IMPDbgConnectionSetup.exe". The setup installs an IDE debug modem for on device debugging (see Chapter 10). The IMP Debug Connection may be installed separately, but can also be installed as part of CMTK - see Section 3.3.
- Java SDK
  - Recent JDK for using the WTK
- NetBeans IDE 7.3
  - netbeans-7.3-javase-windows.exe
- Eclipse Juno SR2 (v4.2.0)
  - eclipse-mobile-juno-SR2-win32-x86.zip

- Eclipse Juno SR2 (64-bit) (v4.2.0)
  - eclipse-mobile-juno-SR2-win32-x86_64.zip
- Updated MTJ-Plugin for Eclipse Juno
  - org.eclipse.mtj.update-site.zip
- Documentation is distributed on the CD under "Program Files\Cinterion\CMTK\BGS5 Documentation" and includes AT Command Set as well as this Java Users Guide.

3.3 Cinterion Mobility Toolkit Installation

The CMTK is distributed on CD. The installation program automatically installs the necessary components and IDE integrations. The software can be uninstalled and updated with the install program.

This section covers the installation and removal of the CMTK including the installation of the prerequisite JDK and supported IDEs.

3.3.1 Installation Prerequisites

Before the CMTK is installed from CD a standard Java Development Kit (JDK) should be installed

- A recent Java SE Development Kit is distributed as part of the installation CD under "Contribution\". To install the JDK please call the contribution file and follow the instructions on the screen. If there is no appropriate JDK installed, the installation of the provided JDK will be offered automatically during the CMTK installation process. Once the JDK has been installed, the environment variable "path" can be altered to comfortably use the JDK tools without IDE (this is not necessary for using the Cinterion CMTK):
  - Open the Control Panel:
    Open System.
    Click on Advanced.
    Click on the Environment Variables button.
    Choose path from the list of system variables.
    Append the path for the bin directory of the newly installed SDK to the list of directories for the path variable.

Apart from the JDK installation it is recommended to install a Java Development Environment (IDE):

- The Eclipse as well as the NetBeans IDE are distributed as part of the installation CD under "Contribution\eclipse-mobile-juno-SR2-win32.zip" resp. "Contribution\eclipse-mobile-juno-SR2-win32_64.zip" (for 64-bit systems with only a 64 bit version of the JDK installed) or "Contribution\netbeans-7.3-javase-windows.exe". To install NetBeans call the contribution setup file. To install Eclipse unzip the appropriate contribution archive to the desired destination directory. An Eclipse IDE may also be installed as part of the CMTK installation process described in Section 3.3.2.
  Note that the Eclipse IDE provided on the installation CD is a specially adapted "Eclipse for Mobile Developers package" with a pre-integrated MTJ plugin that is necessary for J2ME development. If employing any other Eclipse IDE variant or the NetBeans IDE also provided on the installation CD, it is therefore required to install additional plugins containing mobile resp. mobility tools. Please refer to Chapter 10 for more information on how to install these plugins for Eclipse and NetBeans.

3.3.2 Installing CMTK

Before you start the installation please make sure all applications - especially possible IDEs - are closed:

1. Insert the installation CD and start Setup.exe. When the dialog box appears press the Next button to start the CMTK installation.

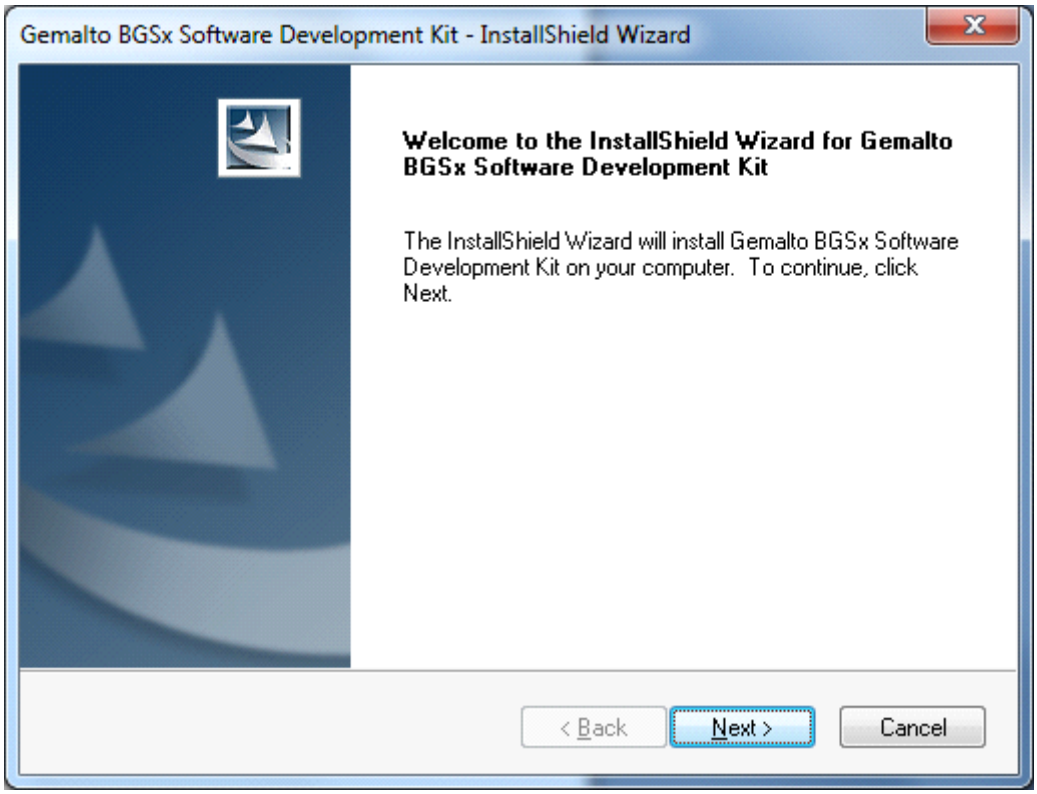

Figure 2: CMTK - InstallShield Wizard

2. Read the CMTK license agreement. If you accept the agreement, press "Yes" to continue with the installation.
3. Read the information about the installation and the use of the CMTK. Press "Next" to continue.

4. Install the Module Exchange Suite (MES) as part of the CMTK installation. MES provides tools to access the Flash file system on the module from the development environment over a serial interface. File transfers between PC and module are greatly facilitated by this suite. MES is installed to "Program Files\Cinterion\Module Exchange Suite".

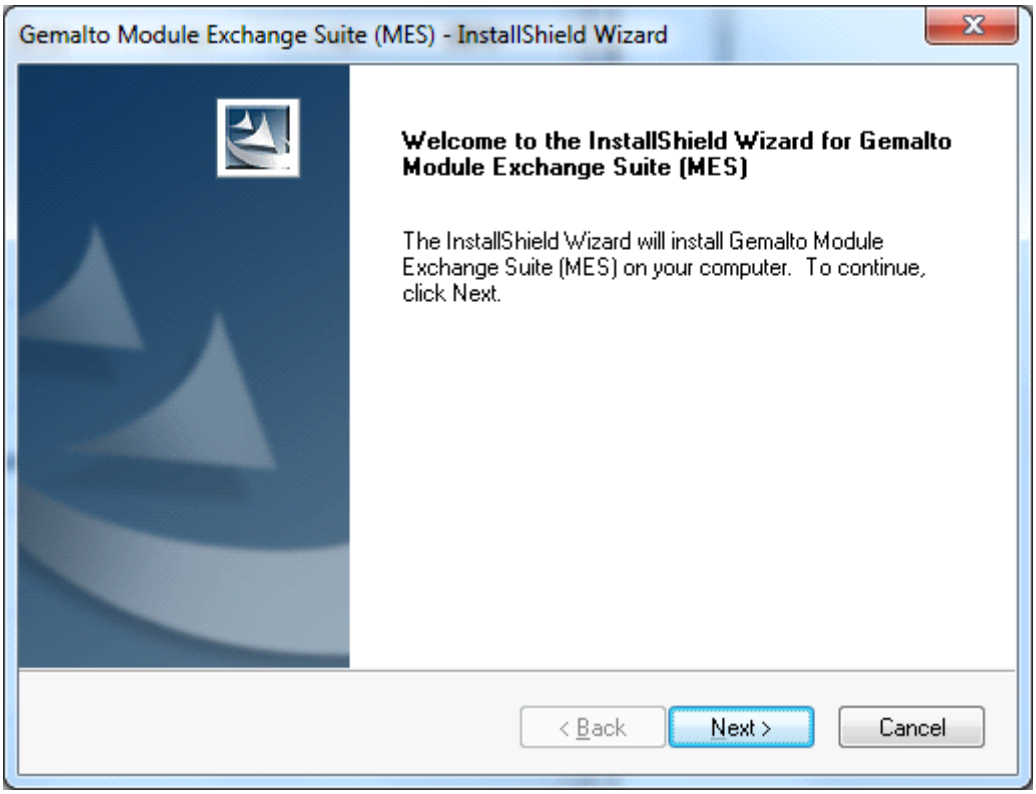

Figure 3: Module Exchange Suite - InstallShield Wizard

5. Read the MES license agreement. If you accept the agreement, press "Yes" to continue with the installation.
6. Click Finish to conclude the MES installation as part of the CMTK installation (a computer restart is required at some later point to complete the MES installation).

7. Install the IMP debug connection as part of the CMTK installation.

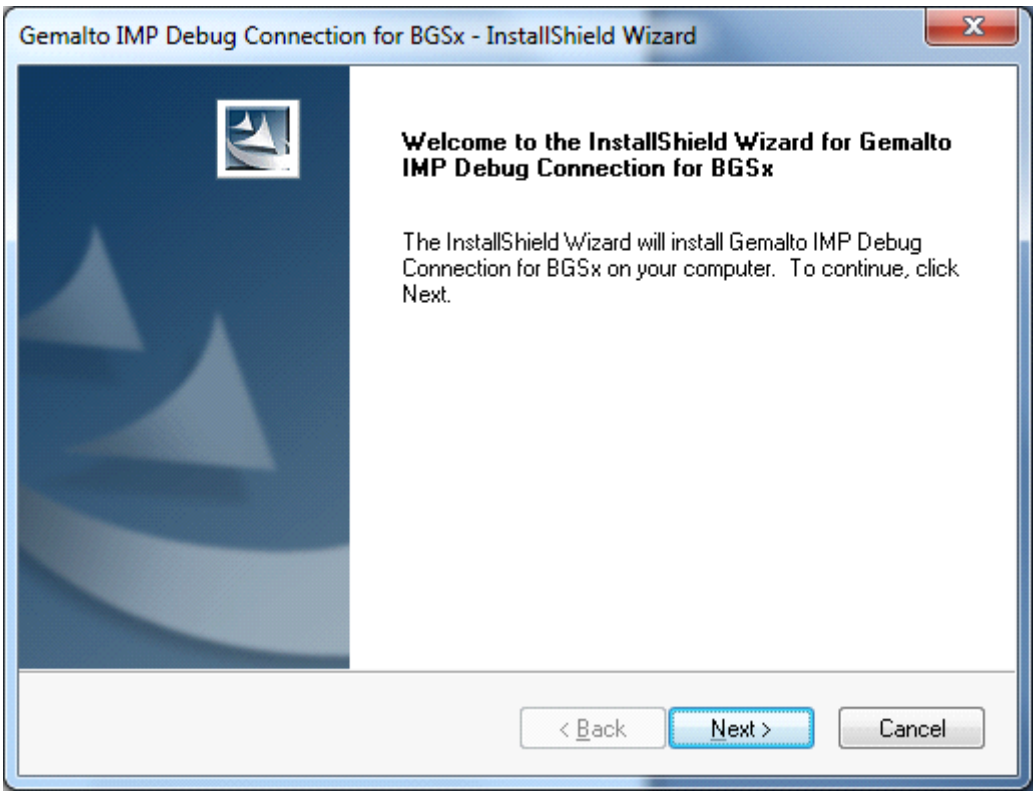

Figure 4: IMP Debug Connection - InstallShield Wizard

8. Continue with installation by installing the corresponding Gemalto M2M device software, i.e., modem driver.

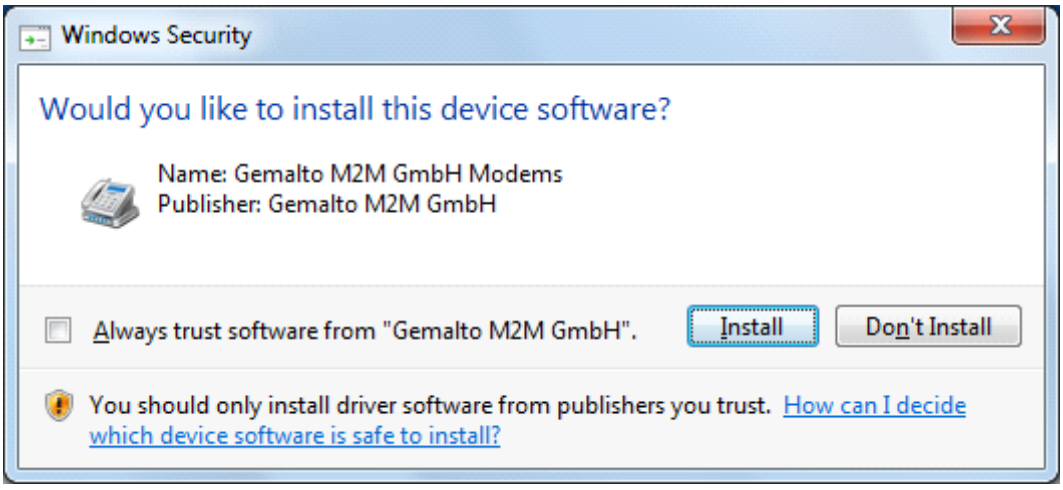

Figure 5: Device software: Modem driver installation

Note that this dialog appears only as of Windows Vista and only if the checkbox "Always trust software from "Gemalto M2M GmbH" was not selected during previous Gemalto M2M device software installations. Otherwise the necessary drivers are installed automatically.

9. Scan COM ports for available Java module. The scan may be skipped and can be repeated later as part of a repair installation. This is done by selecting the "Gemalto IMP Debug Connection for BGSx" from Control Panel and clicking "Change".

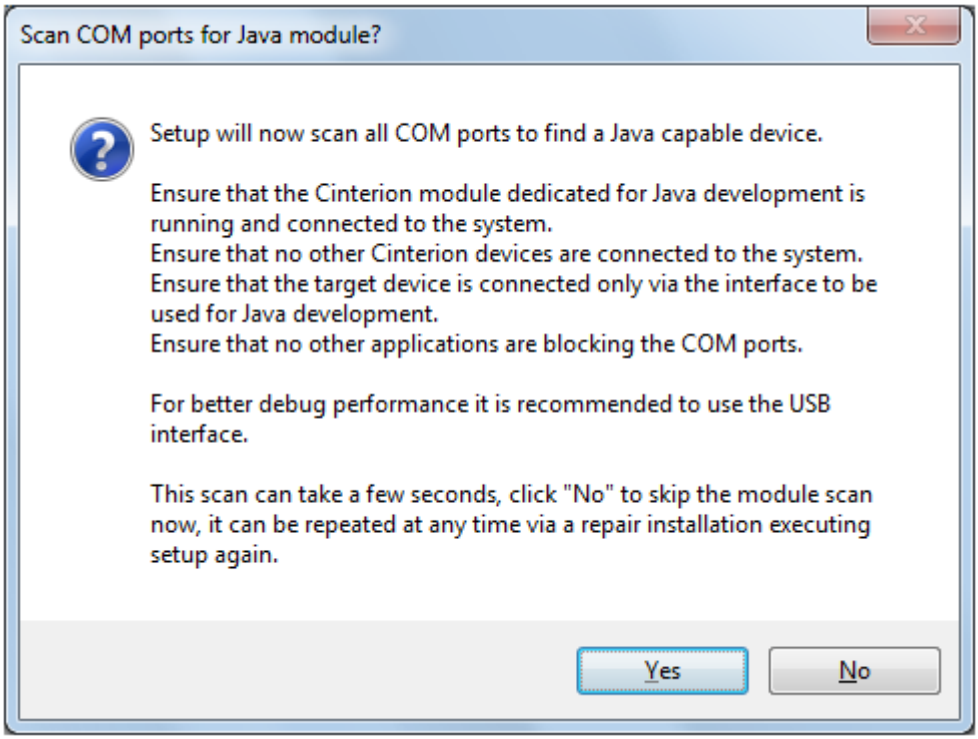

Figure 6: Scan COM ports for available JAVA module

10. Click Finish to complete the IMP Debug Connection installation.
11. Scan system for supported Java IDEs to automatically integrate the WTK into. Please ensure that none of the possibly installed Java IDEs is running before the scan is started. The scan may be skipped and can be repeated later as part of a repair installation. This is done by selecting the "Gemalto BGSx Software Development Kit" from Control Panel and clicking "Change".

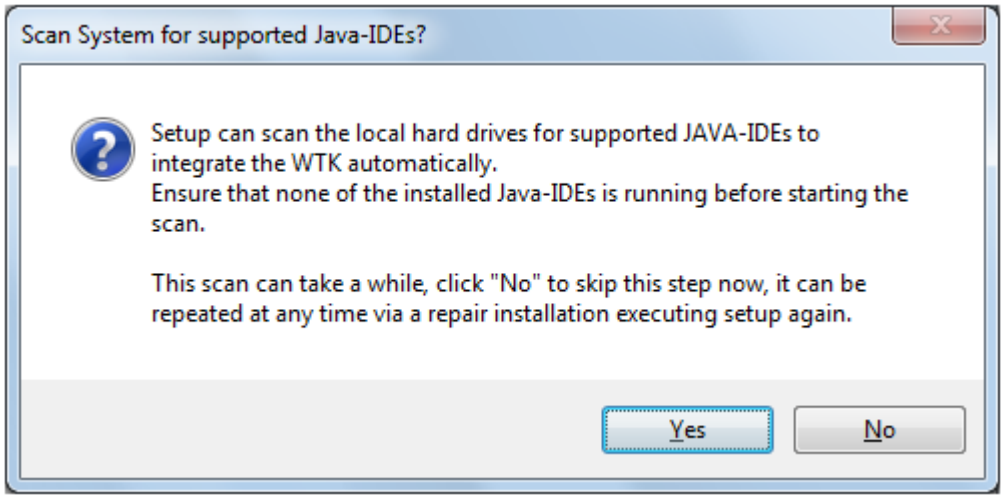

Figure 7: Scan for supported JAVA IDEs

12. If the above scan did not deliver any supported Java IDE, it is possible to install the Eclipse IDE provided on the installation CD.

Figure 8: Query to install Eclipse IDE as part of CMTK installation

13. Click Finish to complete the CMTK installation.

3.4 CMTK Uninstall

The CMTK install package comes with an uninstall facility. The entire CMTK or parts of the package can be removed. To start the uninstall facility, open the Control Panel, select Add/Remove Programs, select the desired CMTK, e.g. "Gemalto BGSx Software Development Kit" and follow the instructions. The standard modem and dial-up network connection (DUN) are uninstalled automatically.

The Module Exchange Suite (MES) is not uninstalled automatically with the CMTK. To uninstall MES as well, please run the MES uninstall facility. To run the uninstall program, open the Control Panel, select Add/Remove Programs, select "Gemalto Module Exchange Suite (MES)" and follow the instructions. MES might still be used by other CMTK versions and should in this case not be uninstalled.

Please keep in mind that standard modem (or USB modem) and dial-up network connection are required for a proper working of the CMTK on-device debugging.

3.5 Upgrades

The CMTK 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 CMTK and the interfaces to it.

4.1 Software Architecture

With the CMTK it is possible to develop a Java application on a PC and to execute it on the Java enabled module. The application is then loaded onto the module. The platform comprises:

- Based on Oracle Java ME™ Embedded 3.2 (http://www.oracle.com/us/technologies/java/embedded/micro-edition/). Platform is compliant to CLDC 1.1 HI (JSR139) and IMP-NG (JSR228) Java standards and is capable of running multiple MIDlets in parallel with inter-MIDlet communication.
- Additional Java standard APIs:
  - JSR75 (FileConnection)
  - JSR177 (CRYPTO)
  - JSR280 (XML)
- Additional Java proprietary APIs:
  - AT Command API
  - Bearer Control API
  - Watchdog API
- Additional accessible periphery for Java applications
  - I/O pins usable for example as: Output: status LEDs or Input: Emergency Button
  - I^2C interface, SPI interface
  - Digital Analog Converter (DAC),
  - Analog Digital Converters (ADC)
  - Serial interfaces (RS-232 API): All of the modules serial interfaces (ASC0, ASC1, USB modem as well as USB ports) can be used, for example, with an external meter.

  For detailed information see Section 4.2.
- Memory space for Java programs:
  Flash File System: around 10 MB.
  RAM: around 6MB for Java user application. Plus additional RAM for Gemalto application ("JRC") and Just-in-Time Compiler execution optimization.
  Application code and data share the space in the flash file system and in RAM.

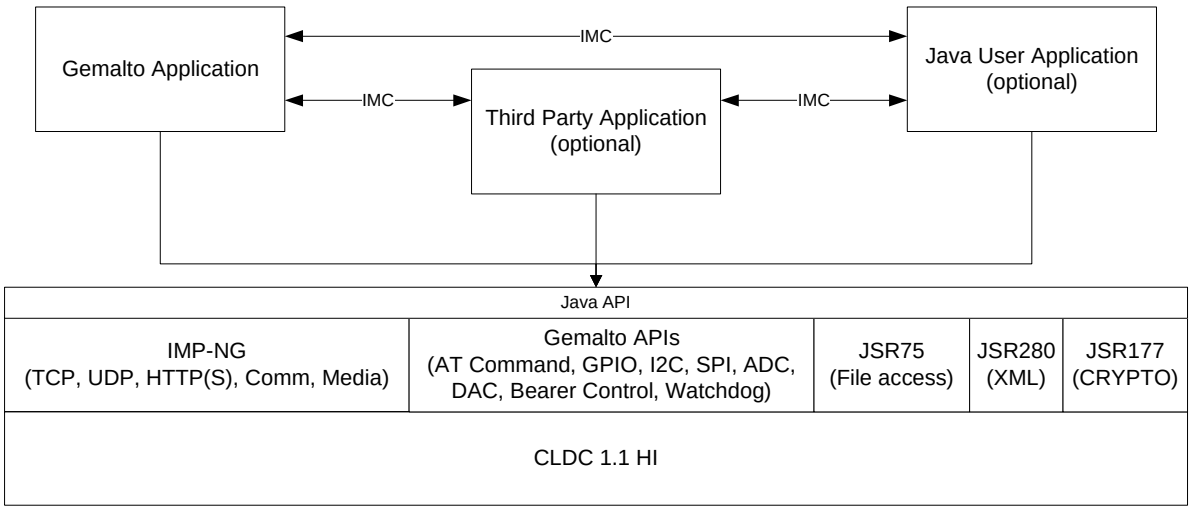

Figure 9: System architecture

4.2 Hardware Interfaces

Not all hardware interfaces can be used at the same time. Some of them share physical pins and can therefore only be used alternatively. The pins will have to be configured by AT command. After doing so, the module will automatically start up with this configuration.

Basically, GPIO pins can alternatively be used as SPI, PWM/DAC, ASC1 or DAI interface. For details see [1]. USB, I^2C and ADC interfaces do not share pins.

4.2.1 ASC0 - Serial Device

ASC0, an Asynchronous Serial Controller, is a 9-wire serial interface. It is described in [2]. When not used by a 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 System.out. Refer to [3] for details.

4.2.2 ASC1 - Serial Device

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 or by a running Java application as RS-232 port or/and System.out.

4.2.3 General Purpose I/O

There are a number of I/O pins that can be configured for general purpose I/O (GPIO). One pin can also be configured as a pulse counter. All lines can be accessed under Java by a Java API. See [1] and [2] for information about usage and startup behavior.

4.2.4 ADC

Accessible by a Java API. See [1] and [2] for details.

4.2.5 PWM/DAC

Accessible by a Java API. See [1] and [2] for details.

4.2.6 Digital Audio Interface (DAI)

The DAI is not directly controlled by Java. It can be configured by AT command and then serves as an alternative audio interface to analog audio. Refer to [1] and [2] for more information.

4.2.7 I2C

Accessible by a Java API. See [1] and [2] for details.

4.2.8 SPI

Accessible by a Java API. See [1] and [2] for details.

4.2.9 USB

Accessed by Java RS-232 API as a serial emulation. The USB interface comprises an USB modem and available USB ports. The available ports provide an AT command interface to the module (i.e., ports 1 and 2). USB modem and ports may be employed as regular USB interfaces, including usage as System.out and for on-device debugging (ODD) during the application development phase.

4.3 Other Interfaces

4.3.1 IP Networking

Because the used network connection (GPRS) is fully transparent to the Java interface, the GPRS parameters must be defined separately either by the AT command AT^SJNET [1] or by parameters given to the Connector.open() method, see [3].

4.3.2 Media

The playTone method and the tone sequence player are supported.

4.3.3 Others

Neither the PushRegistry interfaces and mechanisms nor any URL schemes for the PlatformRequest method are supported. See [3].

4.4 Data Flow of a Java Application Running on the Module

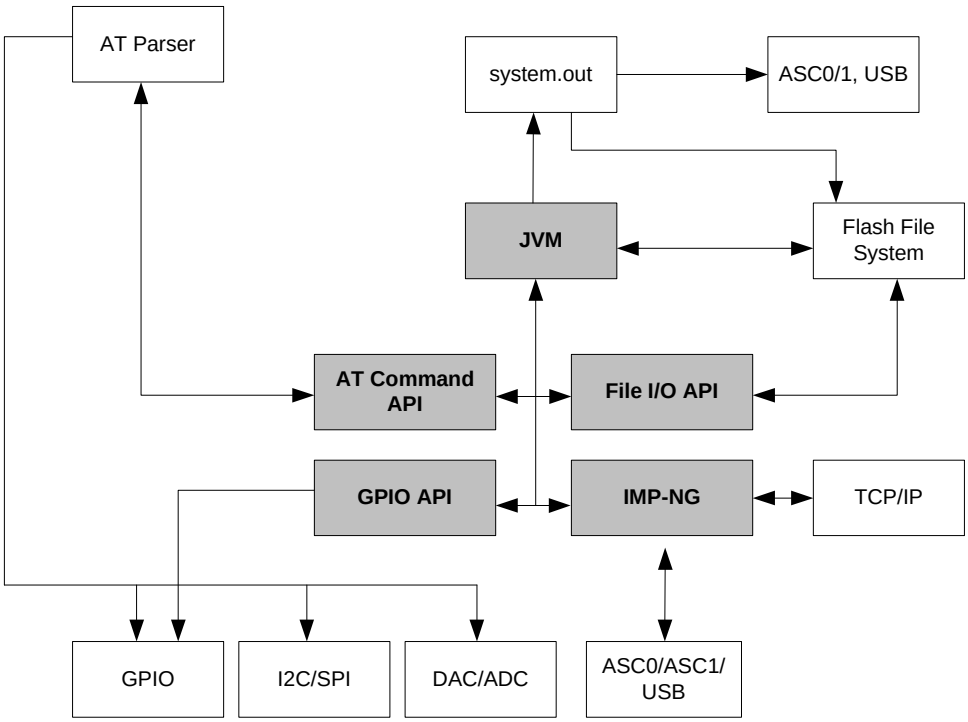

Figure 10: Data flow of a Java application running on the module.

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 18.
The compiled Java applications are stored as JAR files in the Flash File System of the 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
- I^2C: I^2C 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 ASC0/1
- GPIO API: Java API to GPIO pins and pulse counter
- I^2C/SPI API: Java API to access I^2C/SPI Bus

- ADC/DAC API: Java API to access ADC and DAC
- Watchdog API: Java API to HW watchdog
- Bearer Control API: Java API for bearer state information and hang-up.
- XML API: XML parsing and composition services
- Crypto API: encryption services

4.5 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 multiple AT parsers available
- There is hardware access to
  - two serial interfaces: ASC1 and ASC0 (both fully accessible).
  - general purpose I/O. To configure the hardware access, please refer to [1] and [2].
  - I^2C/SPI
  - All restrictions of combinations are described in Section 4.5.1
- A Java application has:
  - instances of the AT command class, each of which would, in turn, be attached to one of the AT parsers.
  - two instances of access to a serial interface, ASC0 and ASC1, through the CommConnection API. Access to the control lines of these interfaces through CommConnectionControlLines.
  - System.out over any serial interface or into the file system

4.5.1 Module States

In contrast to previous Gemalto M2M Java products (e.g., Cinterion® TC65i) where the Java runtime environment was only started when the Java MIDlet was started (by AT^SJRA or autostart) the Java runtime environment now always starts at power-up of the module. But also in contrast to previous products, Java does not immediately take control of all interfaces, i.e., the module is accessible via AT command over all serial interfaces even when Java is running. Interface resources are only claimed when a Java application opens the interface or Java System.out is configured to this interface.

4.5.1.1 State 1: Default – Module after Power-Up

This is the default state.

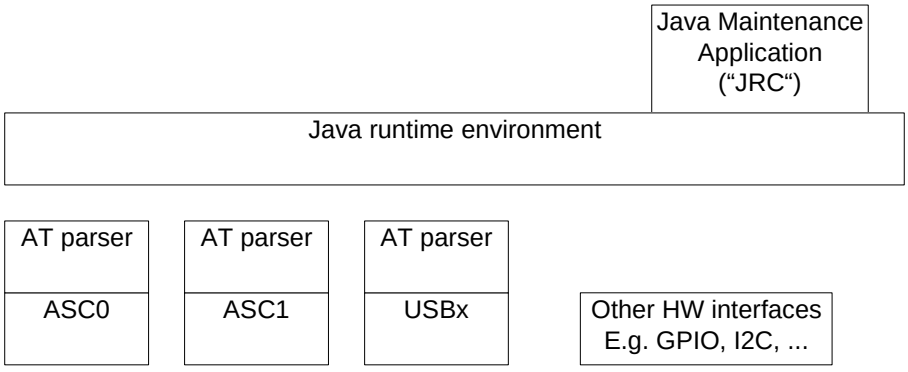

Figure 11: Module State 1

4.5.1.2 State 2: Default - System.out is Configured

This is the default state with a System.out interface configured to ASC1, i.e. one of the at command interfaces is no longer available.

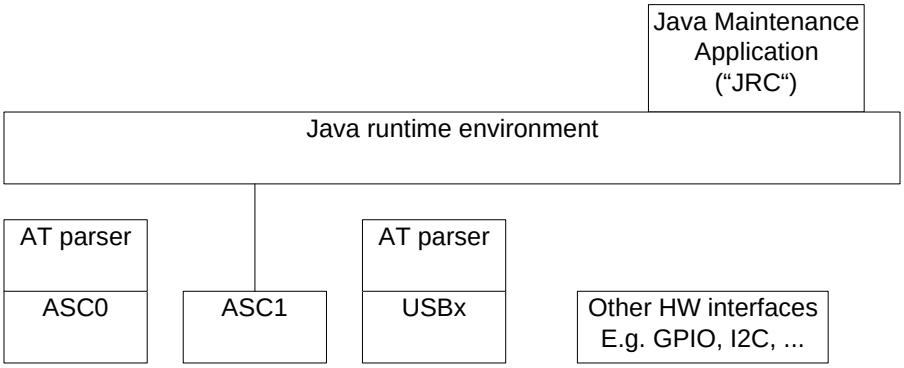

Figure 12: Module State 2

4.5.1.3 State 3: User Specific Java Application Started

A user specific Java application has been installed and started, either with AT^SJRA, AT^SJAM and/or autostart. The application did not open any of the depicted interfaces.

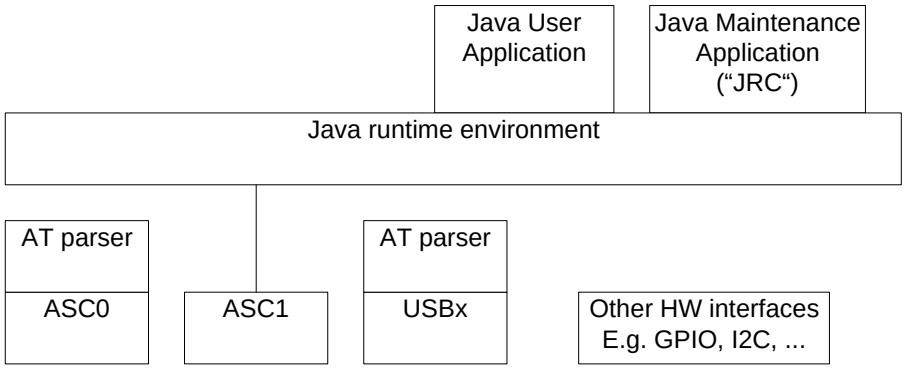

Figure 13: Module State 3

4.5.1.4 State 4: Typical Deployment

A Java user application has been installed and started. The application opened the ASC0 interface, an AT parser and an additional HW interface.

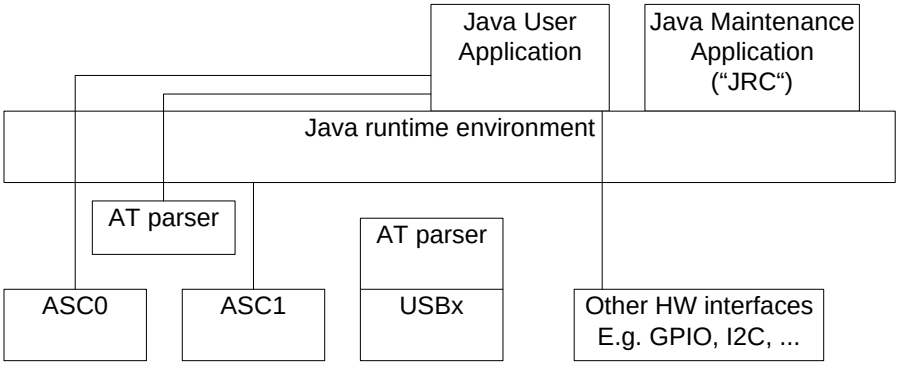

Figure 14: Module State 4

5 Maintenance

The basic maintenance features of the ME are described below. Explicit details of these functions and modes can be found in [1] and [2].

5.1 IP Service

Java supports more than one PDP context via several connections. A connection is defined by means of:

- Optional additional parameters for the Connector.Open() method of the used connection
- The default parameter set configured with AT^SJNET

The unique connection identifier is always its name (APN).

Every access to a network resource tries to open/use a PDP context (with or without PDP context parameters) and the following steps are executed in the given order:

1. Debugger default connection is checked: If the debugger connection is set to default (via AT^SCFG), this connection is used.
2. In case a PDP context with an identical name (APN) is already open, the established connection (with its additional parameters) is used. It is not possible to run different connections with the same APN name.
3. If an APN is set, this APN is used. Should this not be possible because too many contexts are already open, connection setup will fail.
4. If no APN is set, it is checked whether a default context (see AT^SJNET for details on how to set dialup network access parameters) is configured and open. The context is used if already open or opened if configured and not yet open.
5. If no APN is set and no default context is available (via AT^SJNET), find any open context and use it.
6. If no APN is set, no default context is available (via AT^SJNET) and no open context is available as well, then try to open a new context with empty parameters.

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

- As for the Java connector, IP services also use contexts identified by their APN, with the same rules as described above to open a context. Any possible sharing of contexts is possible (Java uses contexts from IP services and vice versa).
- The connection profile is deactivated once all applications are finished: Java may setup its networking idle time for every connection. 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. It is recommended to set the parameters to the same values as the Java networking parameters. This way it does not matter if a connection is activated by the Internet Services or Java.

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

- When an open TCP connection is disrupted (e.g. the other side dies/is switched off) it takes a complete re-transmission timeout during which re-transmissions are sent, until the situation is detected as an error (in Java an exception is thrown). The re-transmission timeout may set via IP service commands (for IP services) or the netExtension class for regular Java network access.
- The number of IP services used in parallel should be kept low. An active IP service uses up resources and may deteriorate the overall performance.
- If a user rapidly closes and opens TCP/IP connections (e.g. SocketConnection, HTTPConnection), a ConnectionNotFoundException reading "No buffer space available" may be thrown, explaining that all TCP/IP socket resources are exhausted. In the worst case, opening further TCP/IP connections is locked for up to 60 seconds.
- 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, or the BearerControl / BearerControlListener Java classes.
- When trying to start the bearer when it is still in the process of shutting down, e.g. right after a "network idle timeout" an IOException is thrown. Either use a delay or wait for bearer state to actually say "down". Use the BearerControl class. This class provides bearer state information and methods to hang up.

5.2 Power Saving

The module supports several power saving modes which can be configured by the AT command AT^SPOW [1]. 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 power saving mode (AT^SPOW=0) the serial interface cannot be accessed. In cyclic power saving mode (AT^SPOW=2) the serial interface can be used with hardware flow control (CTS/RTS).
- In all power saving modes the GPIO polling frequency is reduced when the module goes to sleep so that only signal changes that 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 [1] or refer to [2].

Java power saving efficiency:

- As long as any Java thread is active, power consumption cannot be reduced, regardless whether any power saving 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 prevent power saving.
- When using networking functionality the module also saves power while a PDP context is activated but without ongoing traffic. To further improve power saving close all connectors and hang-up the bearer manually.

5.3 Airplane Mode

While the module is in airplane mode the RF interface 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+CFUN command. A Java application must be able to handle this mode.

Since the RF interface is switched off all classes related to networking connections, e.g. SocketConnection, UDPDatagramConnection, SocketServerConnection, HTTPConnection, will throw an exception when accessed.

5.4 Alarm

The ALARM can be set with the AT+CALA command. Please refer to the AT Command Set [1] and Hardware Interface Description [2] for more information. It is possible to 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 and with it any Java application.

5.5 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 module log off from the network and allows the software to enter a secure state and save all data. Further details can be found in [2].

5.5.1 Automatic Shutdown

The ME is switched off automatically in different situations:

- under- or overtemperature

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.

For further detail refer to the command AT^SCTM described in the AT Command Set [1]. In addition, a description of the shutdown procedure can be found in [2].

5.5.2 Manual Shutdown

The module can be switched off manually with the AT command, AT^SMSO. In this case the MIDlet's destroyApp() method is called and the application has 10s time to clean up and call the notifyDestroyed() method. After the 10s the VM is shut down.

5.5.3 Restart after Switch Off

When the module is switched off without setting an alarm time (see the AT Command Set [1]), e.g. after a power failure, external hardware must restart the module by means of the Ignition line (ON, AUTO_ON). The Hardware Interface Description [2] explains how to power up the module using these lines.

5.5.4 Watchdog

The Watchdog class allows to access the HW watchdog of the system from application level. Depending on the setting (AT^SCFG) the userware watchdog can do nothing, switch-off or reboot the system.

5.6 Special AT Command Set for Java Applications

For the full AT command set refer to [1]. There are differences in the behaviour between AT commands issued from a Java application and AT commands issued over a serial interface.

5.6.1 Switching from Data Mode to Command Mode

Cancellation of the data flow with "+++" is not available in Java applications, see [1] for details. To break the data flow use *breakConnection().* For details refer to [3].

5.6.2 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, 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 [3]) on this AT Command API instance before passing the first AT Command.

5.6.3 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

5.6.4 Configuration of Serial Interface (ASC0, ASC1)

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 physical serial interface are pointless on the AT Command API and should therefore not be used. This includes the commands:
- AT+IPR (sets a fixed local bit rate)
- 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. However, the following restrictions apply in configuring the serial interface: Baudrate: Only 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 115200, 230400 and 460800 are supported; Stopbits: Only "1" is supported; Parity: "7N1" is not supported.

5.6.5 Java Commands

There is a small set of special Java AT commands:
- AT^SJRA, start a Java application
- AT^SJAM, start/stop, install/remove Java applications
- AT^SJNET, configuration of Java networking connections
- AT^SJOTAP, start and configuration of over the air provisioning
- AT^SJMSEC, security configuration (not yet available)

Refer to the AT command Set [1].

5.7 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 [1] for details) and is non-volatile.

5.7.1 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.

5.7.2 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. Because excessive writing to the module's flash file system decreases the life time of the flash memory, we recommend using the System.out to file redirection only during development phases.

5.8 Restrictions

5.8.1 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 126 characters. It is recommended that names of classes and files be distinguished by more than case sensitivity.

The maximum number of directories is limited to 100, the maximum number of files is limited to 1100. Please note that the actual amount of files and directories may be lower, because some of them are used internally by the java application manager.

Avoid using any blank in the names of JAR or JAD files. Otherwise the file explorer might not recognize the names. If the OTAP server adds a blank into the filenames, problems with OTAP will occur.

Please note that the file system will have major performance drops, if small data blocks are appended to large files (>1MByte) or if the file system is nearly full. It is recommended to write blocks of at least 4 KByte to large files. Please also note that writing continuously large data blocks to the file system can slow down execution of all other java tasks. To avoid this a Thread.sleep(100) should be added to a writing loop.

5.8.2 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.

5.8.3 JAD File Size

The Java Application Descriptor (JAD) File can be used to pass configuration data to the MIDlet. JAD file sizes of up to 30 KByte are allowed.

5.8.4 AT Command API

Before using a new AT Command instance it should be resetted (AT&F). Because of the employed multi-MIDlet architecture another MIDlet may have used this instance before, but with a different setup, e.g. echo.

5.9 System Time

The return value System.currentTimeMillis() is derived from the systems real time clock (RTC). Therefore setting the RTC with the AT+CCLK AT command also configures the Java time. The Calendar class also uses System.currentTimeMillis() to set the initial date.

6 MIDlets

Java applications in Java ME™ are called MIDlets. The MIDlet code structure is very similar to applet code. There is no main method and MIDlets always extend the MIDlet class. The MIDlet class in the MIDlet package provides methods to manage a MIDlet's life cycle.

6.1 MIDlet Documentation

IMP-NG and MIDlet documentation can be found in the html document directory of the wtk, …\Cinterion\CMTK\<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^SJAM or 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 1: 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 Multiple MIDlets

The module is capable of running multiple MIDlets in parallel. All MIDlets are alike in that they have to share the same system resources like interfaces, memory and processing power. This has to be considered and controlled - especially when different MIDlets need to access the same system resources.

The maximum number of MIDlets running simultaneously is four while in normal operation and seven while doing on-device debugging. When the maximum number of MIDlets is reached a further MIDlet will still start ("startApp" method is called) but it will then terminate immediately with an error message on System.out.

It is recommended to keep the total number of MIDlets low in real world applications.

6.4 Hello World MIDlet

Here is a sample HelloWorld program.

```
/**
 * 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 and the baud rate is configured to a fixed value of 921600, 460800, 230400, 115200, 57600, 38400 or 19200. Please also note that the use of the Module Exchange Suite resets the module's AT command settings to their factory defaults (just as the command AT&F would). Possible user defined profiles stored non-volatile with AT&W will have to be restored with ATZ after usage.

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 under Module Exchange Suite installation 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 11.

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).
- Ensure that the OTAP server does not add a blank to the names of JAR and JAD files, because this will cause problems with OTAP.

8 Over The Air Provisioning (OTAP)

8.1 Introduction to OTAP

OTA (Over The Air) Provisioning of Java Applications is 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(S) server that is accessible over a TCP/IP connection, and an SMS sender that can send Class1, PID $7d short messages. This is the PID reserved for a module's data download.

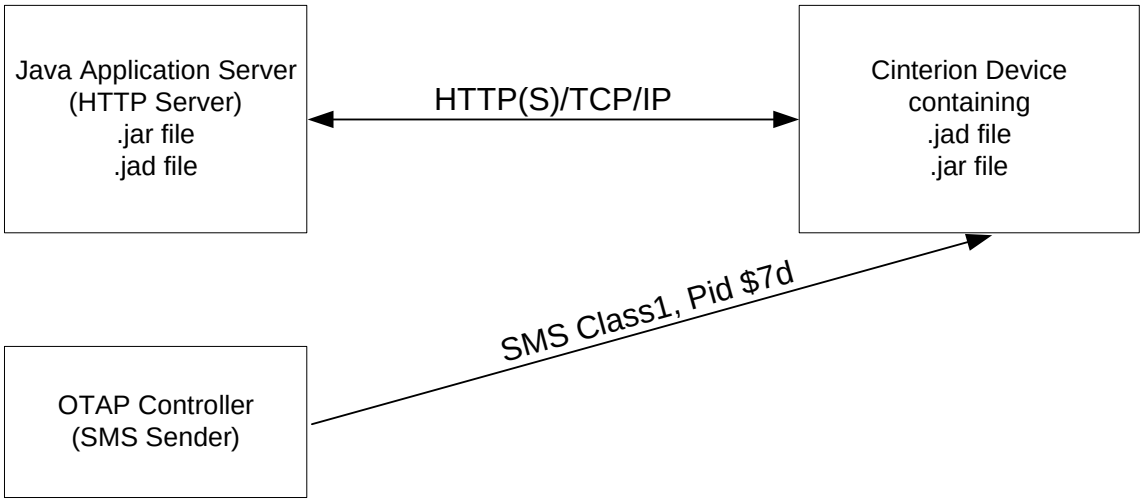

Figure 15: OTAP Overview

The Java Application Server (HTTP(S) 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 module can send a result response over HTTP if supported by the server.

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) 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 [1] 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). If the URL configured by AT command is not the complete JAD file path, the URL send by SM is appended to it. This can be used to handle different Java applications on the same server. If you would like to encrypt the data transmission use HTTPS, e.g. https://someserver.net/somefile.jad.
- 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.
- 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 descriptor.

There are parameters 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.
- Ignore SM PID: when setting this the PID in received SMs is ignored.
- Hide HTTP authentication parameters: this allows to hide the HTTP authentication parameters in the AT^SJOTAP read command and the OTAP tracer.

Table 2: Parameters and keywords

Parameters	Max. Length AT	Keyword SM	Install/update	delete
JAD File URL	100	JADURL	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
Ignore SM PID	--	--	--	--
Hide HTTP authentication parameters	--	--	--	--

The length of the string parameters in the AT command is limited (see Table 2), 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 2). "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 or 7 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
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
JADURL:http://www.greatcompany.com/coolapps/mega.jad
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 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.

OTAP is implemented as an SM listener in the system, i.e. OTAP does not acknowledge or delete any SMs. So the OTAP SM is also subject to the normal SM processing in the module. The customer application needs to take care that OTAP SMs do not fill up any SM storage.

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 (or just mytest.jar if the rest of the URL is equal to the JAD file URL). 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 Section 8.7) is returned.
- MIDlet-Jar-Size must contain the correct size of the jar file, otherwise result 904 (see Section 8.7) is returned.

Example:

```
MIDlet-Name: MyTest
MIDlet-Version: 1.0.1
MIDlet-Vendor: TLR Inc.
MIDlet-Jar-URL: 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

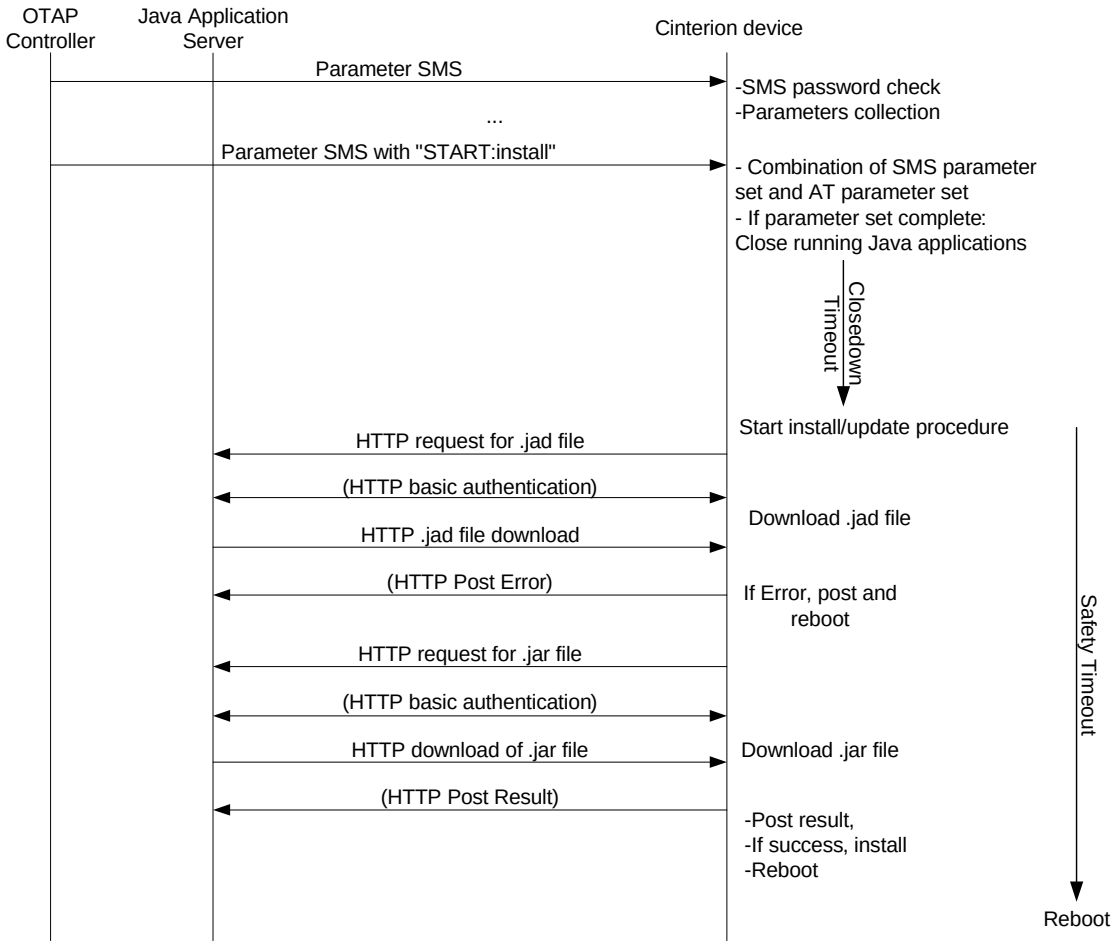

Figure 16: OTAP: Install/Update Information Flow (messages in brackets are optional)

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.

8.6.2 Delete

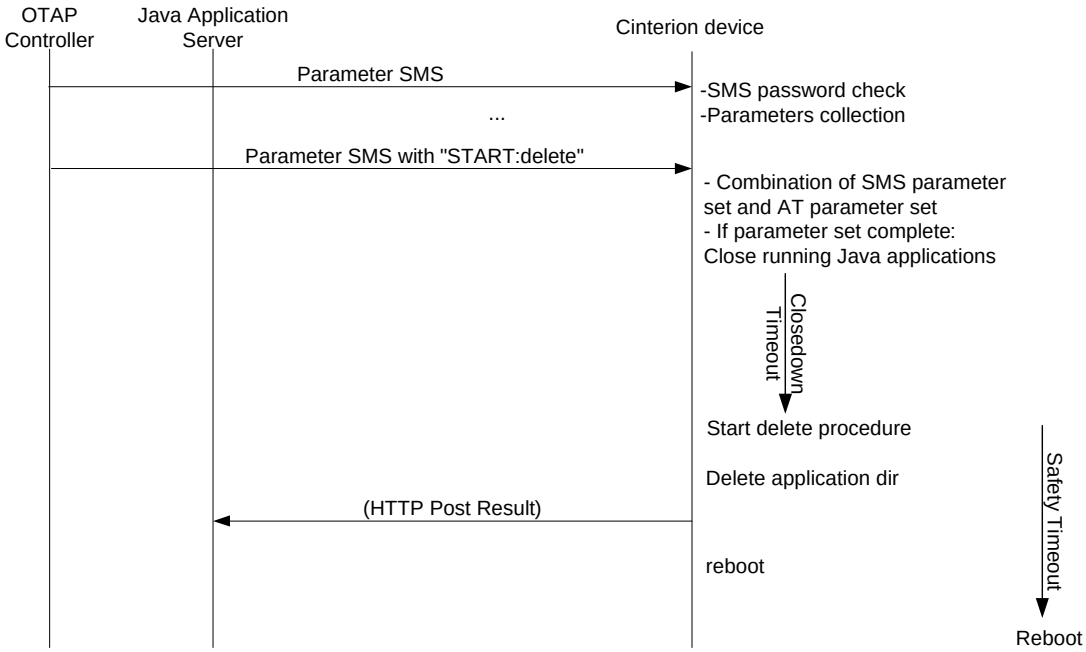

Figure 17: OTAP: Delete Information Flow (messages in brackets are optional)

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.

8.7 Time Out Values and Result Codes

Timeouts:
- Closedown Timeout: 10 seconds
- Safety Timeout: 20 minutes

Result Codes: Supported status codes in body of the HTTP POST request:
- 900 Success
- 901 Insufficient memory in filesystem
- 902 User cancelled, i.e. HTTP authentication failed
- 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.
- 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 Section 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 or the PIN needs to be disabled.

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 11 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.

For security reason the reception of OTAP SMSs is disabled until the AT^SJOTAP write command has been issued at least once. This saves non Java users from using an open module unintentionally.

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 (Section 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 HTTP Connection Interface to test it. If you desire to use encrypted file transfer make sure that your HTTP server is set up correctly for HTTPS.
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 Section 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 Section 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 device if the operation completed as desired.
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 applications 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 applications, 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 Section 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 CMTK 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 7.

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: Test Inc.
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: Test Inc.
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 can be found in [4].

Please note that the property "MIDlet-Jar-Size" must contain the correct size of the JAR file. Otherwise the MIDlet cannot be started. The tools "SetJadProp.exe" and "GetJadProp.exe" under the WTK bin directory provid convenient access to jad file properties via command line.

The batch files for building the provided examples are using these tools to set the property "MIDlet-Jar-Size" automatically during the build process.

Note that the ATCommand class is provided as an external library "cwmlib_1.0.jar" located in the WTK subfolder "resources". So the content of this library must be added to each MIDlet using the ATCommand class. The batch files for building the provided WTK command line examples can be used as reference.

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 samples directory "Documents and Settings\All Users\Cinterion\BGS5 WTK Examples\WTKSamples" under Windows XP or "Users\Public\Cinterion\BGS5 WTK Examples\WTKSamples" under Windows Vista and above. The samples directory can be opened directly via the WTK start menu entry.
- If the compile was successful the program can be transferred to the module and executed as described in the following chapters.

The batch files for compiling the samples are using the system environment variables BGS5_JAVA_HOME and BGS5_WTK_HOME and BGS5_CLASSPATH. The first one points to the root directory of the installed JDK, the second one to the root directory of the Cinterion-CMTK-BGS5-IMPNG installation and the last one contains the class path including all used Java libraries. The installation process sets these environment variables. A modification is usually not necessary.

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^SJAM* is used to install/remove, start/stop 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^SJAM=0,"a:/helloworld.jad",""* to install and then *AT^SJAM=1,"a:/helloworld.jad",""* to start the application.

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 CMTK 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 global autostart functionality configuration. Please refer to [1].
- In addition every MIDlet that is started automatically needs to be configured via its jad file and the following properties:
  - Oracle-MIDlet-Autostart: [0-5], autostart order. 0 means no autostart. For MIDlets with the same level the order is not defined.
  - Oracle-MIDlet-Restart: [true|false], if true the MIDlet is automatically restarted if it terminates non gracefully, e.g. by an uncaught exception.
  - Oracle-MIDlet-Restart-Count: [number], number of allowed MIDlets restarts before the whole module is rebooted
- Reinstall the MIDlet
- Restart the module.

After an automatic MIDlet start or restart, the following information is written to standard output: "MIDlet:" followed by the MIDlet class name and the action performed.

Example:
"MIDlet:com.cinterion.jrc.JRC_Midlet autostart"

9.4.2 Switch off Autostart

There are two methods for switching off the autostart feature:
- The AT^SCFG command to disable autostart globally, or
- Uninstall a single MIDlet or reinstall it with changed jad file properties using AT^SJAM

These AT command are preferable issued over one of the AT commands channels on USB.

9.4.3 Autostart Fail-Safe

There is an autostart fail-safe mechanism implemented that prevents the module from continous reboots caused by a MIDlet. Each time Java ends and the module reboots within 20s after module power-up a counter is incremented. This reboot might be caused by some malfunctioning inside a MIDlet (e.g. uncaught exceptions), by Java itself but also by the AT command AT+CFUN. If this happens six times in a row, autostart (and therefore Java) will be disabled temporarily.

If the module is rebooted after that, or if the module runs longer than 20s in between reboots, the whole autostart fail-safe mechanism starts over again from zero.

A further autostart fail-safe mechanism is based on the JRC MIDlet delivered with the module by Gemalto M2M. If this MIDlet is not started, for example because autostart is disabled, a URC ^SYSINFO: 200 is generated after 40 seconds.

The startup of this JRC MIDlet is monitored via timer. If the timer times out - the timer runtime depends on the JRC MIDlet version, but will as a rule be 30 seconds - a URC ^SYSINFO: 201 is generated, and the module restarts after another 5 seconds.

The number of restarts, initiated by unsuccessful startups from the JRC MIDlet, is monitored. If this number exceeds 8, the startup of the JRC MIDlet is disabled, a URC ^SYSINFO: 202 is generated, and a shutdown timer of 10 minutes is started. After this timeout, the module switches off.

10 Compile and Run a Program with a Java IDE

10.1 Debug Environment

10.1.1 Data Flow of a Java Application in the Debug Environment

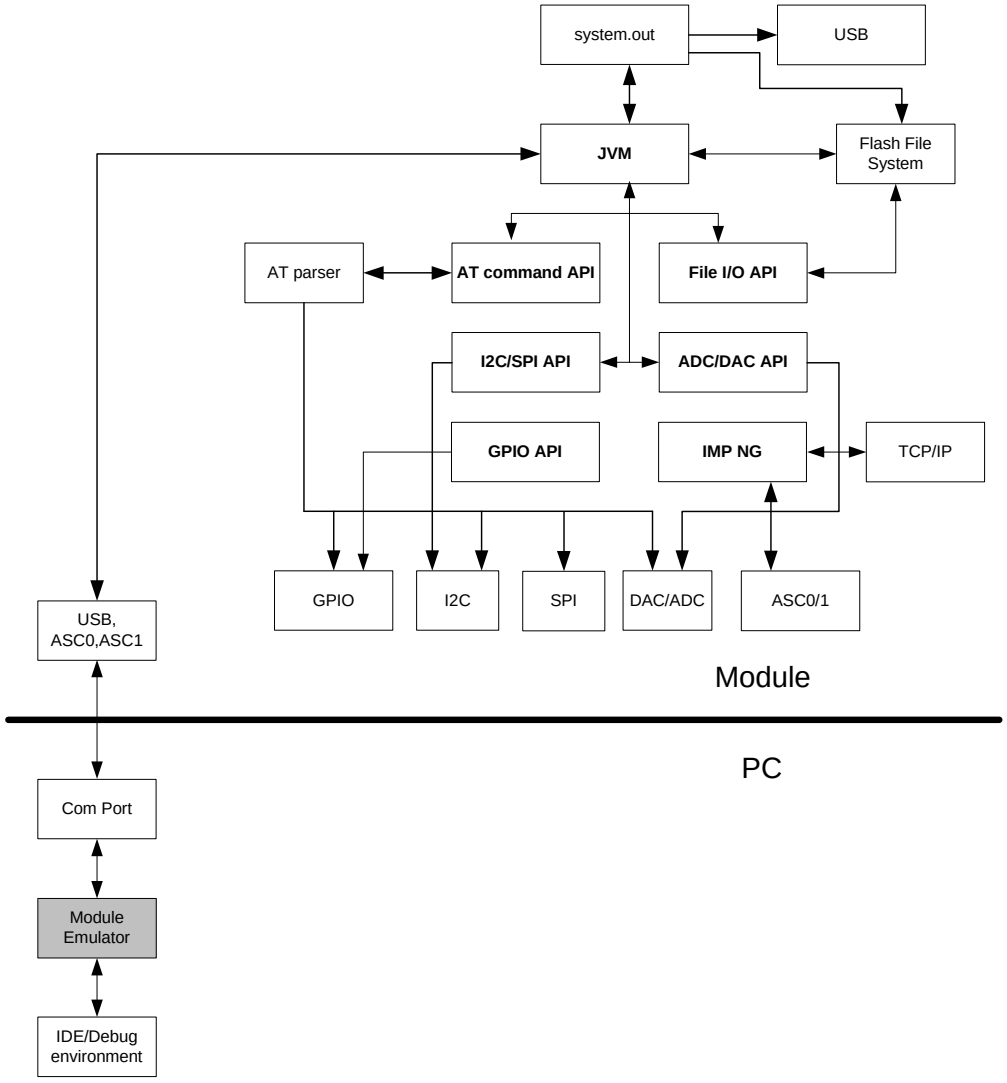

Figure 18: Data flow of a Java application in the debug environment

In the debug environment the module is connected to a PC via a serial interface. This can be a USB or 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) and 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.

10.1.2 Emulator

The ME emulator is part of the CMTK 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 runs fine without changes in WM_Debug_config.ini file and should not be modified under normal circumstances.

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 of BGSx"
- Modems: "Cinterion BGSx Java Debug Modem USB" for USB serial modem and "Cinterion BGSx Java Debug Modem Ser"
- Phone number: *88#
- Disable the Redial if line dropped option.
- Enable Connect automatically

This dial-up network (DUN) connection is installed automatically together with the required modem device during installation of the CMTK. The emulator uses always the serial port configured for this DUN connection.

It is possible to use any of the three serial interfaces (USB, ASC0, ASC1) to connect with the module, but then functionality normally available over the interface will be lost. Also, the DUN under Windows needs the DCD line of the serial interface. The ASC1 interface of the BGSx modules does not have this DCD line and the line will therefore have to be electrically set to high for debugging over ASC1 - something that cannot be done by AT command. In addition, the missing DTR input on ASC1 may prevent the module from detecting the termination of the DUN, so in some cases the module has to be reset manually after the debug session was terminated. Because of the above mentioned issues for ASC1 and for general performance reasons it is recommended to use the USB interface for debugging (see also Section 10.1.3).

If using the USB interface in a debugging session, please note that it is recommended to wait some seconds before starting a debugging session once again after the end of a previous debugging session, because the Windows operation system needs some seconds to enable the USB port once again after the "IP connection for remote debugging of BGSx" is closed.

If necessary, the IP addresses used for the debug connection can be changed. This is done in the file "WM_Debug_config.ini". For details, see also the AT^SCFG command and its "Userware/DebugInterface" parameters described in [1]. Please keep in mind that the IP address range 10.x.x.x is not supported for in device debugging!

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

10.1.3 Change Baud Rate

By default, the Cinterion CMTK installation process implements the connection necessary for on device debugging with the maximum possible baud rate supported by both the target computer and the connected module. For optimized performance Gemalto M2M recommends to use the USB interface for debug connections, because then the baud rate used on that virtual USB COM port is irrelevant.

If a serial COM port is employed however, the configured baud rate might not work reliably, depending on the quality of the used serial cable and COM port hardware. In these cases it might become necessary to change the baud rate of the debug connection manually. This has to be done as follows:

- First, the baud rate of the modem device "Cinterion BGSx Java Debug Modem Ser" has to be set to the required maximum port speed. The modem properties are available via the Windows device manager or the phone and modem options (see Figure 19).

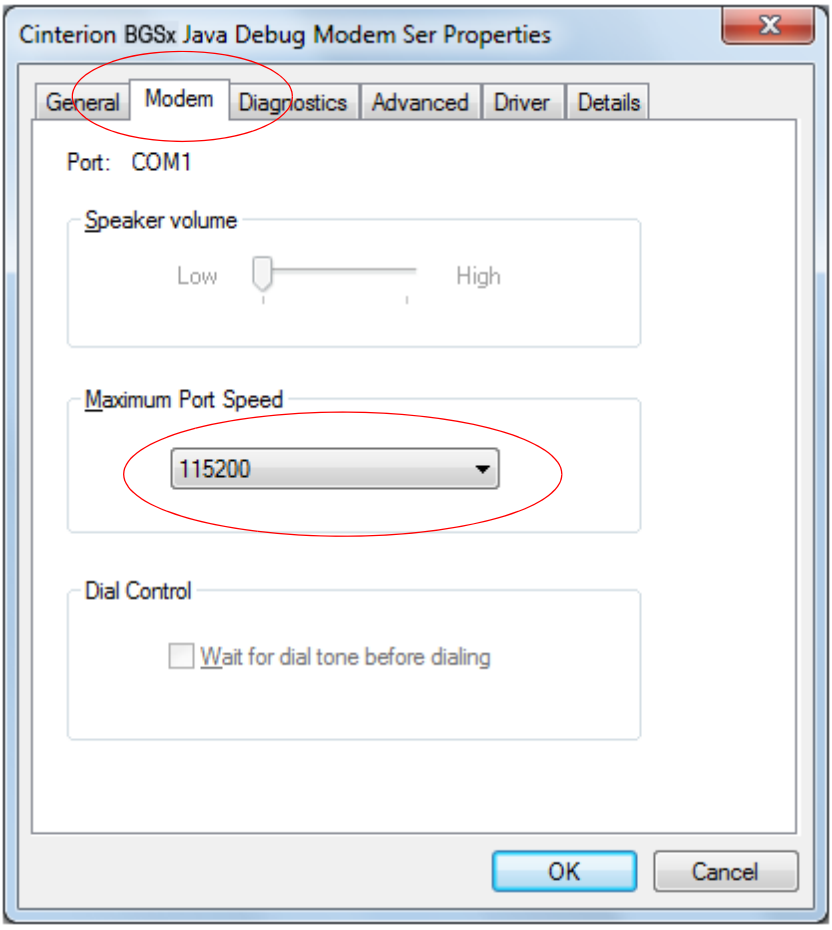

Figure 19: Specify maximum port speed for modem device

- Then the baud rate used by the debug connection "IP connection for remote debugging of BGSx" must be set to exactly the same value. The properties of the debug connection are available via the Windows network connection settings (see Figure 20 and Figure 21).

Please note that on device debugging does not work, if the maximum baud rates specified for modem device and debug connection are not identical. In such a case the emulator aborts debugging with an appropriate error message.

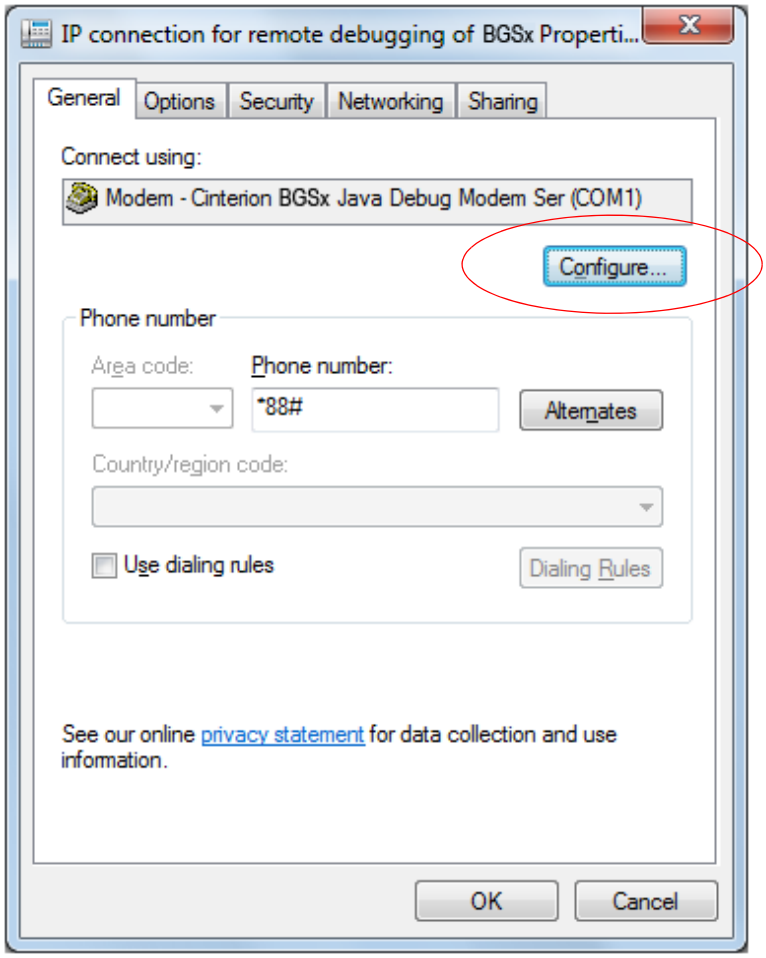

Figure 20: Configure debug connection

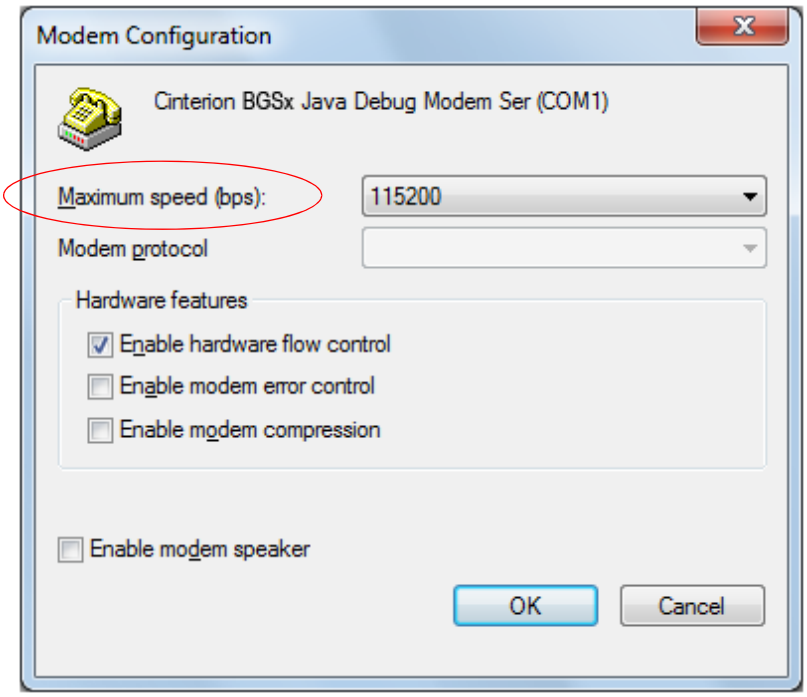

Figure 21: Specify baud rate for debug connection

10.2 Using Eclipse for Java Development

The Cinterion WTK is integrated automatically into the Eclipse IDE as of version 4.2.0 named "Juno" by the setup process if an appropriate Eclipse installation is found on the target computer. For usage of the Cinterion WTK inside Eclipse the "Mobile Tools for Java" (MTJ) plugin must be installed. Refer to Section 10.2.1 for information how to install this plug in.

In case there is no usable IDE found on the target computer the setup process offers the automatic installation of an adapted Eclipse Juno for Mobile Developers package. This platform is pre-configured for mobile development and contains already the required MTJ plugin so that there are no further steps required. Please note that the Eclipse installation is not done via the Windows installer and therefore the installed files must be removed manually for a deinstallation. The setup process offers the possibility to enter the target path for the eclipse installation which is by default the all users profile.

10.2.1 Installing the "Mobile Tools for Java" Plugin

The WTK requires an adapted MTJ plugin for Eclipse. Gemalto M2M provides a corresponding plugin for Eclipse Juno and later. Please note that the MTJ plugin available via the Eclipse plugin manager in earlier Eclipse releases is not fully compatible to the BGS5 WTK - its usage is not recommended.

If there is already an appropriate Eclipse installation on the target computer the required MTJ plugin provided on the installation medium can be installed manually for usage of the Cinterion WTK. To install the plugin select the menu item "Help" -> "Install New Software…". Then simply drag the file "Contribution\org.eclipse.mtj.update-site.zip" into the opened window. The table in the Window will then show the content of the provided plugin archive. Select the complete content, click "Next" and follow the required steps.

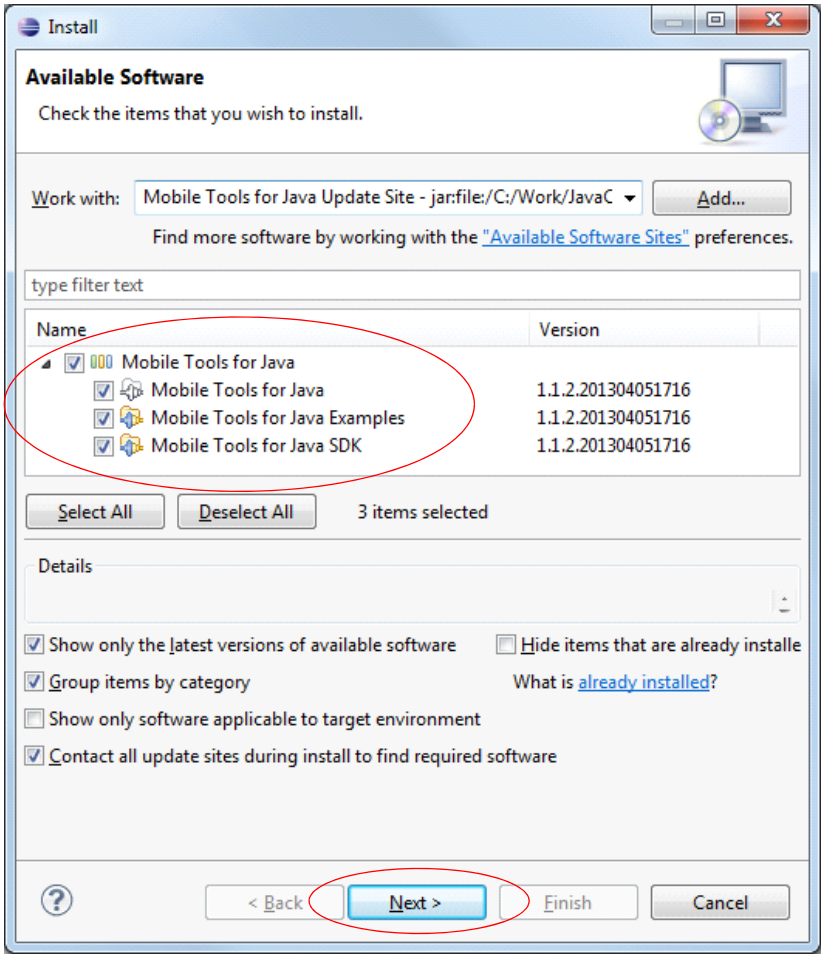

Figure 22: Installing the "Mobile Tools for Java" plugin

10.2.2 Integrating Cinterion WTK Manually

The integration of the Cinterion WTK into an appropriate Eclipse installation with added MTJ plugin can be done by the setup program automatically during first installation or afterwards in maintenance mode. Therefore it can be done manually as is described in this section. To add the WTK open the Eclipse menu item "Window" -> "Preferences". In the following window expand the preference list item "Java ME" and select the sub list item "Device Management". Then a list with the already installed Java ME devices is shown that is empty in the following example. For a manual installation select the button "Manual Install…".

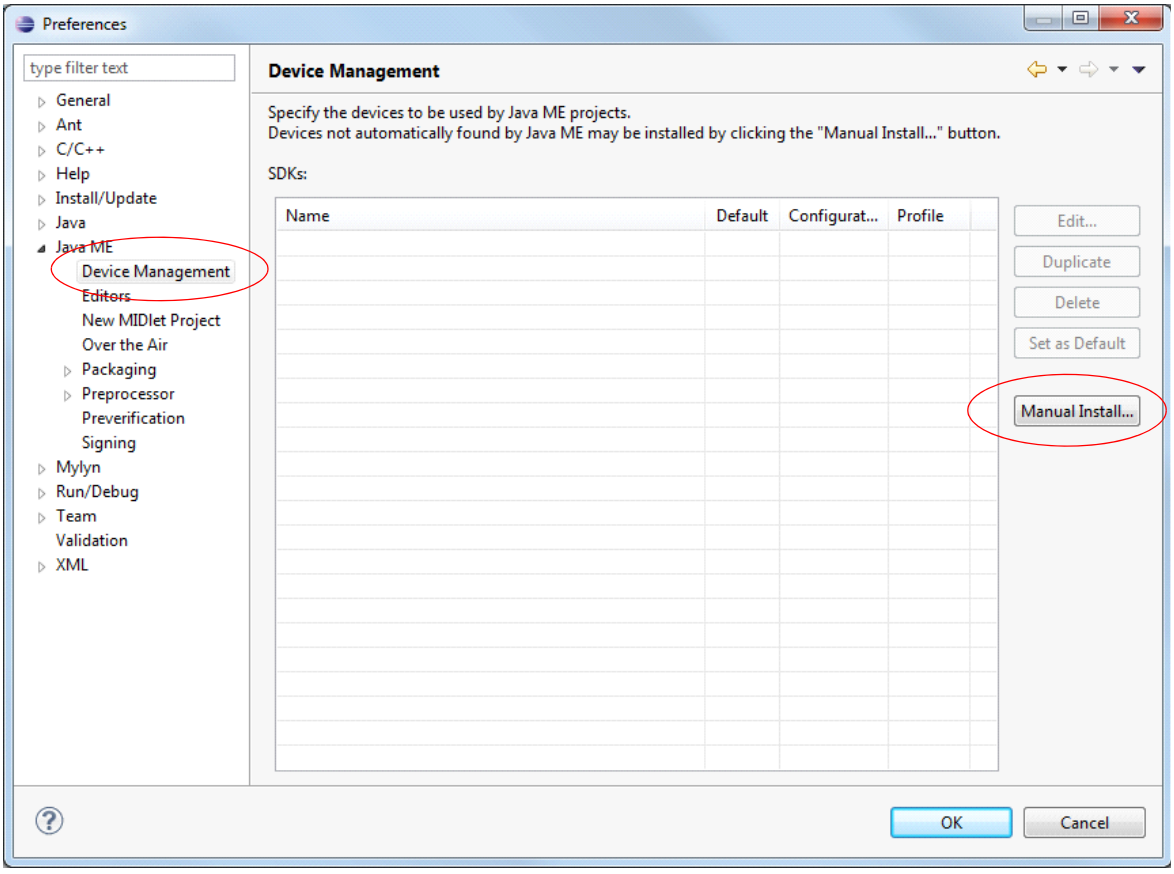

Figure 23: Integrating Cinterion WTK manually - Select preference

In the following window select the Browse… button, navigate to the root directory of the Cinterion BGSx WTK or enter the path "Program Files\Cinterion\CMTK\BGS5\WTK" directly. After the path has been entered it is scanned for usable CLDC device configurations and the found Cinterion WTK is added to the list. Please note that during WTK integration the connected devices are queried. For this reason the module must be connected, switched on and the debug connection must be configured properly during the integration process. In case of Windows firewall notification all connection requests must be granted. If there were firewall notifications it is possible that the WTK detection fails and must be repeated after the connections have been allowed.

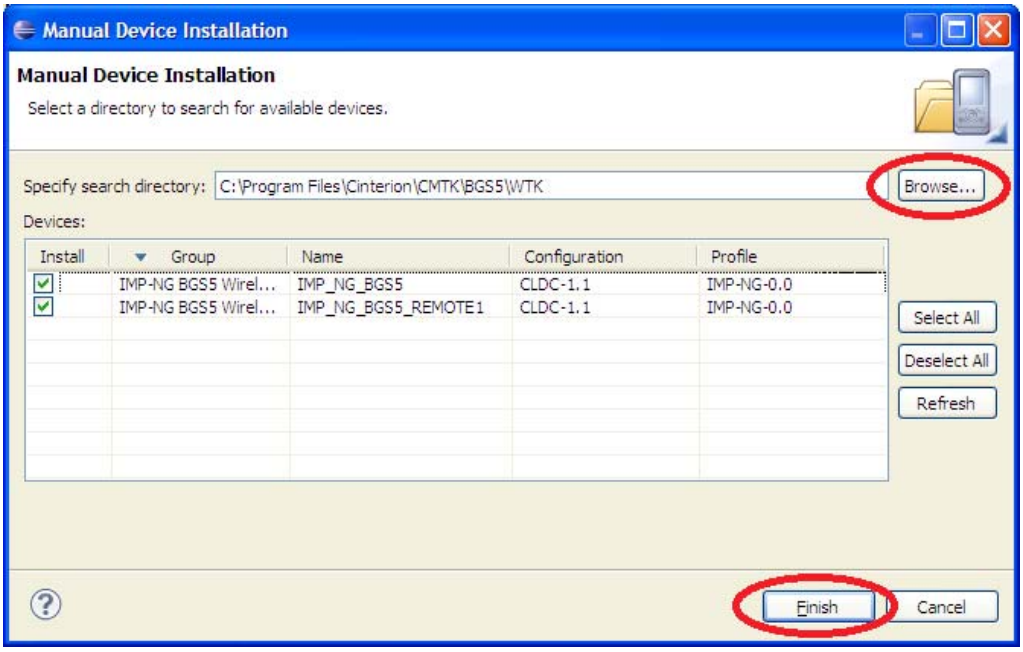

Figure 24: Integrating Cinterion WTK manually - Browse

Now the Cinterion WTK is available in the list of installed Java ME SDKs. Select the device "IMP_NG_BGS5_REMOTE1" and select "Set as default". To have the Cinterion WTK Java documentation directly available in your MIDlet projects the following additional steps are necessary. Select the new installed IMP_NG2_BGS5 device and click the Edit… button.

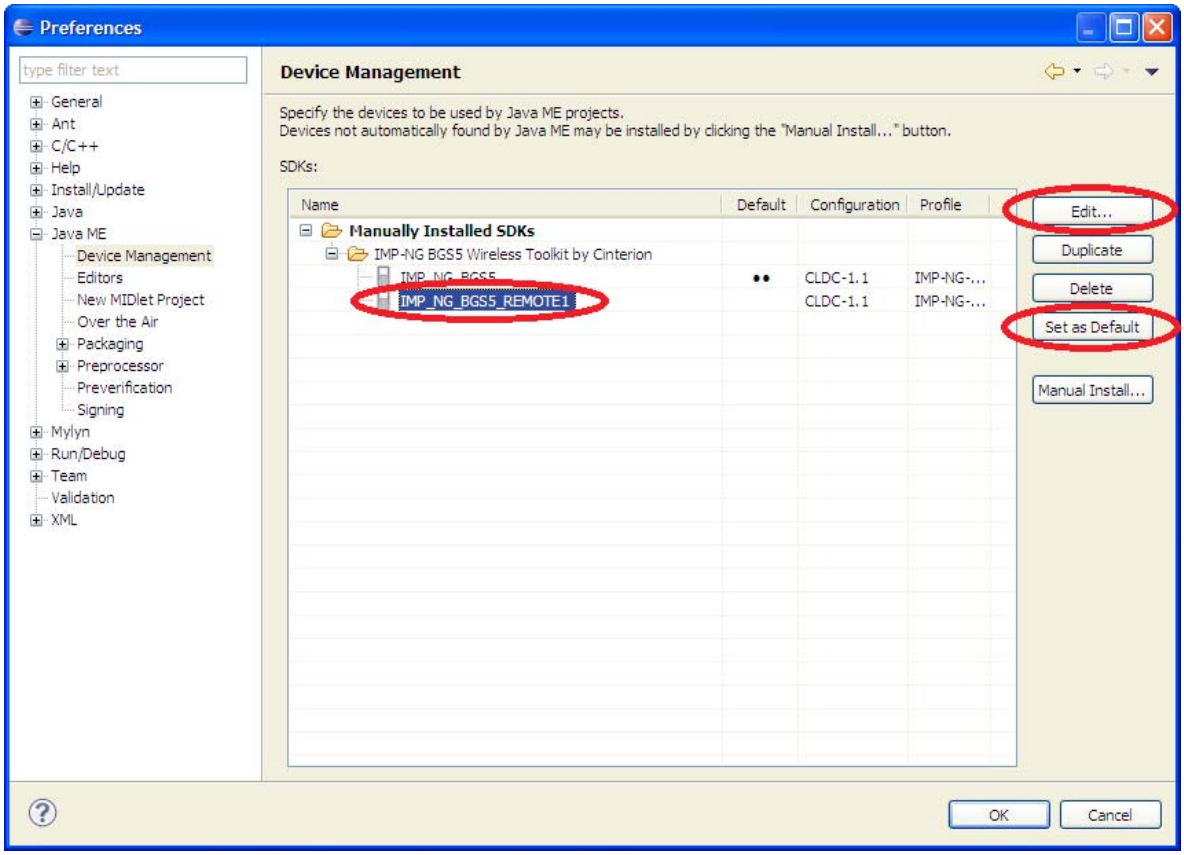

Figure 25: Integrating Cinterion WTK manually - Edit

In the following window select the Libraries tab and click into the Javadoc column for each jar library where the corresponding documentation was not added automatically. Once the table field is activated, a small button with ellipsis appears that allows browsing to the WTK documentation folder.

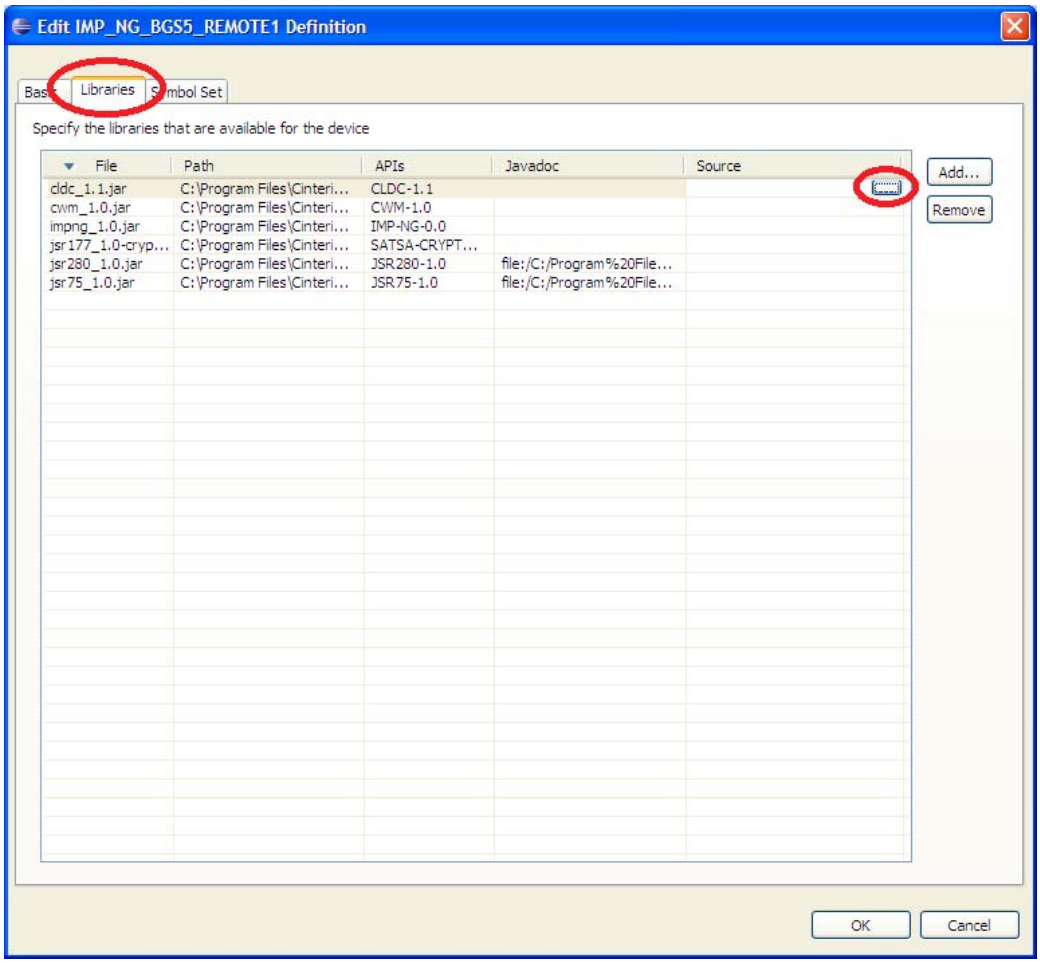

Figure 26: Integrating Cinterion WTK manually - Library

In the following window select "External location" and navigate to the documentation folder that is "Program Files\Cinterion\CMTK\BGS5\WTK\doc" after clicking the "External Folder…" button. If there exist a subdirectory with a name corresponding to the selected library file select this documentation. In all other cases select the common "html_impng" documentation tree. The OK button then adds the Cinterion WTK documentation to the new Java ME device.

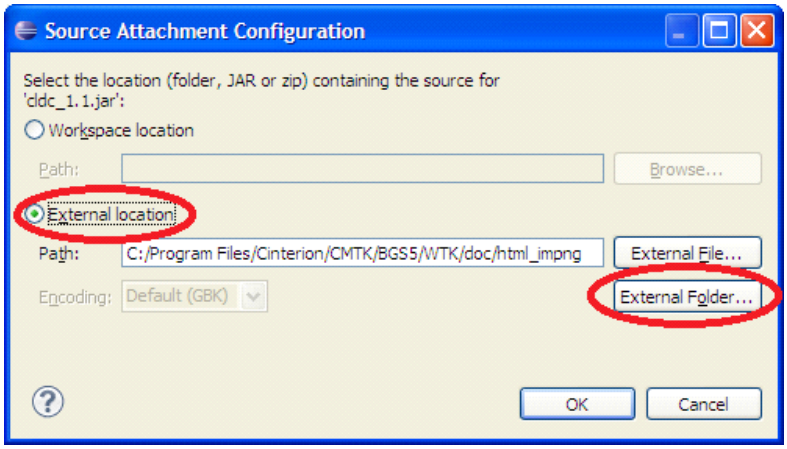

Figure 27: Integrating Cinterion WTK manually - Add WTK documentation

10.2.3 Import the provided WTK Samples

The Cinterion WTK provides the existing samples in an Eclipse project format as well. To import them into an Eclipse workspace select the Eclipse menu item "File" -> "Import…" In the following window expand the list item General and select the sub list item "Existing Projects into Workspace". After clicking the Next button it is possible to navigate to the folder containing the Cinterion WTK Eclipse samples called "Documents and Settings\All Users\Cinterion\BGS5 WTK Examples\EclipseSamples" under Windows XP or "Users\Public\ Cinterion\BGS5 WTK Examples\EclipseSamples" under Windows Vista and above.

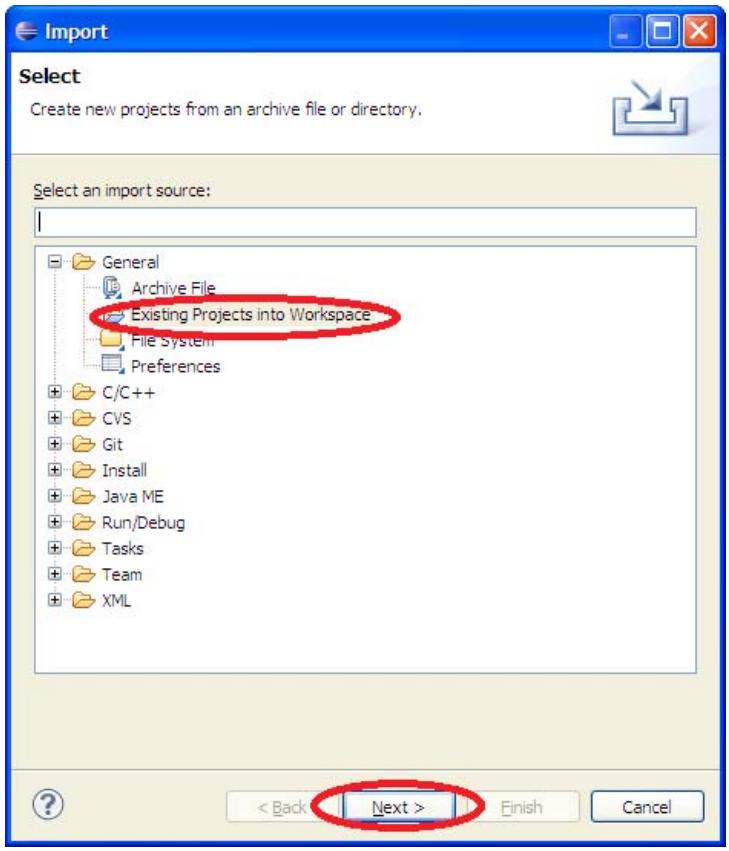

Figure 28: Import the provided WTK Samples - Select

Once the root directory of the samples has been entered the existing projects are added to the list and can be separately selected for import. Checking or unchecking the checkbox "Copy projects into workspace" controls whether the samples are copied into the current Eclipse workspace or kept where they are. After clicking the Finish button the selected samples are available for editing and debugging.

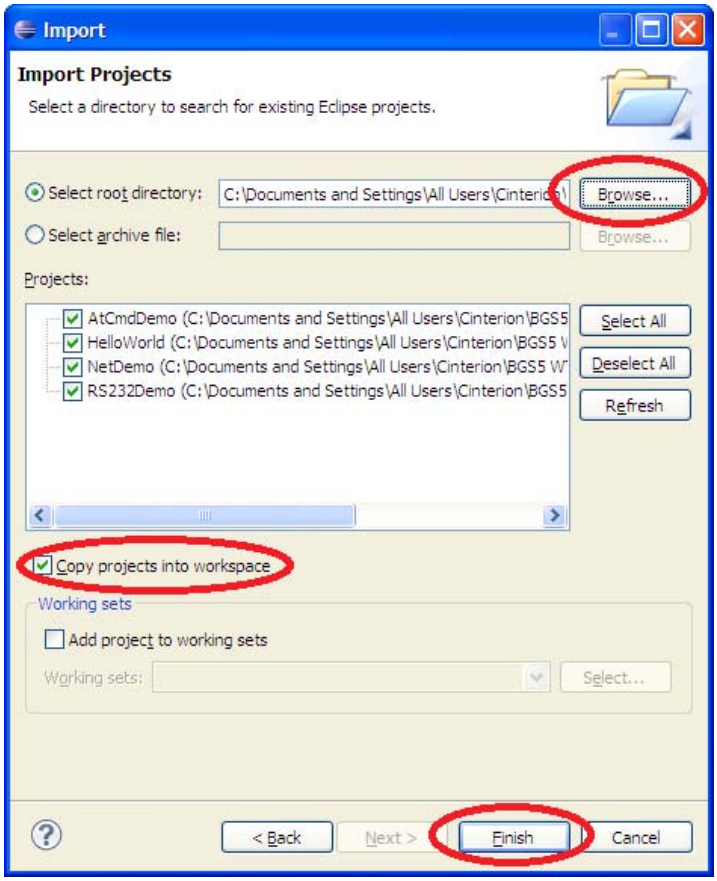

Figure 29: Import the provided WTK Samples - Copy

10.2.4 Creating a new MIDlet

To create a new MIDlet select the menu item File-->New-->Project…. In the following window expand the list item "Java ME" and select the sub list item "MIDlet Project".

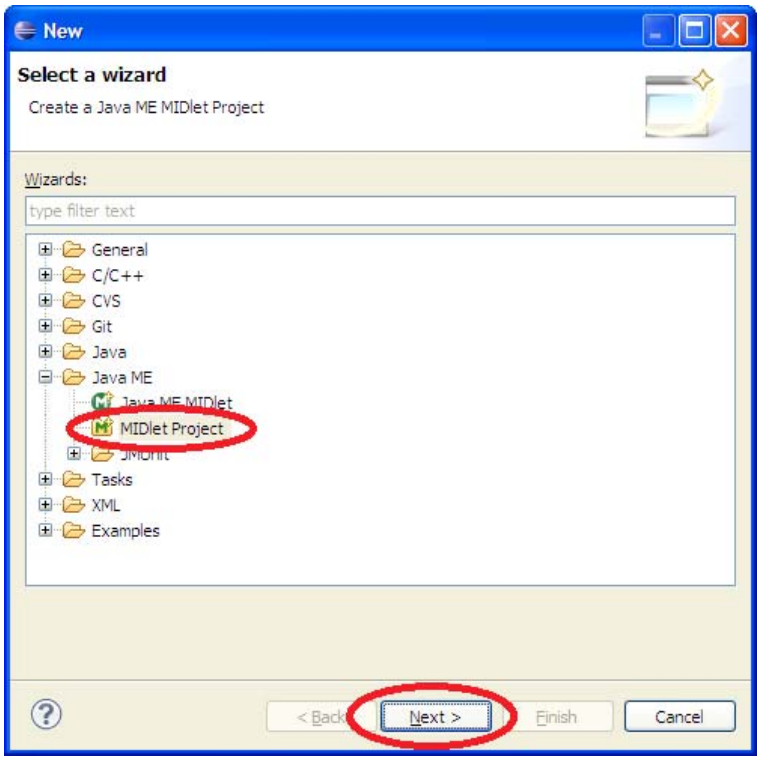

Figure 30: Creating a new MIDlet - Select wizard

After clicking the Next button a wizard for MIDlet creation appears. Specify the appropriate options as required for the new project. Ensure that only the "IMP_NG_BGS5" configuration is selected in the configuration list as active.

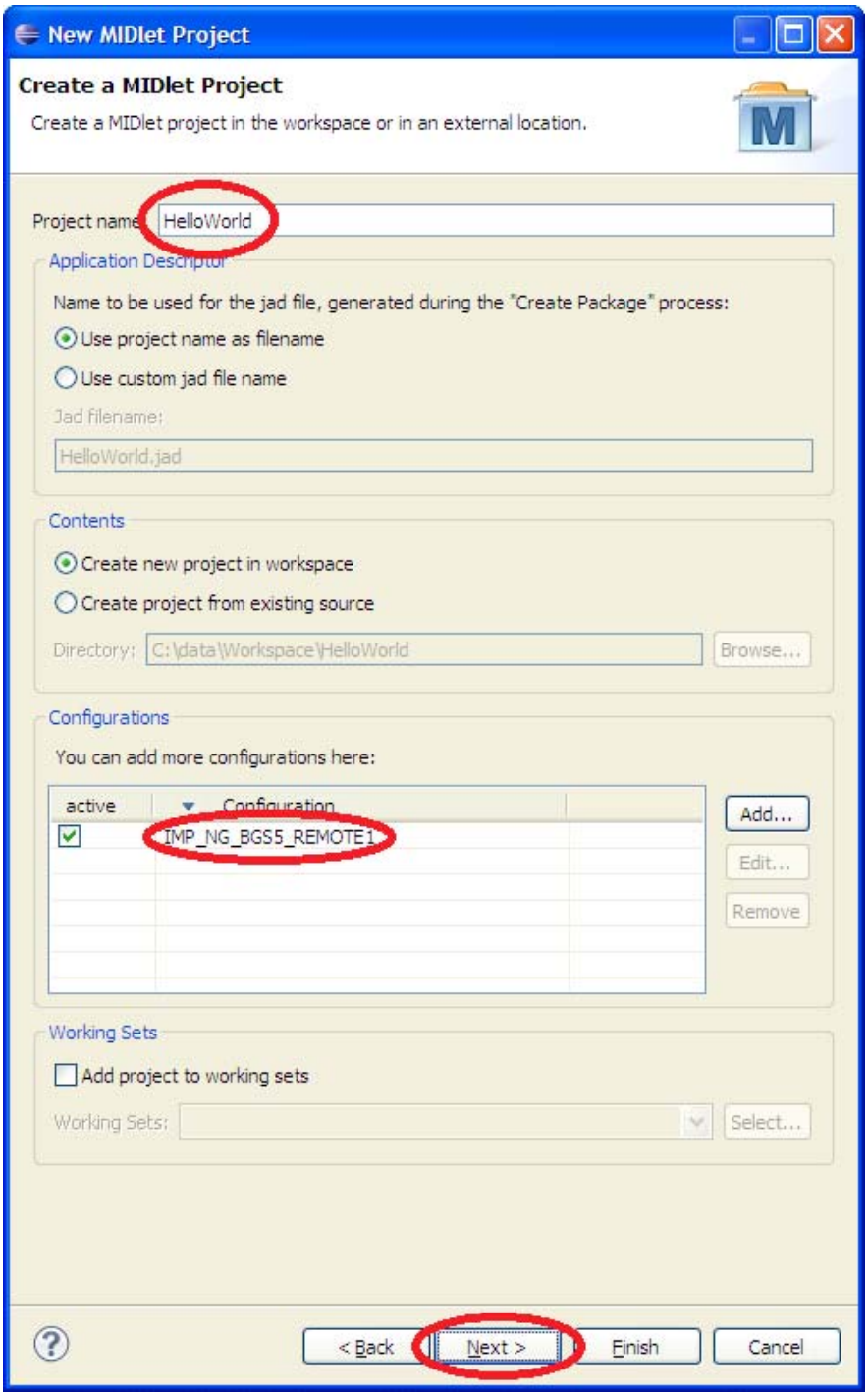

Figure 31: Creating a new MIDlet - Create project

After clicking the Next button a window for configuring the MIDlet project content opens. Ensure that the "Microedition Configuration" is set to "Connected Limited Device Configuration (1.1)" and the "Microedition Profile" is set to "Information Module Profile (NG)". After all necessary modifications have been done the project can be created by clicking the Finish button.

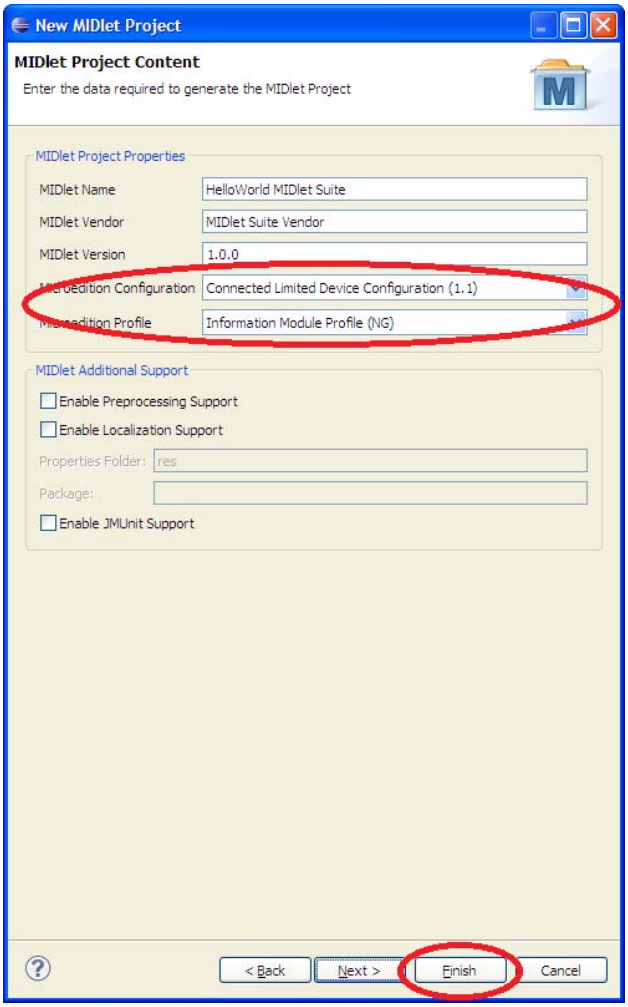

Figure 32: Creating a new MIDlet - Configure project

Now a newly created MIDlet project is available in the workspace.

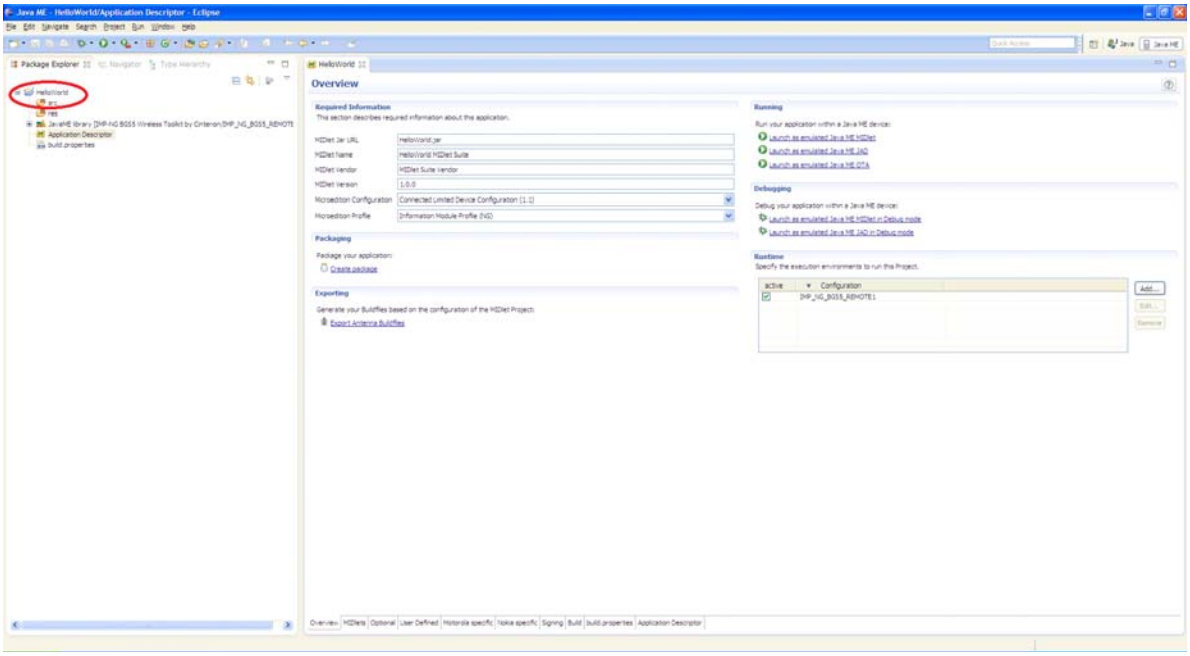

Figure 33: Creating a new MIDlet - Project overview

To ensure the correct Java compiler setting right-click on the newly created project in the package tree explorer on the left side of the Eclipse window and select "Properties". In the opened project properties window select the item "Java Compiler" in the list on the left side. Ensure that the check box "Enable project specific settings" is selected and the "Compiler compliance level" is set to 1.3 or 1.4.

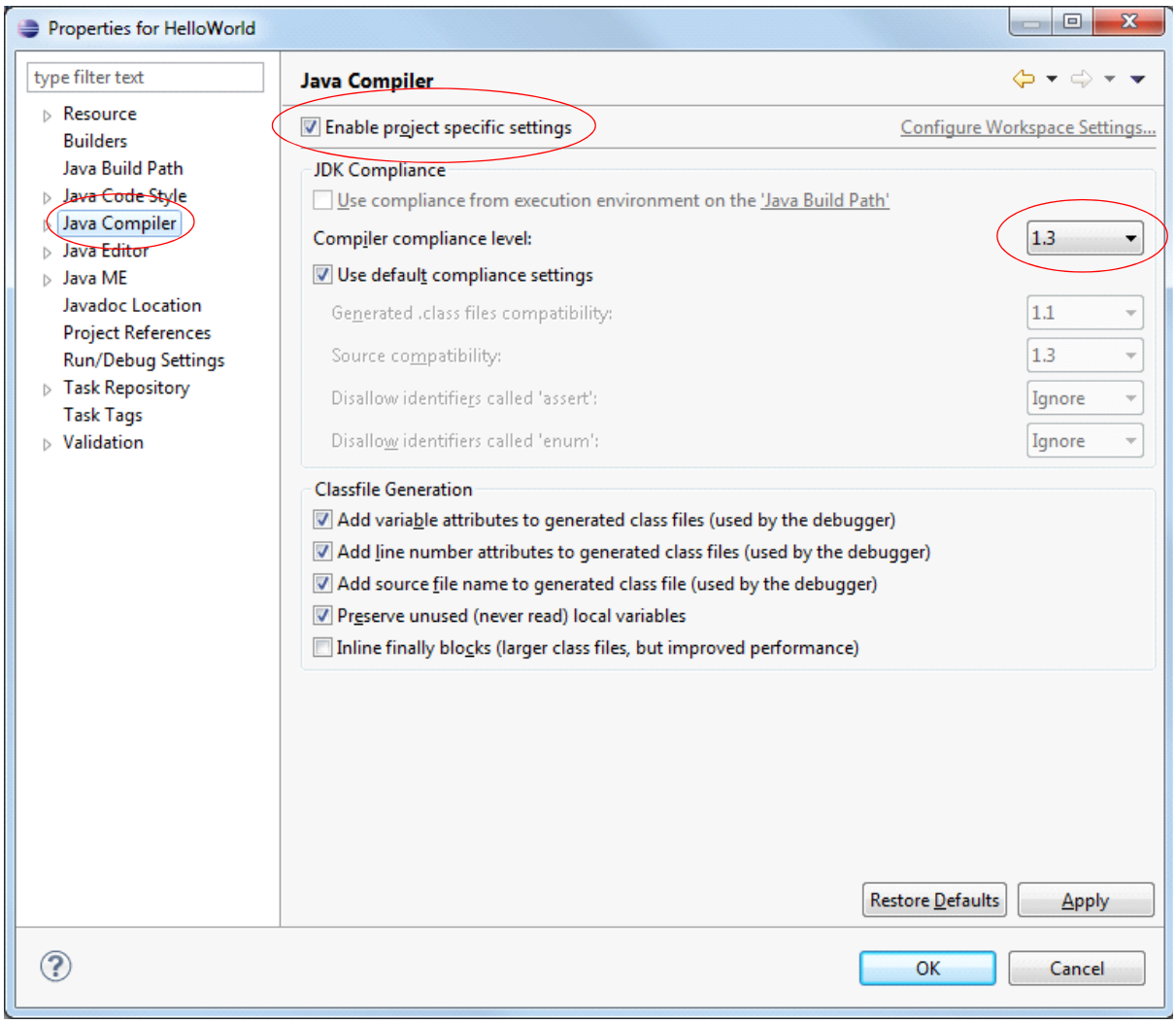

Figure 34: Creating a new MIDlet - Configure compliance level

If the ATCommand class is to be used the corresponding library must be added to the project. Right click into the project tree and select "Build Path -> Add External Archives…". Then navigate to the folder "Program Files\Cinterion\CMTK\BGS5\WTK\resources" and select the library file "cwmlib_1.0.jar". If the Eclipse integration has been done by the BGS5 setup process a classpath variable was created named "BGS5_WTK_CWMLIB" pointing to that external archive for convenience. In that case the archive can be added by right-clicking into the project tree and selecting "Build Path -> Configure Build Path…". In the following dialog select the property tab "Libraries", click on the button "Add Variable…" and select the variable "BGS5_WTK_CWMLIB" in the following window. The provided example "AtCmdDemo" uses this class path variable.

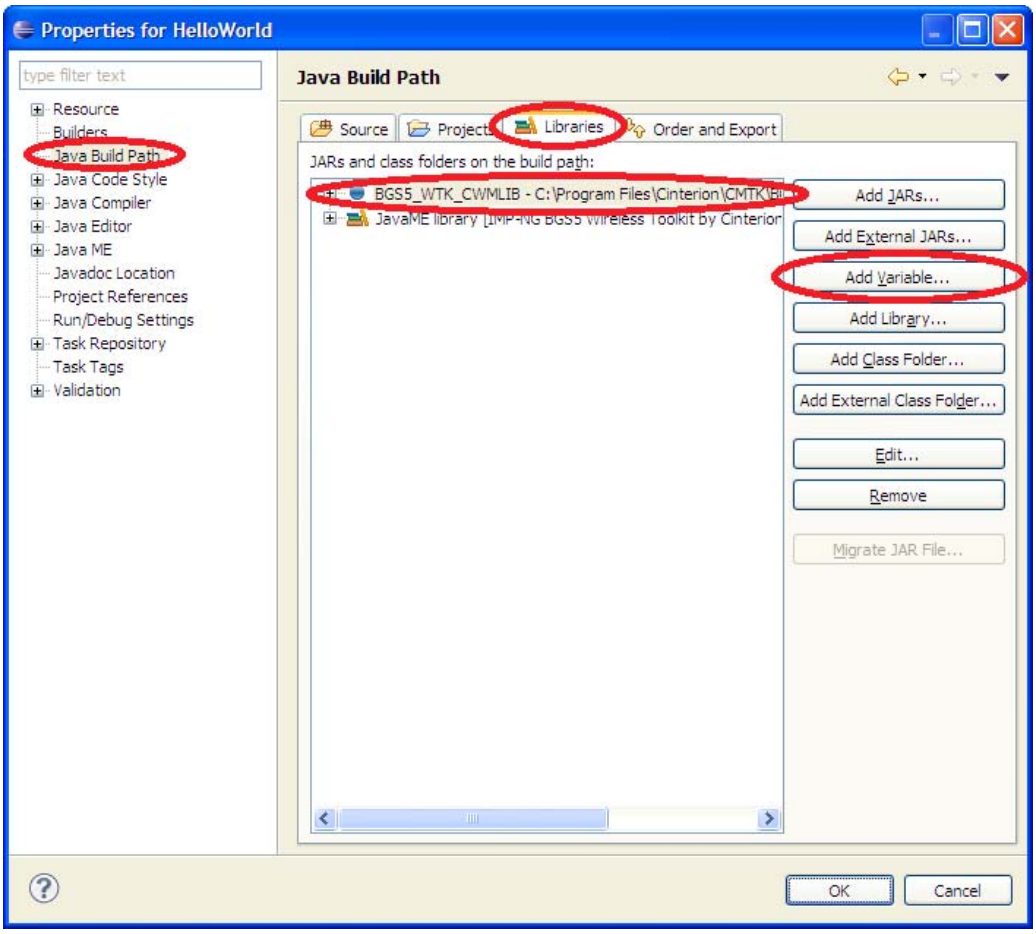

Figure 35: Add cwmlib to project

To include the content of the cwmlib to the project package in the build path configuration the cwmlib_1.0.jar or alternatively the variable "BGS5_WTK_CWMLIB" must be selected on the property tab "Order and Export".

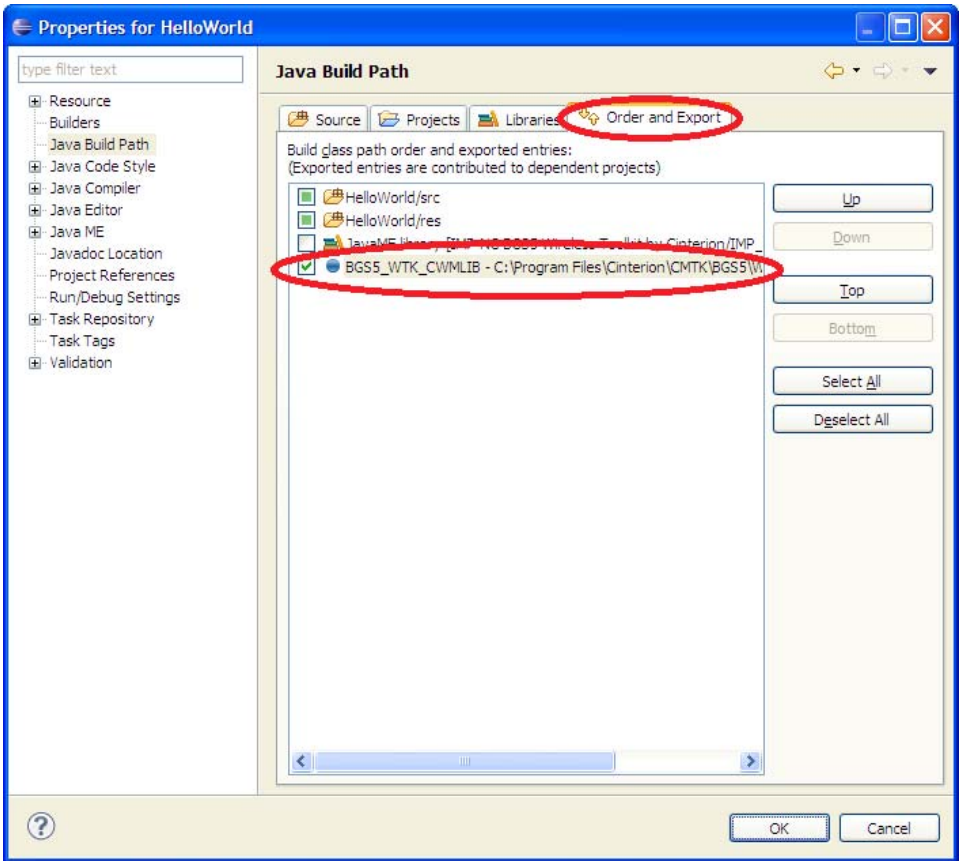

Figure 36: Include cwmlib into project package

To create the frame for a MIDlet program right-click into the project tree and select New-->"Java ME MIDlet". In the following wizard enter the name for the MIDlet and click the Finish button.

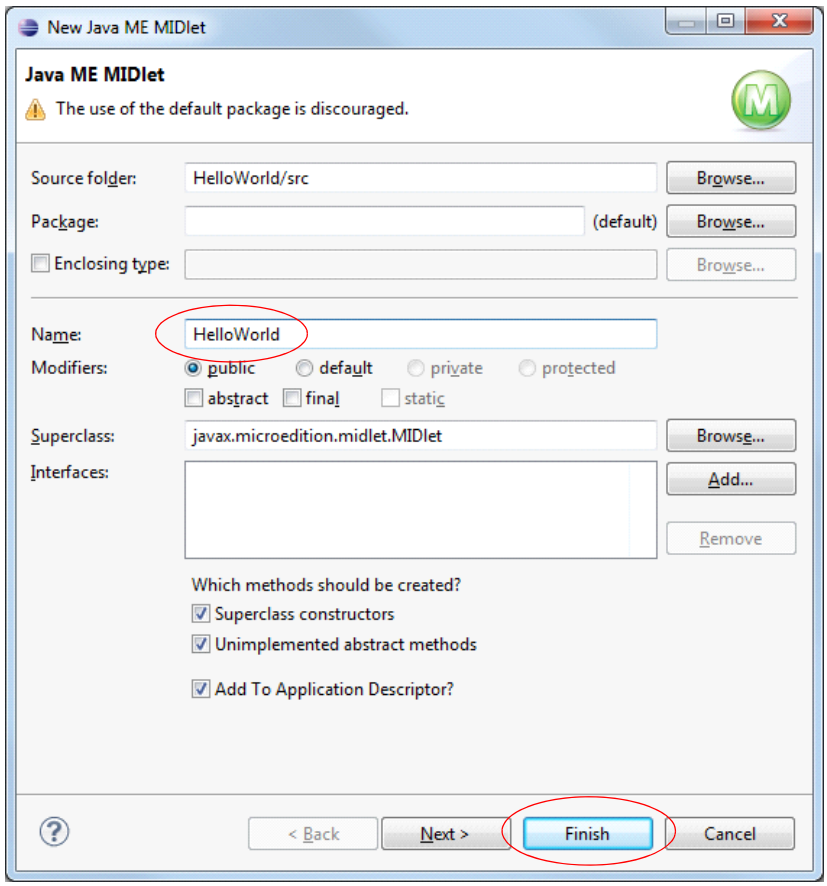

Figure 37: Creating a new MIDlet - Program name

Now, a corresponding Java file with the source of the basic MIDlet interface is created inside the source tree of the project. The only thing that remains to be done is to add a call of "notifyDestroyed()" to the "destroyApp()" method.

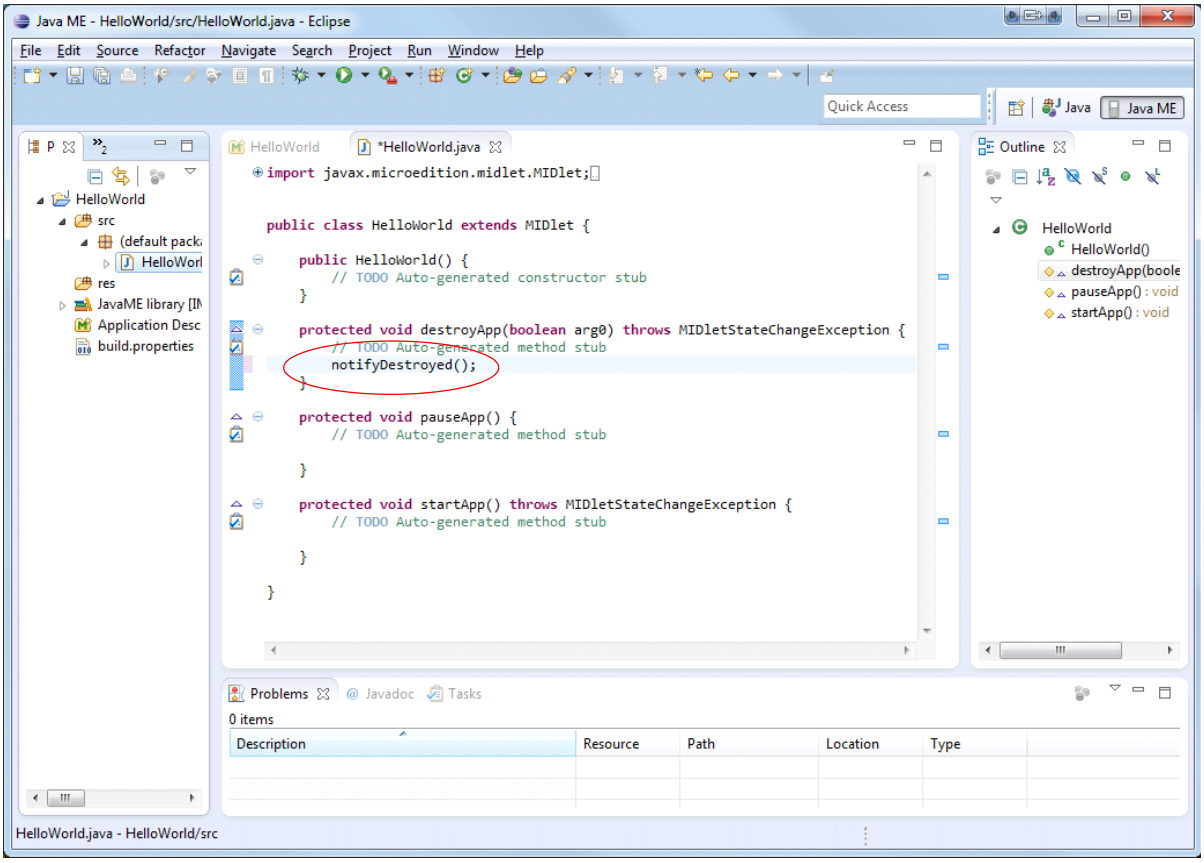

Figure 38: Creating a new MIDlet - HelloWorld.java

Finally, the actual functionality can be implemented and the project can be started and debugged inside the module as with other Eclipse projects. Please note that you must allow Eclipse and the debug agent to pass the firewall and configure the debug IP connection as a home network to be able to execute and debug MIDlets inside the device.

10.2.5 Using Eclipse Workspaces

Eclipse follows the concept of different "workspaces". Basically, these are folders on the hard disk used by Eclipse to store settings and new projects - just as an operating system might store documents and settings of different users in different directory structures. Eclipse can manage different workspaces allowing for an easy switching between different development environments requiring different IDE setups. If not disabled, Eclipse asks before each start for the workspace to be used.

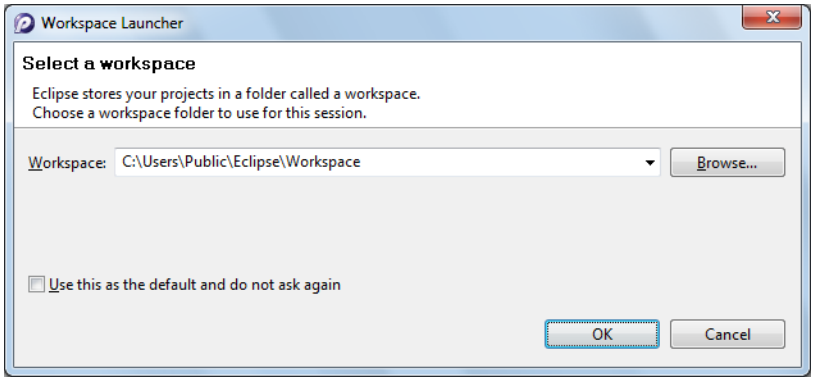

Figure 39: Using Eclipse workspaces

If Eclipse is installed via the Cinterion WTK setup a default workspace is created automatically with all required settings. But because Eclipse stores all settings inside the workspaces a new workspace which has been created via Eclipse it doesn't contain the necessary settings for using the Cinterion WTK. In this case the required settings can be added manually as described in Section 10.2.2. Alternatively the Cinterion WTK setup can be executed in maintenance mode. After scanning for supported IDEs the required settings will be added to all available workspaces of a found Eclipse installation automatically.

10.3 Using NetBeans for Java Development

If an appropriate NetBeans installation (as of version 6.7 or higher) is found on the target computer, the Cinterion WTK is integrated automatically into the NetBeans IDE by the setup process. NetBeans 7.3 is distributed as part of the Gemalto M2M installation CD and may be installed from there, if required. The package is located under the "Contribution" directory. It is recommended to use NetBeans 7.3. On later or earlier versions the version of the Java client implemented in the module might not be supported properly.

For usage of the Cinterion WTK inside NetBeans the "Mobility" plugin must be installed. Refer to Section 10.3.1 for information on how to install this plugin.

10.3.1 Installing the "Mobility" Plugin

The required "Mobility" plugin can be easily installed via the NetBeans plugin manager. This can be called via the menu item "Tools" --> "Plugins". Inside the plugin manager activate the tab "Available Plugins" and inside the plugin list select the item "Mobility" in the category "Java ME". To install the selected plugin click the Install button and follow the given steps.

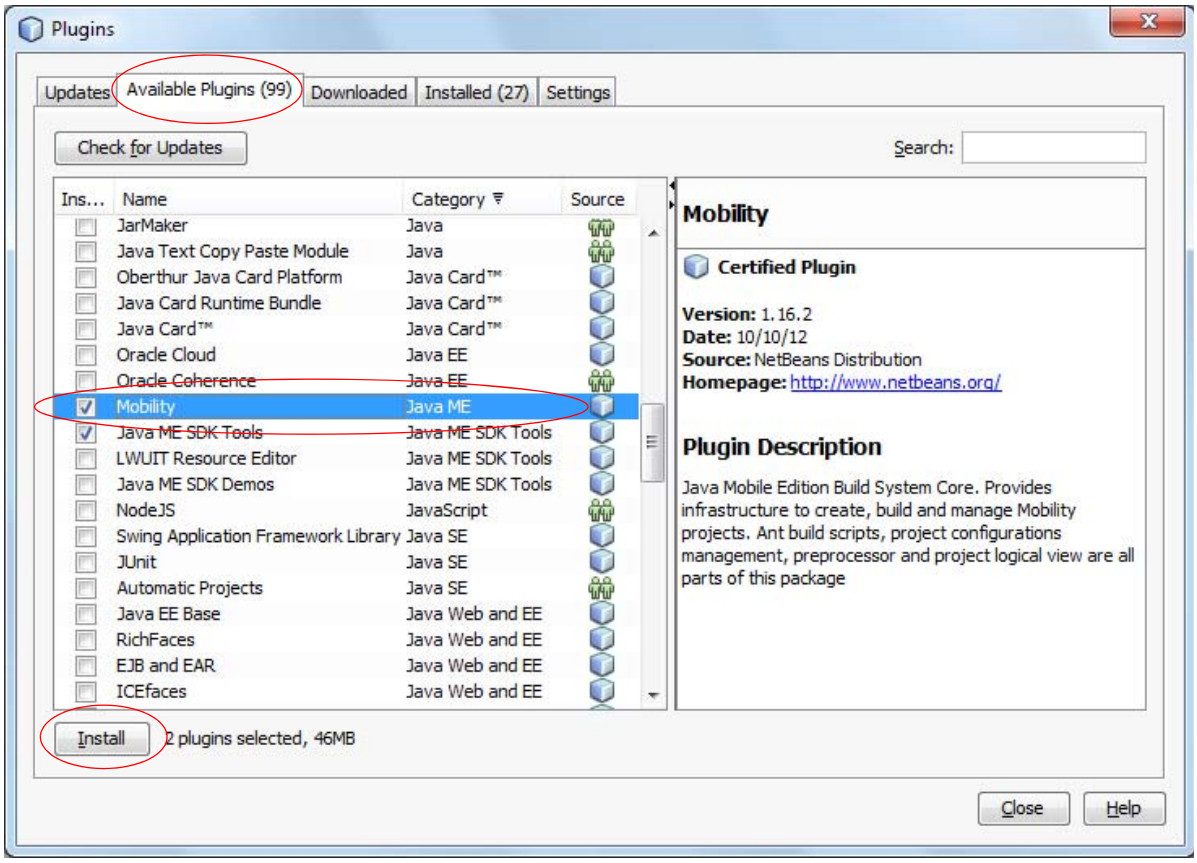

Figure 40: Installing "Mobility" plugin

Please note that the BGSx modules are supporting on device profiling. This feature provides the possibility to log and analyze the consumed CPU time on a per function base. For using this feature the plugin "Java ME SDK Tools" from the category "Java ME SDK Tools" must be installed as well. On NetBeans versions different to 7.3 the "Java ME SDK Tools" might not be available via the NetBeans plugin manager. In this case the " Java ME SDK 3.2 " must be downloaded from the following URL and installed manually:
http://www.oracle.com/technetwork/java/javame/javamobile/download/sdk/default-303768.html

10.3.2 Integrating Cinterion WTK Manually

The integration of the Cinterion WTK into an appropriate NetBeans installation with added Mobility plugin can be done by the setup program automatically during first installation or afterwards in maintenance mode. Nevertheless it can be done although manually which is described in this chapter. To add the WTK open the NetBeans menu item "Tools" - "Java Platforms" and click the button "Add Platform…" in the following window.

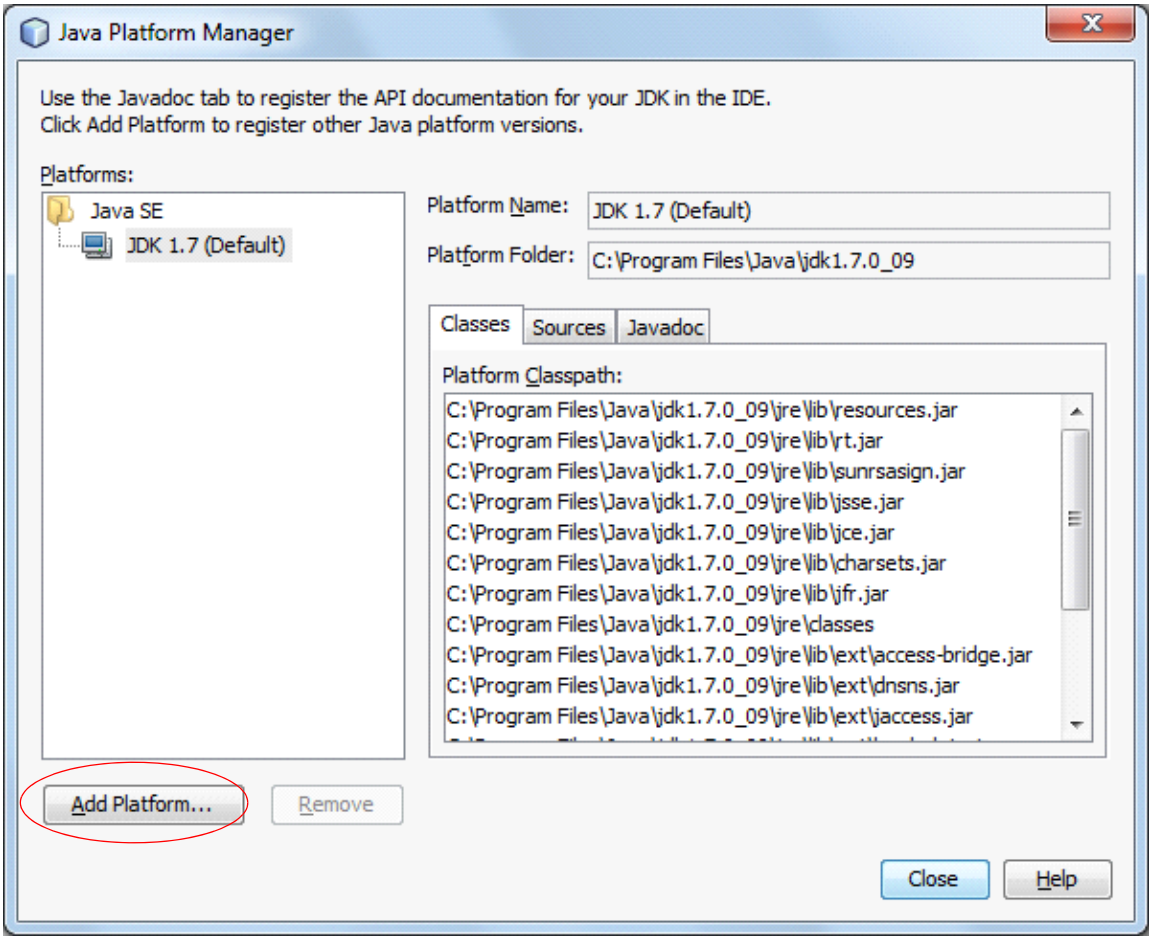

Figure 41: Integrating Cinterion WTK manually - Select platform

In the following dialog choose the platform type "Java ME MIDP Platform Emulator" or "Java ME CLDC Platform Emulator" and click the Next button.

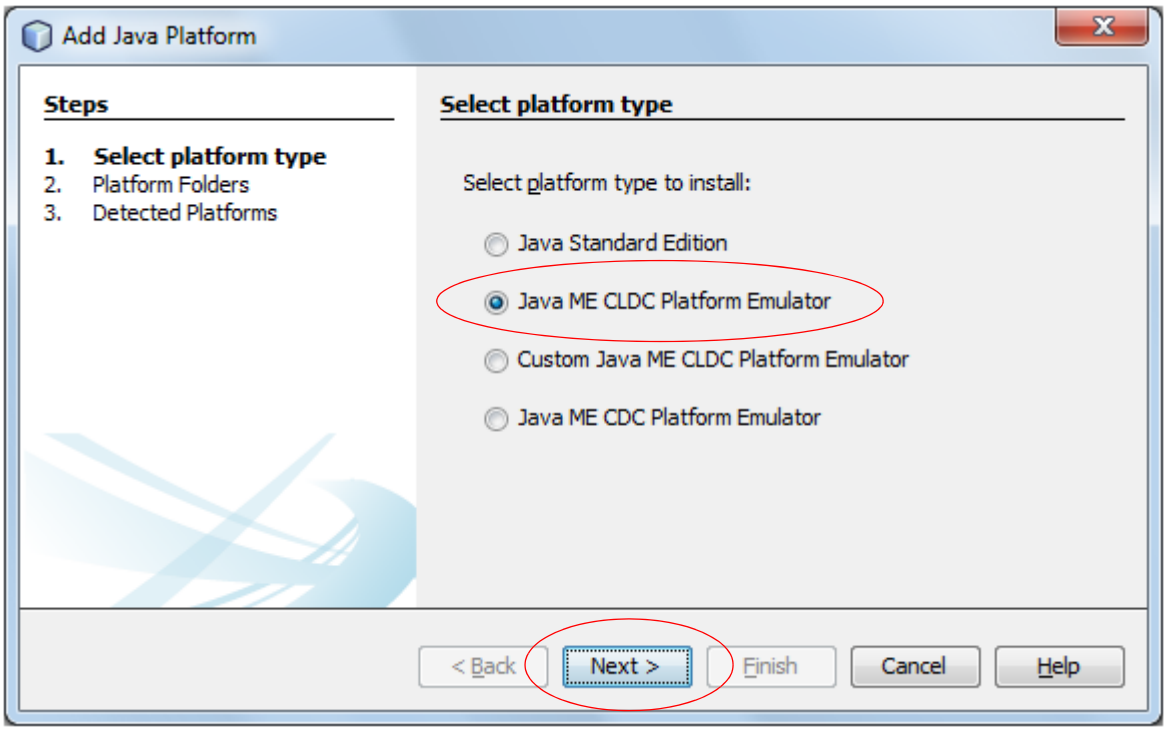

Figure 42: Integrating Cinterion WTK manually - Select type

In the file dialog browse to the root folder of the Cinterion WTK directory "Program Files\Cinterion\CMTK\BGS5\WTK". Now the Cinterion WTK is listed in the platform dialog. Please note that during WTK integration the connected devices are queried. For this reason the module must be connected, switched on and the debug connection must be configured properly during the integration process. In case of Windows firewall notification all connection requests must be granted. If there were firewall notifications it is possible that the WTK detection fails and must be repeated after the connections have been allowed.

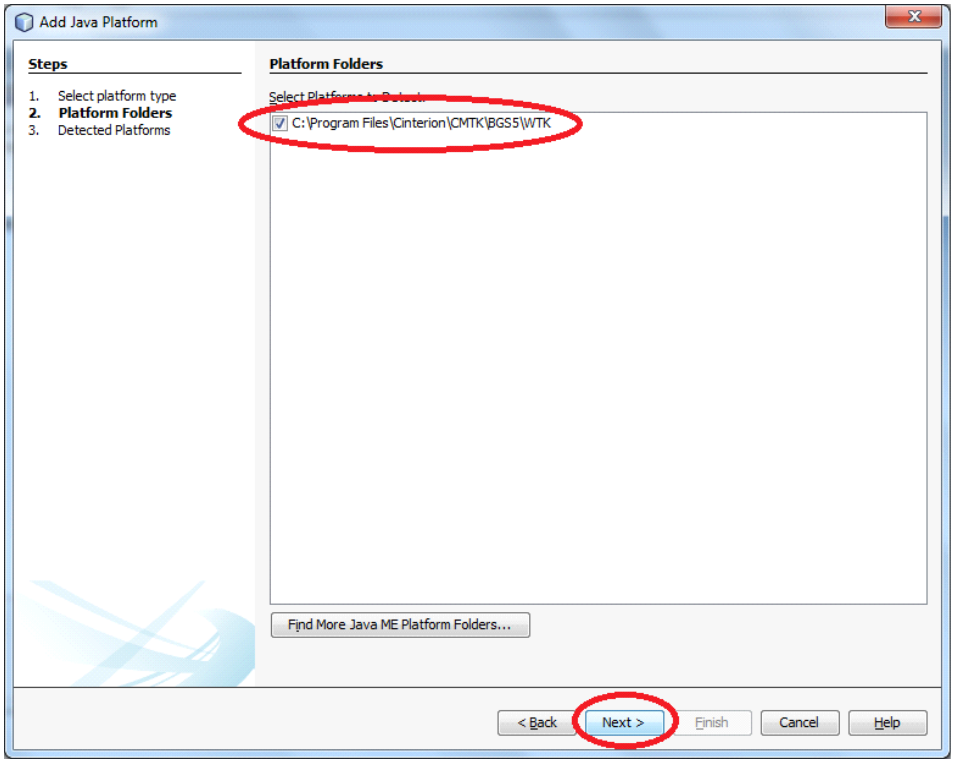

Figure 43: Integrating Cinterion WTK manually - Platform folder

After clicking the Next button the WTK interface is queried, the available emulator data displayed and the integration can be finished with the Finish button.

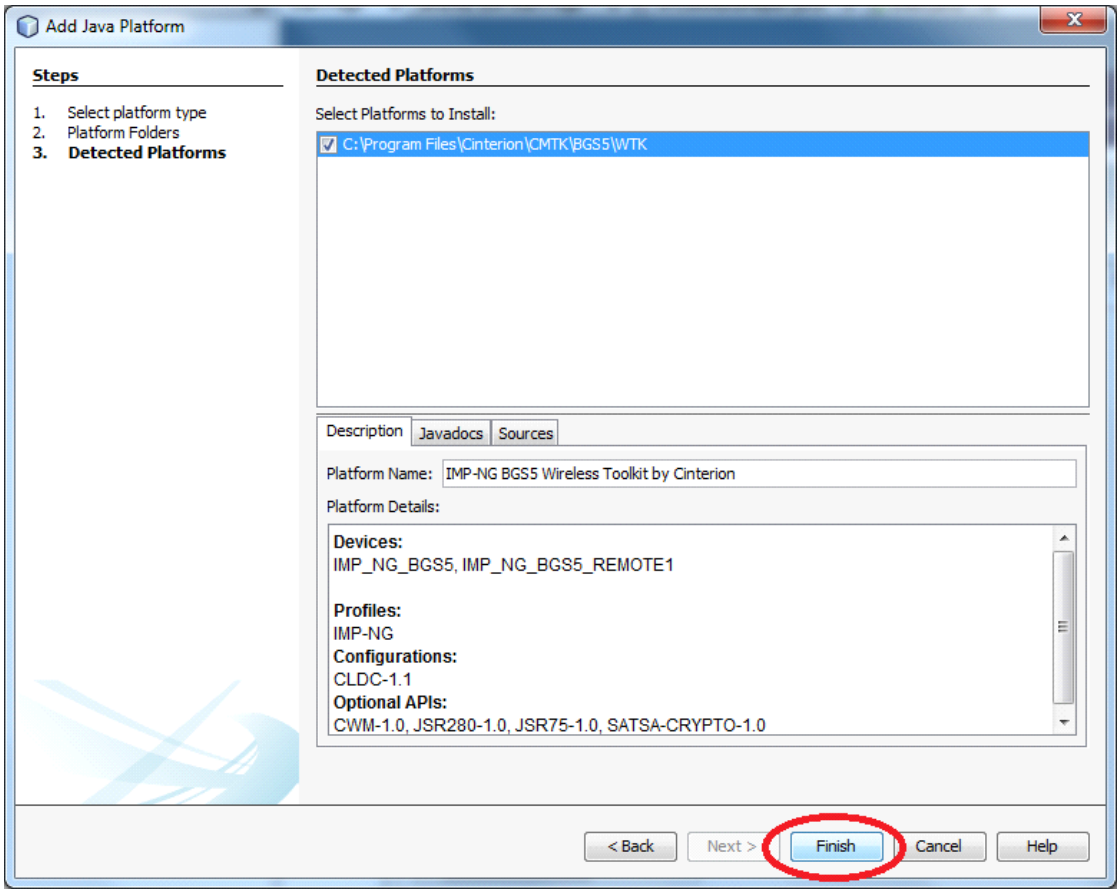

Figure 44: Integrating Cinterion WTK manually - Finish

10.3.3 Opening the Provided WTK Samples

The Cinterion WTK provides the existing samples in a NetBeans project format as well. They can simply be opened inside NetBeans via the menu item "File" --> "Open Project…" and navigating inside the open dialog to the sample location of either "Documents and Settings\All Users\Cinterion BGS5 WTK Examples\NetBeansSamples" under Windows XP or "Users\Public\Cinterion BGS5 WTK Examples\NetBeansSamples" under Windows Vista and above.

10.3.4 Creating a New MIDlet

To create a new MIDlet select the menu item "File" --> "New Project…". In the following window select the category "Java ME" and the project type "Mobile Application".

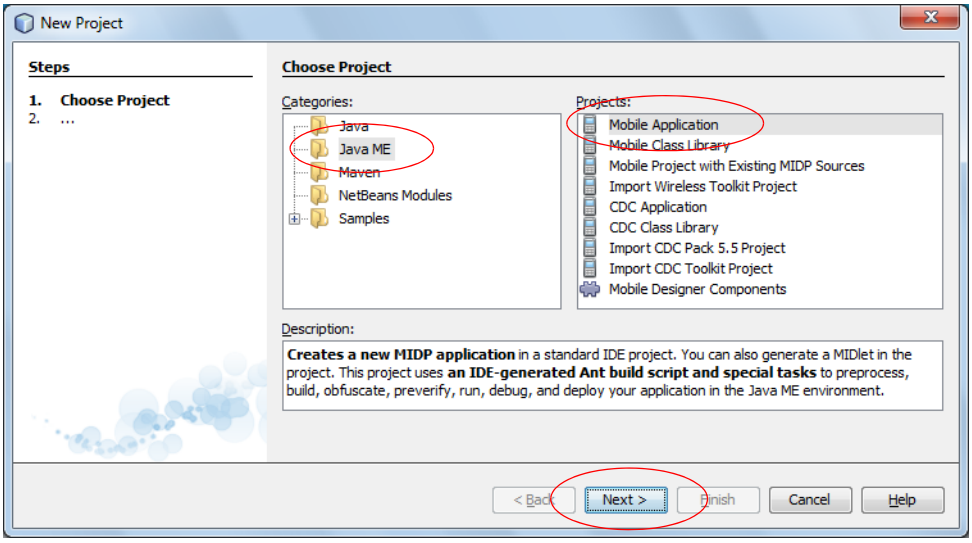

Figure 45: Creating a new MIDlet - Choose project

After clicking the Next button the project name and location can be entered. Depending on the employed NetBeans version please unselect the option "Create Hello MIDlet" if available, because the provided MIDlet template does not fit the IMP profile very well. If the option "Create Default Package and Main Executable Class" is available, it may remain selected causing the frame of an empty MIDlet being created automatically.

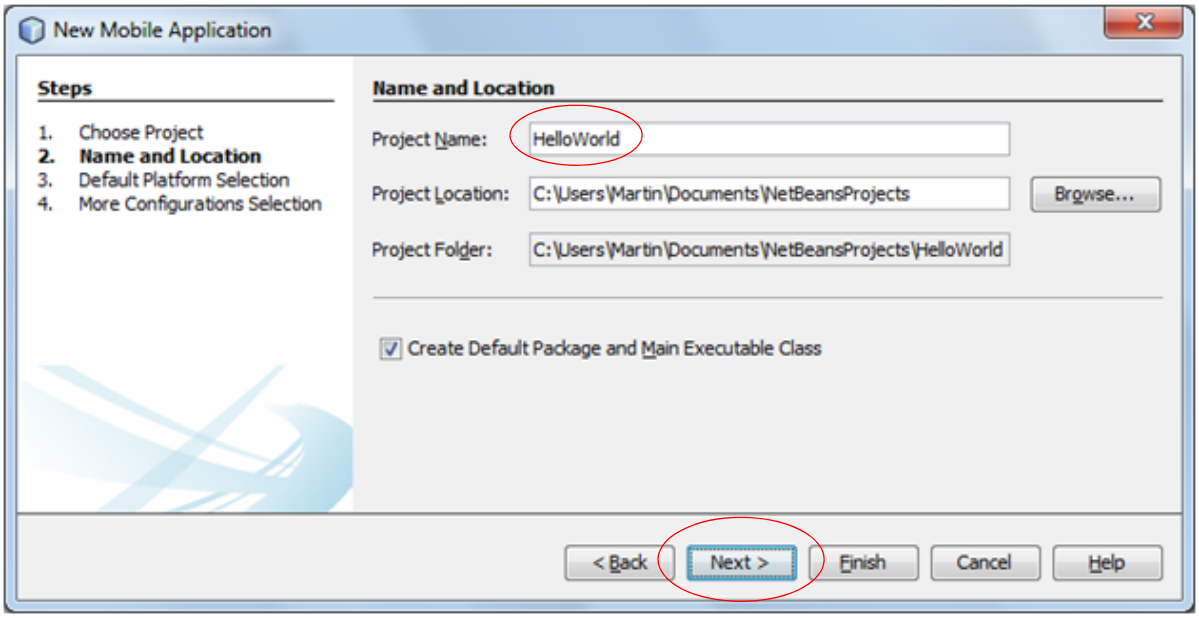

Figure 46: Creating a new MIDlet - Enter name and location

In the next dialog it must be ensured that the correct emulator platform "IMP-NG BGS5 Wireless Toolkit by Cinterion", the correct device "IMP_NG_BGS5_REMOTE1", the correct device configuration "CLDC-1.1" and the correct device profile "IMP-NG" are selected.

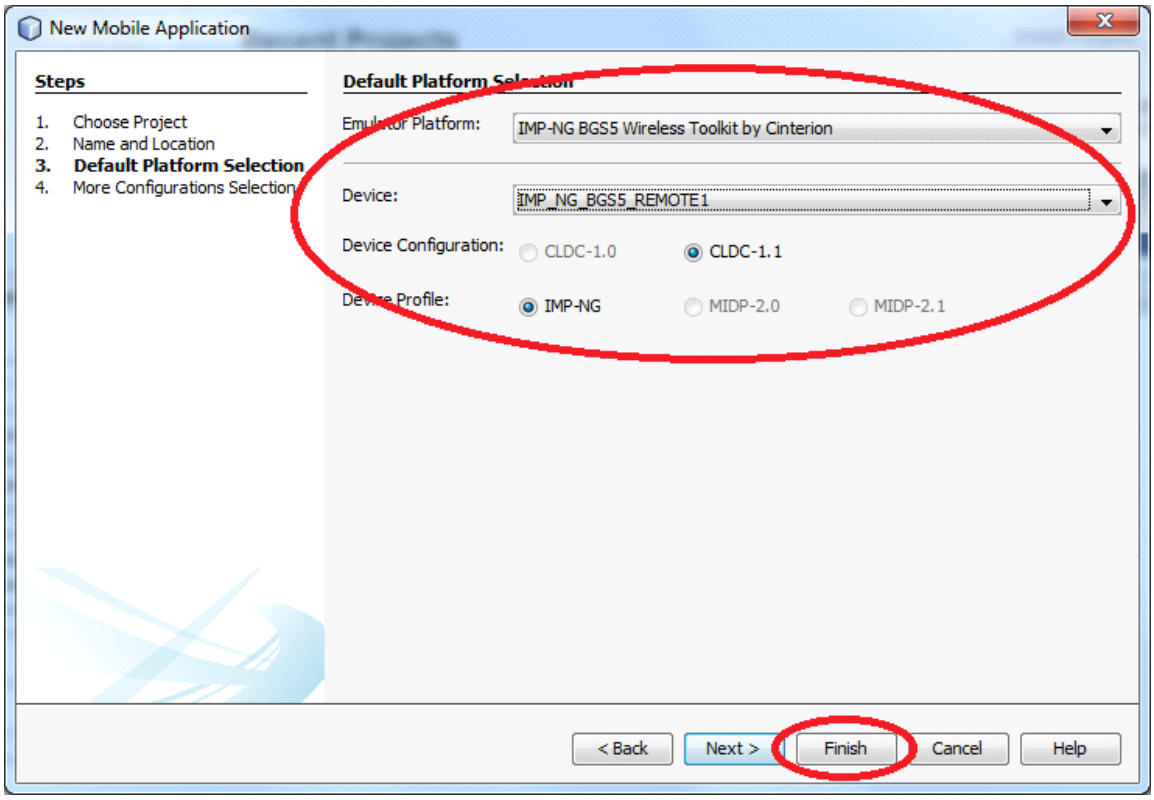

Figure 47: Creating a new MIDlet - Configure platform

After clicking the Finish button a newly created MIDlet project is available under the NetBeans project tree. If not already completed the frame of an empty MIDlet program can be created by right-clicking into the project tree and selecting "New" --> "MIDlet…". In the following wizard enter the name for the MIDlet and the MIDlet class and click the Finish button.

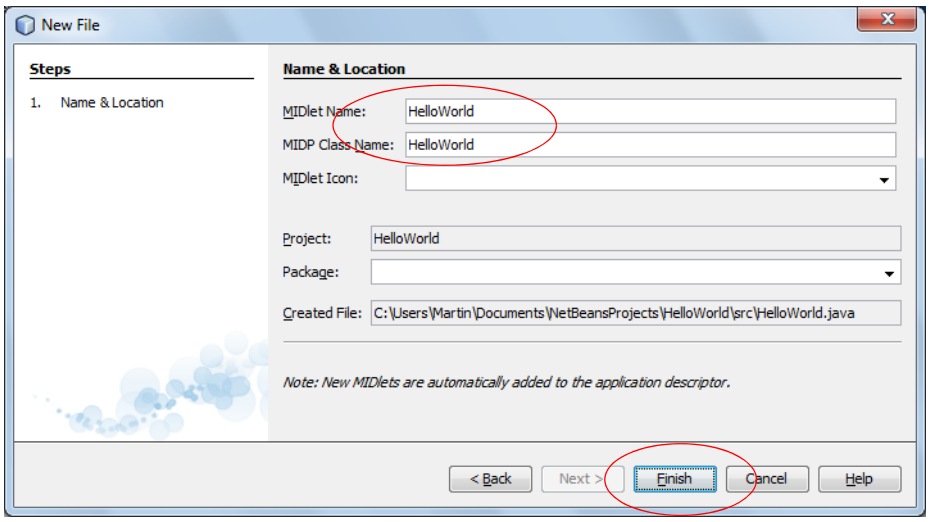

Figure 48: Creating a new MIDlet - Program name

Now, a corresponding Java file with the source of the basic MIDlet interface is created inside the source tree of the project. The only things that remains to be done is to add a call of "notifyDestroyed()" to the "destroyApp()" method and if required a class constructor.

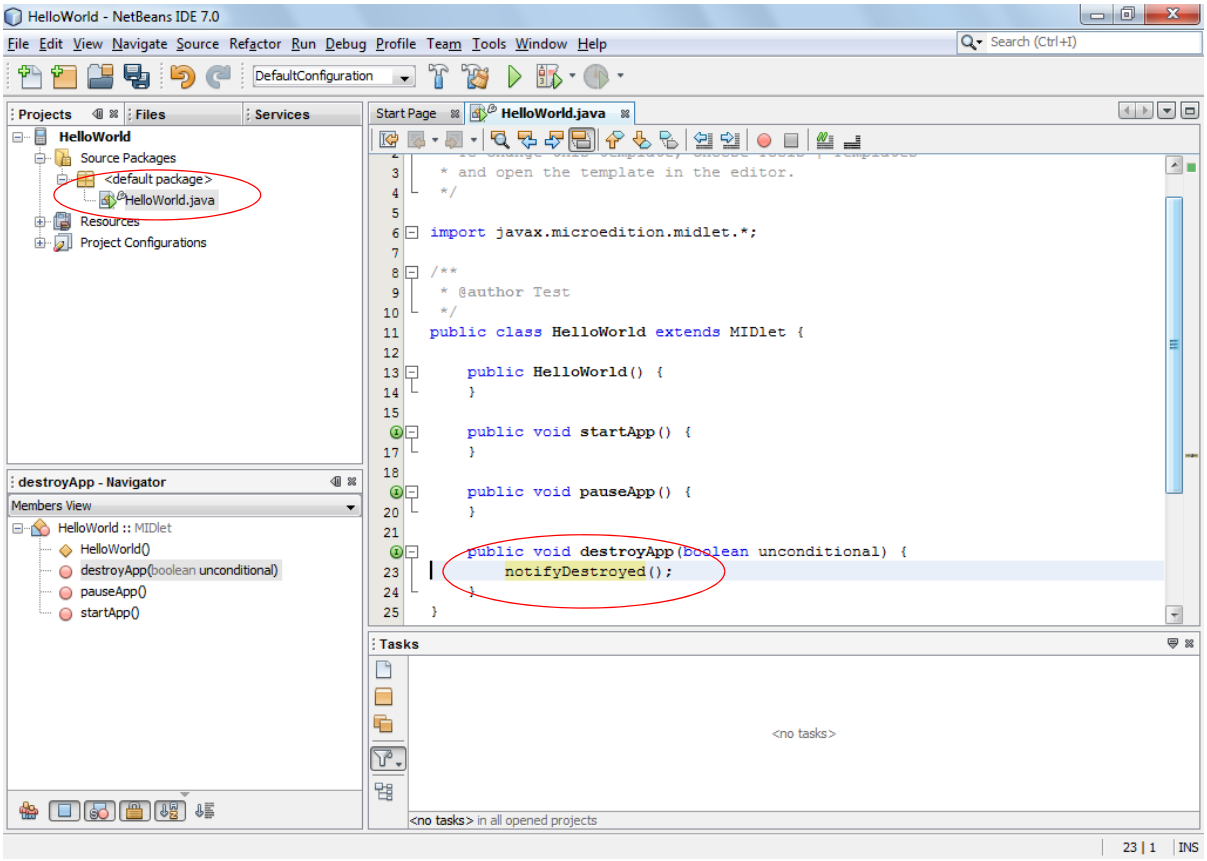

Figure 49: Creating a new MIDlet - HelloWorld.java

When the ATCommand class is to be used the corresponding library must be added to the project resources. Right-click on the "Resources" item in the "Projects" binder and select "Add Jar/Zip…" from the popup menu. Then navigate to the folder "Program Files\Cinterion\CMTK\BGS5\WTK\resources" and select the library file "cwmlib_1.0.jar". The provided example "AtCmdDemo" uses a local copy of this library. This is also possible.

Finally, the actual functionality can be implemented and the project can be started and debugged inside the module as other NetBeans projects. Please note that you must allow NetBeans and the debug agent to pass the firewall and configure the debug IP connection as a home network to be able to execute and debug MIDlets inside the device.

11 Java Security

The Java Security model follows the specification of MIDP 2.0 and conforms to IMP-NG. It integrates only a simple protection domain concept since protection domains are not required for module use cases.

Java Security is divided into two main areas:
- Secure MIDlet data links (HTTPS, Secure Connection) (see Section 11.1)
- Execution of signed/unsigned MIDlets (see Section 11.2)

The interface of Java Security offers the following functionality:
- Insert/delete keystores for MIDlet security (see Section 11.2.1)
- Toggle MES (default is ON see Section 11.3)
- Toggle HTTPS certificate verification (default is OFF, see Section 11.1)
- Add / Delete certificates for HTTPS certificate verification
- Add / Delete certificates and private keys for HTTPS client authentication (see Section 11.1.1)

Restrictions:
- The module does not supply user independent date/time base. Therefore no examination of the validity of the expiration date/time of the certificate takes place.

11.1 Secure Data Transfer

Secure data transfer allows 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 implementation follows the recommendations laid out in IMP-NG:

HTTPSConnection, SecureConnection

- HTTP / SecureConnection over SSL version 3.0 and TLS versions 1.0, 1.1 and 1.2.
  TLS_DHE_RSA_WITH_AES_256_CBC_SHA
  TLS_DHE_RSA_WITH_AES_128_CBC_SHA
  TLS_RSA_WITH_AES_256_CBC_SHA
  TLS_RSA_WITH_AES_128_CBC_SHA
  TLS_RSA_WITH_NULL_SHA
  SSL_RSA_WITH_RC4_128_SHA
  SSL_RSA_WITH_RC4_128_MD5
  SSL_RSA_WITH_3DES_EDE_CBC_SHA
  ECC cipher suites:
  TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
  TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
  TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
  TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
  TLS_ECDHE_RSA_WITH_RC4_128_SHA
  TLS_ECDHE_ECDSA_WITH_RC4_128_SHA
  TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
  TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
  Static ECDH cipher suites:
  TLS_ECDH_RSA_WITH_AES_256_CBC_SHA
  TLS_ECDH_RSA_WITH_AES_128_CBC_SHA
  TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA
  TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA
  TLS_ECDH_RSA_WITH_RC4_128_SHA
  TLS_ECDH_ECDSA_WITH_RC4_128_SHA
  TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA
  TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA
  CyaSSL extension - eSTREAM cipher suites:
  TLS_RSA_WITH_HC_128_CBC_MD5
  TLS_RSA_WITH_HC_128_CBC_SHA
  TLS_RSA_WITH_RABBIT_CBC_SHA
  SHA-256 cipher suites:
  TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
  TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
  TLS_RSA_WITH_AES_256_CBC_SHA256
  TLS_RSA_WITH_AES_128_CBC_SHA256
  TLS_RSA_WITH_NULL_SHA256
  AES-GCM cipher suites:
  TLS_RSA_WITH_AES_128_GCM_SHA256
  TLS_RSA_WITH_AES_256_GCM_SHA384
  TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
  TLS_DHE_RSA_WITH_AES_256_GCM_SHA384

ECC AES-GCM cipher suites:
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384
AES-CCM cipher suites:
TLS_RSA_WITH_AES_128_CCM_8_SHA256
TLS_RSA_WITH_AES_256_CCM_8_SHA384
TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_CCM_8_SHA384

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. See Figure 50.

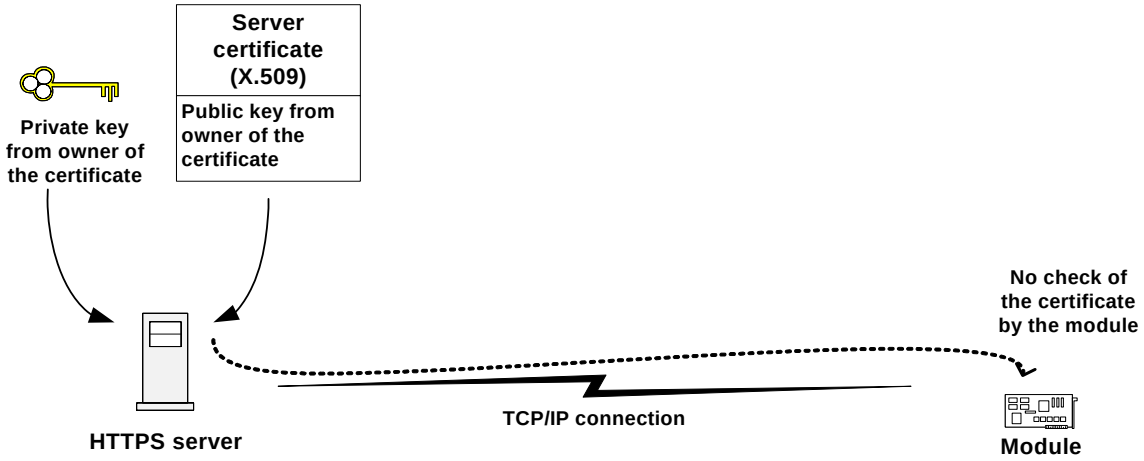

Figure 50: Mode 1 - Customer Root certificate does not exist

Mode 2 (see Section 11.2.1, 1. Step):

- HTTPS server certificate (that is a certificate store of trusted CA certificates) is inside of the module (cmd: "Add Certificate" was executed).
- Certificates are saved in DER format.
- Command to 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. See Figure 51 and Figure 52.

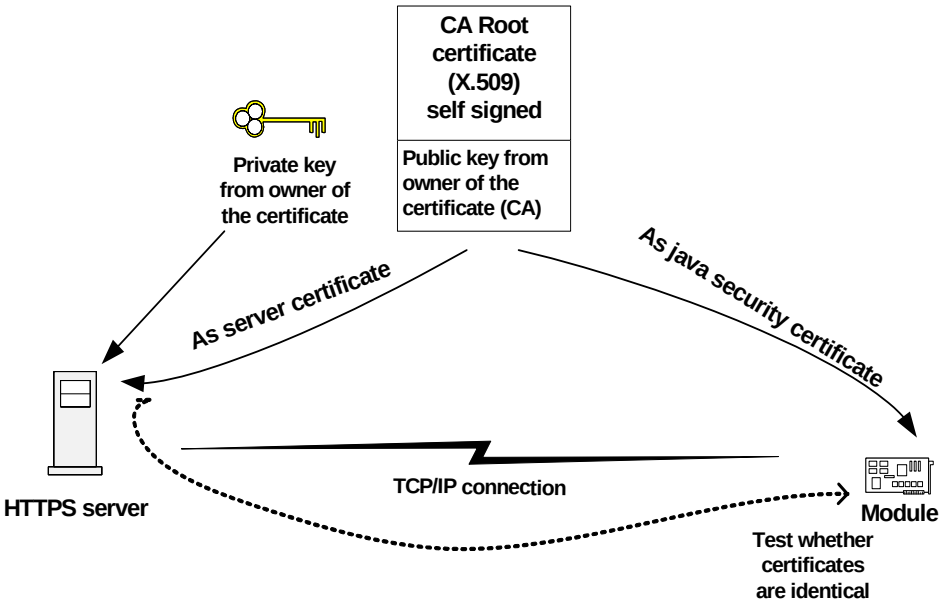

Figure 51: Mode 2 - Server certificate and certificate into module are identical

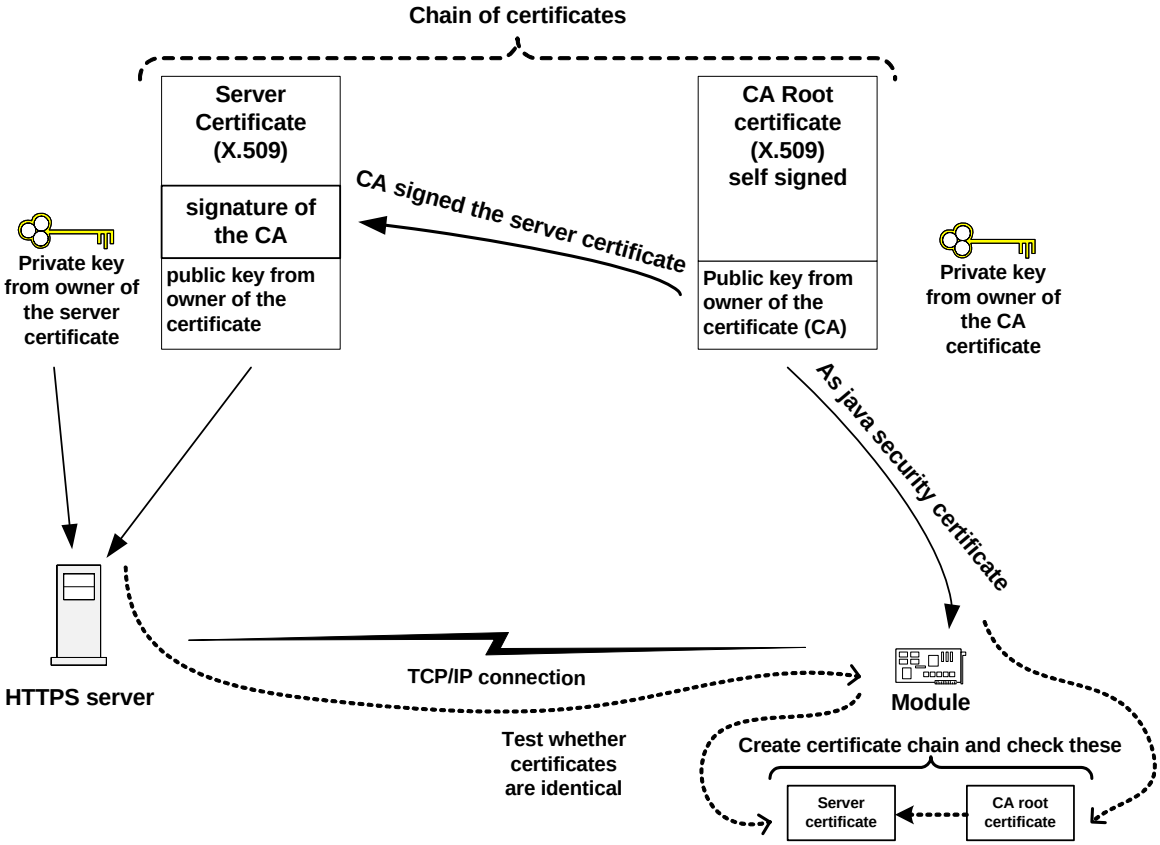

Figure 52: Mode 2 - Server certificate and self signed root certificate in module form a chain

11.1.1 HTTPS - Client Authentication

SSL enabled server can be configured to require client authentication. When an SSL enabled server requests client authentication it requires a separate piece of digitally signed data from the client to authenticate itself. The server uses the digitally signed data to validate the public key in the certificate and to authenticate the identity the certificate claims to represent.

The SSL protocol requires the client to create a digital signature by creating a one-way hash from data generated randomly during the handshake and known only to the client and server. The data hash is then encrypted with the private key that corresponds to the public key in the certificate being presented to the server.

To use this process, it is possible to deposit a client certificate (with public key) and a client private key on the module (for configuration command see Section 11.5.3).

The client certificate file and the client private key file must be in the file format "*.pem".

11.2 Execution Control

The Java environment of the ME supports two modes.

Unsecured mode:
- The device starts all Java applications (MIDlets).
- Java Security Commands that will be accepted in this mode:
  Set Customer Keystore
  Switch on/off Certificate Verification of the HTTPS Connections, Untrusted
  Add Certificate for Verification of the HTTPS Connections, Untrusted
  Del Certificate for Verification of the HTTPS Connections, Untrusted
  Del all Certificates for Verification of the HTTPS Connections, Untrusted
  Add Client Certificate for Client-Verification of the HTTPS Connections, Untrusted
  Del Client Certificate for Client-Verification of the HTTPS Connections, Untrusted

Secured mode:
- A condition for the secured mode of the device is the existence of a customer ME keystore with one certificate for the protection domain "operator" inside of the module.
- The customer can activate the secured mode of the device. To do so, the customer sends an ME keystore to the device (over an AT interface). The device changes from unsecured mode to secured mode. Now, the module will only start Java applications with a valid signature. In addition, the device will only accept special commands from the customer as long as they are marked with a signature. The device examines each command with the public key of the "operator" certificate from customer ME keystore.
- Java Security Commands that will be accepted in this mode:
  Del Customer Keystore
  Switch on/off Certificate Verification of the HTTPS Connections
  Switch on/off OBEX Functionality
  Add Certificate for Verification of the HTTPS Connections
  Del Certificate for Verification of the HTTPS Connections
  Del all Certificates for Verification of the HTTPS Connections
  Add Client Certificate for Client-Verification of the HTTPS Connections
  Del Client Certificate for Client-Verification of the HTTPS Connections

Standard behavior of the module:
Gemalto M2M supplies modules with unsecured mode as the default configuration.

Insert the customer ME keystore:
- The module changes into the mode "trusted" for MIDlet execution. "Untrusted Domain" is OFF.
- HTTPS certificate verification is ON.
- MES is OFF.

Remove the customer ME keystore:
- The module changes into the mode "untrusted" for MIDlet execution.
- HTTPS certificate verification is OFF
- MES is ON.

11.2.1 Change to Secured Mode Concept

Create and insert an ME keystore:
A condition for the change to secured mode is the existence of a customer ME keystore with certificate within the domain "operator".

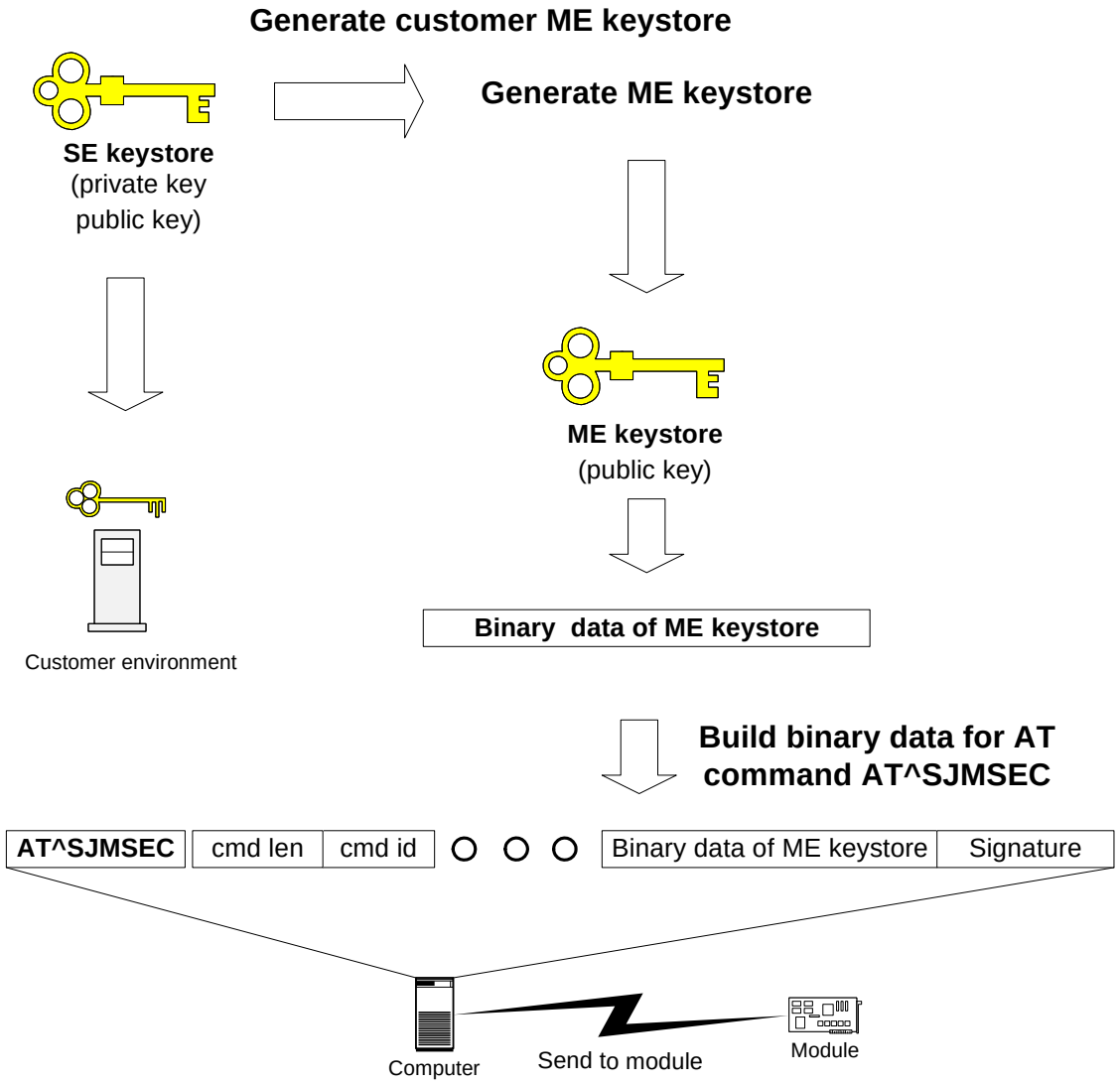

Figure 53: Insert customer ME keystore

After this action the module is in the following conditions:

- The module changes into the mode "trusted" for MIDlet execution. "Untrusted Domain" is OFF.

11.2.2 Concept for the Signing the Java MIDlet

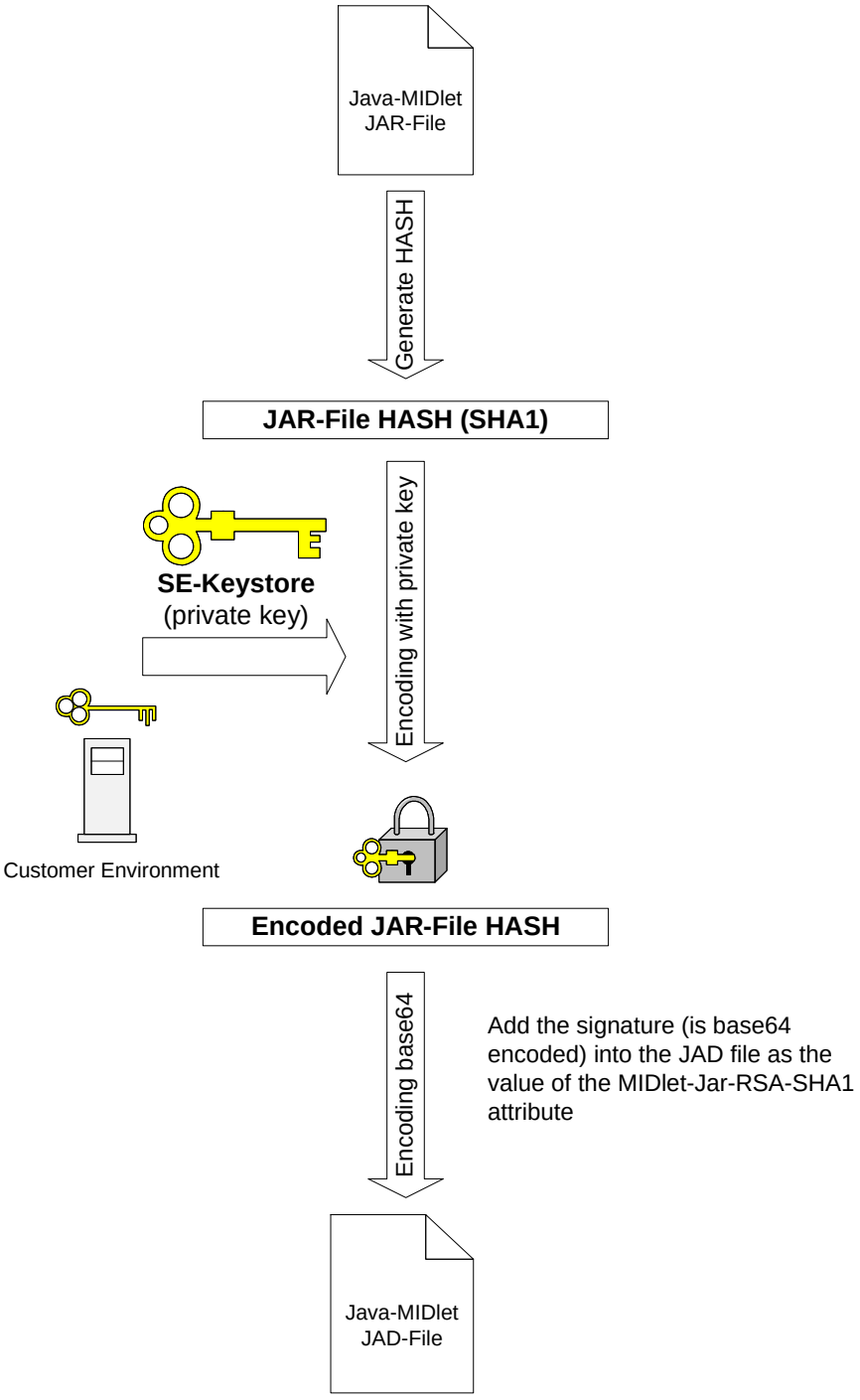

Figure 54: Prepare MIDlet for secured mode

11.3 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^SJMSEC, it is no longer possible to access 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 ME keystore inside of the module (see Section 11.2.1).

The default state of MES is ON.

11.4 Structure and Description of the Java Security Commands

Specific Java Security commands are available in the Java Security environment. These commands are transferred to the module with the help of the AT command AT^SJMSEC. This command allows to send binary data to the module. The binary data contain the actual Java Security commands.

Each Java Security command is module specific. It contains the IMEI of the module. Before the command is executed, the IMEI is checked.

11.4.1 Structure of the Java Security Commands

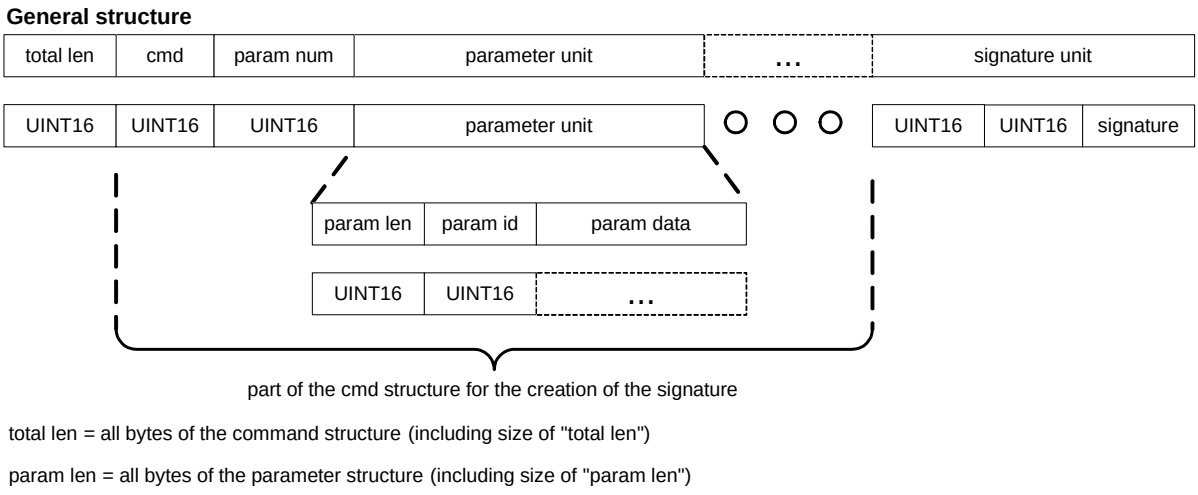

Figure 55: Structure of Java Security commands

Table 3: List of commands

CMD ID	Description
0x0001	Set Manufacturer Keystore
0x0002	Del Manufacturer Keystore
0x0003	Switch on/off Certificate Verification of the HTTPS Connections
0x0031	Switch on/off Certificate Verification of the HTTPS Connections, Untrusted
0x0004	Switch on/off OBEX Functionality
0x0005	Set Customer Keystore
0x0006	Del Customer Keystore
0x0007	Add Certificate for Verification of the HTTPS Connections
0x0071	Add Certificate for Verification of the HTTPS Connections, Untrusted
0x0008	Del Certificate for Verification of the HTTPS Connections
0x0081	Del Certificate for Verification of the HTTPS Connections, Untrusted
0x0009	Del all Certificates for Verification of the HTTPS Connections
0x0091	Del all Certificates for Verification of the HTTPS Connections, Untrusted
0x000A	Add Client Certificate for Client-Verification of the HTTPS Connections
0x00A1	Add Client Certificate for Client-Verification of the HTTPS Connections, Untrusted
0x000B	Del Client Certificate for Client-Verification of the HTTPS Connections
0x00B1	Del Client Certificate for Client-Verification of the HTTPS Connections, Untrusted

Table 4: List of parameters

Param ID	Param Len	Param Data	Description
0x0001	D	Binary data	DER/PEM coded certificate, keystore data
0x0002	0x0005	0x00 or 0x01	on/off switch, 0x00 = off, 0x01 = on
0x0003	0x0014	IMEI	numeric numbers in ASCII format (zero terminated string)
0x0004	256	Signature data	SHA-1 signature the of command
0x0005	D	File data	file name, used to store a certificate in the module (zero terminated string)
0x0006	D	Binary key data	PEM coded private key data

Table 5: Command structure

Set Manufacturer Keystore

Total len	0x0001	0x0003	Param unit IMEI	Param unit keystore	Param unit signature

Del Manufacturer Keystore

Total len	0x0002	0x0002	Param unit IMEI	Param unit signature

Switch on/off Certificate Verification for HTTPS Connections

Total len	0x0003	0x0003	Param unit IMEI	Param unit switch	Param unit signature

Switch on/off Certificate Verification for HTTPS Connections, Untrusted

Total len	0x0031	0x0001	Param unit switch

Switch on/off OBEX Functionality

Total len	0x0004	0x0003	Param unit IMEI	Param unit switch	Param unit signature

Set Customer Keystore

Total len	0x0005	0x0003	Param unit IMEI	Param unit keystore	Param unit signature

Del Customer Keystore

Total len	0x0006	0x0002	Param unit IMEI	Param unit signature

Add Certificate

Total len	0x0007	0x0004	Param unit IMEI	Param unit file name	Param unit bin data	Param unit signature

Add Certificate, Untrusted

Total len	0x0071	0x0002	Param unit file name	Param unit bin data

Del Certificate

Total len	0x0008	0x0003	Param unit IMEI	Param unit file name	Param unit signature

Del Certificate, Untrusted

Total len	0x0081	0x0001	Param unit file name

Del all Certificate

Total len	0x0009	0x0002	Param unit IMEI	Param unit signature

Del all Certificate, Untrusted

Total len	0x0091	0x0000

Add HTTPS Client Certificate

Total len	0x000A	0x0004	Param unit IMEI	Param unit bin data	Param unit bin key data	Param unit signature

Add HTTPS Client Certificate, Untrusted

Total len	0x00A1	0x0002	Param unit bin data	Param unit bin key data

Del HTTPS Client Certificate

Total len	0x000B	0x0002	Param unit IMEI	Param unit signature

Del HTTPS Client Certificate, Untrusted

Total len	0x00B1	0x0000

11.4.2 Build Java Security Commands

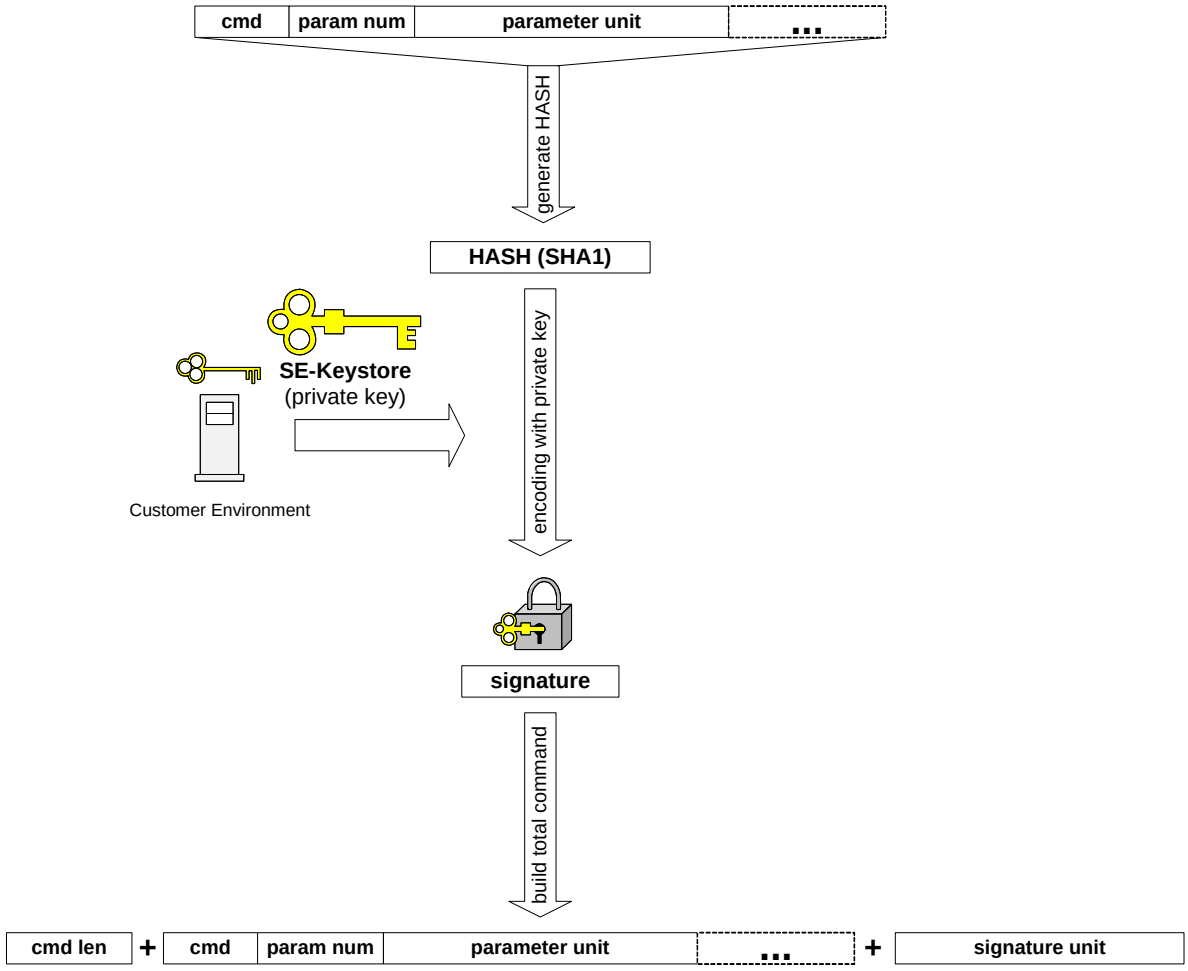

Figure 56: Build Java Security command

11.4.3 Send Java Security Command to the Module

The AT command AT^SJMSEC operates in two modes.

- Direct command mode, i.e., the Java Security command is part of the AT command
  - Del customer keystore
  - Switch on/off certificate verification for HTTPS connections
  - Switch on/off OBEX functionality
  - Del certificate
  - Del all certificate
  - Del HTTPS client certificate
- Indirect command mode, i.e., the Java Security command is stored in a file within the module
  - Set customer keystore
  - Add certificate
  - Add HTTPS client certificate

For the **direct commands** you can use the jseccmd tool (see Section 11.5.3). The output of this tool can be used for the <data> parameter of the AT^SJMSEC command.

Example:

```
AT^SJMSEC="cmd","230103000300140003003030343430313038303832313538360005000
20000040104007D140282B3BF73AA6F542C6F93B1FAE94783F23B57241EFE57DFD7B0E7B96
F0B934AA6C33B8BBD873746FE9BBEF5E238DEC1549C0E7B5FC6BE1001D6F361B8077FF7333
07C06C297A26CE411C182E6757DD7181D20DF097044B38F9D22F19A503F719A67F00E0E244
DABC90DA2782E96E85A8B5DFFE6128138629087D443A97D19E010393CC9EFB64D58D139084
31C52DD17E44150ECC4D5B58179263EEA3C288269E52C9BCB3DF21BEB9753E847AE4BB09BA
EFC48A90ECB853CF34CF6AC8486E5F92F3715509160EDFFEC40380018122F9B26AE6E385AA
ABE42A9DE094164A6132286EF9E0848DE1169E4EEA830D96873023E524723E03A4D45D7021
81DA079"
```

For the **indirect commands** you can also use the jseccmd tool (see Section 11.5.3). The output of this tool is a binary stream. You can save the stream to a file.

This file is copied into the file system of the module (root). MES is used for it. Then the AT command will be executed.

Example:

```
AT^SJMSEC="file","SetCustomerKeystore.bin"
```

11.4.3.1 AT^SJMSEC Command Syntax

Test command syntax

Test Command

```
AT^SJMSEC=?
```

Response(s)

```
^SJMSEC: <list of supported <CmdMode>s),(max. string length of <CmdData>)

OK
```

Read command syntax

Read Command

```
AT^SJMSEC?
```

Response(s)

```
^SJMSEC: <keystore state>,<https state>,<obex state>,<https client cert>

OK
ERROR
+CME ERROR: 21
```

Write command syntax

Write Command

```
AT^SJMSEC=<CmdMode>,<CmdData>
```

Response(s)

```
[^SJMSEC: <err code>,<str>]

OK
ERROR
+CME ERROR: 21
```

Parameter description

`<CmdMode>`(str)

Command mode (string).

"cmd"	Direct command mode, i.e., the subsequent <CmdData> contains the Java Security command.
"file"	Indirect command mode, the subsequent <CmdData> specifies the file name, where the Java security command is stored. Each file shall be copied into the root directory of the module's flash file system. To copy the files the MES has to be used.

`<CmdData>`(str)

Command data (string).

The <CmdData> depends on the <CmdMode> setting:

- If <CmdMode> is set to "cmd", <CmdData> contains the ASCII coded hex data of the Java Security command.
- If <CmdMode> is set to "file", <CmdData> provides the name of the file that contains the Java Security command in binary format.

`<keystore state>`(str)

Keystore usage (string).

"0"	No keystore installed. Internal mode dedicated for manufacturer.
"1"	No keystore installed. Unsigned midlets can still be installed (delivered).
"2"	Keystore installed. Only signed midlets can be installed.

`<https state>`(str)

Security check (string).

"0"	HTTPS connection or Secure Connection without check against the certificate store (default)
"1"	HTTPS connection or Secure Connection with check against the certificate store

`<obex state>`(str)

Obex state (string).

"0"	Start of Module Exchange Suite is not permitted
"1"	Start of Module Exchange Suite is permitted (default)

`<https client cert>`(str)

HTTPS client certificate (string).

"0"	Client certificate not installed (default)
"1"	Client certificate installed

`<err code>`(str)

Error codes (string).

"0"	(no error)
"1"	wrong command format
"2"	wrong command parameter format
"3"	parameter unknown
"4"	command unknown
"5"	parameter IMEI is missing
"6"	parameter signature is missing
"7"	keystore error
"8"	certificate file exits
"9"	cannot create certificate file
"10"	certificate file does not exists
"11"	jsec.cfg error
"12"	cannot delete certificate file
"13"	certificate store does not exists
"14"	cannot read command file
"15"	IMEI is wrong
"16"	cannot create key file
"17"	unknown error

`<str>`(str)

Error string contains the Java exception message.

Please note that the module must be reset after each command.

11.5 Create a Java Security Environment Step by Step

11.5.1 Create SE Keysstore

Java keystores (SE keystore) provide a mechanism for storing and deploying X.509 certificates and private keys.

The SE keystore 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 Development Kit (JDK). For a description see http://docs.oracle.com/javase/6/docs/technotes/tools/solaris/keytool.html.

For the RSA key a key size of 2048 is used.

Example:

```
keytool -genkey -alias operator -keypass keypass -keystore ./se_customer.ks
     -storepass keystorepass -sigalg SHA1withRSA -keyalg RSA -keysize 2048
```

11.5.2 Create ME Keystore

The MEKeytool utility enables you to import keys from Java SE keystores.
MEKeytool is an executable file located under "wkt\bin".

Example:

```
mekeytool -import -MEkeystore ./me_customer.ks -alias operator
     -domain operator -keystore ./se_customer.ks -storepass keystorepass
```

11.5.3 Create Java Security Commands

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

Command to insert the customer ME keystore

```
java -jar jseccmd.jar -cmd SetCustomerKeystore
                      -imei 012345678901234 -alias operator
                      -keypass keypass -storepass keystorepass
                      -keystore ./se_customer.ks
                      -filename ./me_customer.ks > SetCustomerKeystore.bin
```

Command to remove the customer ME keystore

```
java -jar jseccmd.jar -cmd DelCustomerKeystore
                      -imei 012345678901234 -alias operator
                      -keypass keypass -storepass keystorepass
                      -keystore ./se_customer.ks > DelCustomerKeystore.txt
```

Command to switch ON certificate verification for HTTPS connections

```
java -jar jseccmd.jar -cmd HttpsVerifyOn
                      -imei 012345678901234 -alias keyname
                      -storepass keystorepassword -keypass keypassword
                      -keystore ./se_customer.ks > HttpsVerifyOn.txt
```

Command to switch ON certificate verification for HTTPS connections (Unsecured mode)

```
java -jar jseccmd.jar -cmd HttpsVerifyOnUntrusted >
                           HttpsVerifyOnUntrusted.txt
```

Command to switch OFF certificate verification for HTTPS connections

```
java -jar jseccmd.jar -cmd HttpsVerifyOff
                      -imei 012345678901234 -alias keyname
                      -storepass keystorepassword -keypass keypassword
                      -keystore ./se_customer.ks > HttpsVerifyOff.txt
```

Command to switch OFF certificate verification for HTTPS connections (Unsecured mode)

```
java -jar jseccmd.jar -cmd HttpsVerifyOffUntrusted >
                           HttpsVerifyOffUntrusted.txt
```

Command to ADD certificate for verification for HTTPS connections using the certificate file in DER format

```
java -jar jseccmd.jar -cmd AddHttpsCertificate
                      -imei 012345678901234 -alias keyname
                      -storepass keystorepassword -keypass keypassword
                      -keystore ./se_customer.ks
                     -filename ./certificate.der > AddHttpsCertificate.bin
```

Command to ADD certificate for verification for HTTPS connections using the certificate file in DER format (Unsecured mode)

```
java -jar jseccmd.jar -cmd AddHttpsCertificateUntrusted
                     -filename ./certificate.der > AddHttpsCertificate.bin
```

Command to DELete certificate for verification for HTTPS connections

```
java -jar jseccmd.jar -cmd DelHttpsCertificate
                      -imei 012345678901234 -alias keyname
                      -storepass keystorepassword -keypass keypassword
                      -keystore ./se_customer.ks
                      -filename certificate.der > DelHttpsCertificate.txt
```

Command to DELete certificate for verification for HTTPS connections (Unsecured mode)

```
java -jar jseccmd.jar -cmd DelHttpsCertificateUntrusted >
           -filename certificate.der > DelHttpsCertificateUntrusted.txt
```

Command to DELete ALL certificates for verification for HTTPS connections

```
java -jar jseccmd.jar -cmd DelAllHttpsCertificates
                      -imei 012345678901234 -alias keyname
                      -storepass keystorepassword -keypass keypassword
                  -keystore ./se_customer.ks > DelAllHttpsCertificates.txt
```

Command to DELete ALL certificates for verification for HTTPS connections (Unsecured mode)

```
java -jar jseccmd.jar -cmd DelAllHttpsCertificatesUntrusted >
                           DelAllHttpsCertificatesUntrusted.txt
```

Command to ADD client certificate for verification for HTTPS connections using the client certificate file and the private key file in PEM format

```
java -jar jseccmd.jar -cmd AddHttpsClientCertificate
                      -imei 012345678901234 -alias keyname
                      -storepass keystorepassword -keypass keypassword
                      -keystore ./se_customer.ks
                      -filename ./certificate.pem
               -keyfilename clientkey.pem > AddHttpsClientCertificate.bin
```

Command to ADD client certificate for verification for HTTPS connections using the client certificate file and the private key file in PEM format (Unsecured mode)

```
java -jar jseccmd.jar -cmd AddHttpsClientCertificateUntrusted
                      -filename ./certificate.pem
      -keyfilename clientkey.pem > AddHttpsClientCertificateUntrusted.bin
```

Command to DELete client certificate for verification for HTTPS connections

```
java -jar jseccmd.jar -cmd DelHttpsClientCertificate
                      -imei 012345678901234 -alias keyname
                      -storepass keystorepassword -keypass keypassword
                  -keystore ./se_customer.ks > DelHttpsClientCertificate.txt
```

Command to DELete client certificate for verification for HTTPS connections (Unsecured mode)

```
java -jar jseccmd.jar -cmd DelHttpsClientCertificateUntrusted >
                           DelHttpsClientCertificateUntrusted.txt
```

Command to switch ON module exchange functionality

```
java -jar jseccmd.jar -cmd ObexActivationOn
                      -imei 012345678901234 -alias keyname
                      -storepass keystorepassword -keypass keypassword
                      -keystore ./se_customer.ks > ObexActivationOn.txt
```

Command to switch OFF module exchange functionality

```
java -jar jseccmd.jar -cmd ObexActivationOff
                      -imei 012345678901234 -alias keyname
                      -storepass keystorepassword -keypass keypassword
                      -keystore ./se_customer.ks > ObexActivationOff.txt
```

11.5.4 Sign a MIDlet

Use the tool "jadtool.exe" to sign a Java MIDlet and add certificates into it. This program is located under "wkt\bin".

First step: Add signature

```
jadtool -addjarsig -jarfile ./helloworld.jar
        -inputjad ./helloworld.jad
        -outputjad ./helloworld.jad
        -alias operator -storepass keystorepassword
        -keypass keypassword -keystore ./se_customer.ks
        -encoding UTF-8
```

Second step: Add certificates

```
jadtool -addcert
        -inputjad ./helloworld.jad
        -outputjad ./helloworld.jad
        -alias operator -storepass keystorepassword
        -keypass keypassword -keystore ./se_customer.ks
```

The best way is of course to use a development environment (Netbeans, Eclipse).

11.6 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 useless. 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**.

12 Differences to EGS5/TC65i

This chapter gives a short overview of the main differences in the Java implementation between established Gemalto M2M products like TC65i or EGS5 and the newer product BGSx.

- The structure of the wtk/sdk has been changed slightly. Most notably classes.zip has been split up in multiple .jar files, so called stubs. The HTML documentation has also been split up into different packages. There is one core package (html_impng) containing CLDC, IMP-NG and all Cinterion APIs and there are additional packages for further JSRs.
- A view hardware interface functions share the same hardware pins. Therefore some hardware interfaces need to be enabled by AT command before they can be used via the Java API.
- MIDlet installation, start, stop and de-installation is controlled by the AT command AT^SJAM. The AT command AT^SJRA is deprecated. After installation the MIDlet is stored in an internal section of the flash file system. This section also holds the MIDlets' "RecordStore". This means MIDlets and their record stores are not visible in the flash file system.
- In order to fit to the improved MIDlet installation mechanisms and also to handle multiple MIDlets the OTAP mechanism has been adapted. The "Application directory" parameter does no longer exist. The "JAD file URL" parameter can now also be a partial URL (e.g. to a specific server), omitting the file name itself.
- The Java security AT command AT^SJSEC has been replaced by AT^SJMSEC offering extended functionality.
- The mechanism to start MIDlets automatically at system startup does require an AT command setting as well as certain properties in the descriptor of MIDlet.
- The Java file access API (FileConnection) is now a certified JSR75 implementation and is therefore no longer in the Cinterion package.
- The Watchdog class has been replaced by the Watchdog2 class. The Watchdog class is now deprecated.
- The class NetExtension has been added to offer improved network functionality and error reporting.
- The class FileExtension has been added to offer improved error reporting.
- The BearerControl class has been extended to handle multiple bearer (multiple PDP contexts) in parallel.
- Java standard APIs have been added: JSR280 (XML), JSR177 CRYPTO
- System.currentTimeMilis() always returns the time of the RTC that can be set with AT+CCLK. This also affects the Calendar class.
- The AT^SJNET parameter <timeout> has been changed in its meaning.

About Gemalto

Gemalto (Euronext NL0000400653 GTO) is the world leader in digital security with 2011 annual revenues of €2 billion and more than 10,000 employees operating out of 74 offices and 14 Research & Development centers, located in 43 countries.

We are at the heart of the rapidly evolving digital society. Billions of people worldwide increasingly want the freedom to communicate, travel, shop, bank, entertain and work - anytime, everywhere - in ways that are enjoyable and safe. Gemalto delivers on their expanding needs for personal mobile services, payment security, authenticated cloud access, identity and privacy protection, eHealthcare and eGovernment efficiency, convenient ticketing and dependable machine-to-machine (M2M) applications.

Gemalto develops secure embedded software and secure products which we design and personalize. Our platforms and services manage these secure products, the confidential data they contain and the trusted end-user services they enable. Our inovations enable our clients to offer trusted and convenient digital services to billions of individuals.

Gemalto thrives with the growing number of people using its solutions to interact with the digital and wireless world.

For more information please visit
m2m.gemalto.com, www.facebook.com/gemalto, or Follow@gemaltom2m on twitter.

Gemalto M2M GmbH
St.-Martin-Str. 60
81541 Munich
Germany

➲ M2M.GEMALTO.COM

gemalto
security to be free

© Gemalto 2013. All rights reserved. Gemalto, the Gemalto logo, are trademarks and service marks of Gemalto and are registered in certain countries. April 2013