All information herein is either public information or is the property of and owned solely by Gemplus S.A. who shall have and keep the sole right to file patent applications or any other kind of intellectual property protection in connection with such information.

Nothing herein shall be construed as implying or granting to you any rights, by license, grant or otherwise, under any intellectual and/or industrial property rights of or concerning any of Gemplus’ information.

This document can be used for informational, non-commercial, internal and personal use only provided that:

- The copyright notice below, the confidentiality and proprietary legend and this full warning notice appear in all copies.
- This document shall not be posted on any network computer or broadcast in any media and no modification of any part of this document shall be made.

Use for any other purpose is expressly prohibited and may result in severe civil and criminal liabilities.

The information contained in this document is provided “AS IS” without any warranty of any kind. Unless otherwise expressly agreed in writing, Gemplus makes no warranty as to the value or accuracy of information contained herein. The document could include technical inaccuracies or typographical errors. Changes are periodically added to the information herein. Furthermore, Gemplus reserves the right to make any change or improvement in the specifications data, information, and the like described herein, at any time.

Gemplus hereby disclaims all warranties and conditions with regard to the information contained herein, including all implied warranties of merchantability, fitness for a particular purpose, title and non-infringement. In no event shall Gemplus be liable, whether in contract, tort or otherwise, for any indirect, special or consequential damages or any damages whatsoever including but not limited to damages resulting from loss of use, data, profits, revenues, or customers, arising out of or in connection with the use or performance of information contained in this document.

Gemplus does not and shall not warrant that this product will be resistant to all possible attacks and shall not incur, and disclaims, any liability in this respect. Even if each product is compliant with current security standards in force on the date of their design, security mechanisms’ resistance necessarily evolves according to the state of the art in security and notably under the emergence of new attacks. Under no circumstances, shall Gemplus be held liable for any third party actions and in particular in case of any successful attack against systems or equipment incorporating Gemplus products. Gemplus disclaims any liability with respect to security for direct, indirect, incidental or consequential damages that result from any use of its products. It is further stressed that independent testing and verification by the person using the product is particularly encouraged, especially in any application in which defective, incorrect or insecure functioning could result in damage to persons or property, denial of service or loss of privacy.

© Copyright 2002 Gemplus S.A. All rights reserved. Gemplus, the Gemplus logo, and GemXpresso are trademarks and service marks of Gemplus S.A. and are registered in certain countries. All other trademarks and service marks, whether registered or not in specific countries, are the property of their respective owners. Certain Smart Cards produced by Gemplus are covered by Bull CPS Patents.

This product includes software developed by the Apache Software Foundation (http://www.apache.org).

GEMPLUS, B.P. 100, 13881 GEMENOS CEDEX, FRANCE.
Tel: +33 (0)4.42.36.50.00 Fax: +33 (0)4.42.36.50.90
Printed in France. Document Reference: DOC107570A0
May 7, 2002
Contents

Preface xi
Prerequisites .......................................................... xii
Who Should Read This Book ....................................... xii
Conventions Used in This Document ............................. xiii
Contact Our Hotline .................................................xiv

Chapter 1 Developing Project Code 1
Introduction ............................................................. 1
Developing a Project Using Templates ........................... 2
  Installing the Project Templates ................................. 2
  Creating a New Project Using a Project Template ............ 2
  Creating a Java Card Applet .................................... 4
  Creating a Java Card Library ................................... 6
  Creating a Custom Project ..................................... 7
  Importing a GemXpresso Project ............................... 8
  Accessing Other GemXpresso RAD III Tools ................ 9
The OpenCard Framework (OCF) ................................ 9
  Configuring OCF .................................................. 11
Developing Projects Without Templates ......................... 14
  Configuring the Applet Development Environment .......... 14
  Configuring the Client Application Development Environment 15
Development Guidelines ............................................ 17
  Java Language Restrictions ................................... 17
  Applet Class and Method Implementations .................. 17
Next Steps ........................................................... 19

Chapter 2 Using the Project Editor 21
Overview ............................................................... 21
Starting the Project Editor ........................................ 22
The Project Editor Window ........................................ 22
Chapter 6  The GxpLoader  79
Running the GxpLoader .......................................................... 79
Running GxpLoader from the Command Line ............................... 79
Using GxpLoader .................................................................. 80
Configuring Parameters for the GSE ......................................... 80
Configuring Parameters for a Card ............................................. 81

Chapter 7  The Deployment Editor  83
Introduction ........................................................................ 83
Running the Deployment Editor .............................................. 83
Using the Deployment Editor .................................................. 84
Creating a New Deployment File .............................................. 84
Opening an Existing Deployment File ..................................... 86
The Deployment Editor Window .............................................. 87

Chapter 8  The CapDump Utility  89
Launching the CapDump Utility .............................................. 89
Using the CapDump Utility .................................................... 90
Diagnosing the CAP File Contents ........................................ 92

Chapter 9  The GCF to GXCF Converter  93
Running the GCF to GXCF Script Converter ............................... 93
Running the Script Converter with a Batch File ............................ 93
Running the Script Converter from the Command Line ............... 93
Configuring the Script Converter ........................................... 94

Appendix A  Sending and Receiving APDUs  95
Introduction ........................................................................ 95
Command APDU Formats ...................................................... 97
Response APDU Formats ...................................................... 98
Sending and Receiving APDUs ............................................... 98
Sending Command APDUs ................................................... 98
Viewing Response APDUs ................................................... 99
Coding Command and Response APDUs .................................. 99
Coding Command APDUs .................................................... 99
Coding Response APDUs .................................................... 101
Appendix B  Sample Code  103
The OPPurse Applet ............................................................... 103

Appendix C  Using Sun’s Converter Tool  119
Launching the Converter from the Command Line ...................... 119
Launching the Converter with a Configuration File ...................... 121

Appendix D  Key Diversification  123
Diversification Method .............................................................. 123
Diversification Data ................................................................. 124

Appendix E  Checking PC/SC Installation  125
Checking the PC/SC Reader Driver Installation ............................ 126
  Windows 98 ............................................................................ 126
  Windows NT4 ........................................................................ 128
  Windows 2000 ....................................................................... 129
Starting the PC/SC Smart Card Resources Manager ...................... 130
  Windows 98 ............................................................................ 130
  Windows NT4 ........................................................................ 131
  Windows 2000 ....................................................................... 131

Terminology  133
Abbreviations ........................................................................... 133
Glossary .................................................................................. 136

For More Information  141
Standards and Specifications ....................................................... 141
Recommended Reading ............................................................ 142
  GemXpresso RAD III Documentation ..................................... 142
Applet Development ............................................................... 142
Java: The Language and the Platform ........................................ 143
Smart Card and Secure Programming ........................................ 143
Cryptography ........................................................................... 144
Web References ....................................................................... 144

Index  145
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>JBuilder’s Object Gallery Window</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>VisualCafé’s New Project Window</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>The Applet Project Wizard Window, Step 1</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>The Applet Project Wizard Window, Step 2</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>The Library Project Wizard Window, Step 1</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>The Library Project Wizard Window, Step 2</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>The Import GemXpresso Project Wizard Window</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>The GemXpresso RAD Menu</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>Gemplus Services in the OpenCard Framework</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>The Project Editor and GxpConverter</td>
<td>21</td>
</tr>
<tr>
<td>11</td>
<td>The Project Editor Window</td>
<td>22</td>
</tr>
<tr>
<td>12</td>
<td>Determining the Class Directory and Package Name</td>
<td>24</td>
</tr>
<tr>
<td>13</td>
<td>Default Applet AIDs</td>
<td>24</td>
</tr>
<tr>
<td>14</td>
<td>The Cards Selection Window</td>
<td>25</td>
</tr>
<tr>
<td>15</td>
<td>The Conversion Process with GxpConverter</td>
<td>30</td>
</tr>
<tr>
<td>16</td>
<td>The Convert Window</td>
<td>32</td>
</tr>
<tr>
<td>17</td>
<td>Choosing Target Card Types</td>
<td>33</td>
</tr>
<tr>
<td>18</td>
<td>The JCardManager</td>
<td>38</td>
</tr>
<tr>
<td>19</td>
<td>The JCardManager Window</td>
<td>40</td>
</tr>
<tr>
<td>20</td>
<td>The Key File Editor Window</td>
<td>43</td>
</tr>
<tr>
<td>21</td>
<td>The Card Explorer Window</td>
<td>46</td>
</tr>
<tr>
<td>22</td>
<td>The Edit Command List Button</td>
<td>52</td>
</tr>
<tr>
<td>23</td>
<td>The Edit Command List Window</td>
<td>52</td>
</tr>
<tr>
<td>24</td>
<td>The Script Toolbar</td>
<td>54</td>
</tr>
<tr>
<td>25</td>
<td>GSE Architecture and Environment</td>
<td>58</td>
</tr>
<tr>
<td>26</td>
<td>The Gemplus Simulation Environment Launcher Window</td>
<td>69</td>
</tr>
<tr>
<td>27</td>
<td>The Gemplus Simulation Environment Window</td>
<td>70</td>
</tr>
<tr>
<td>28</td>
<td>GSE GUI Status Line</td>
<td>71</td>
</tr>
<tr>
<td>29</td>
<td>The Default Preferences Window</td>
<td>72</td>
</tr>
<tr>
<td>30</td>
<td>Default GSE Values</td>
<td>76</td>
</tr>
<tr>
<td>31</td>
<td>Naming the New Deployment File</td>
<td>84</td>
</tr>
<tr>
<td>32</td>
<td>Selecting Card Profiles</td>
<td>85</td>
</tr>
<tr>
<td>33</td>
<td>Selecting a Deployment File to Open</td>
<td>86</td>
</tr>
<tr>
<td>34</td>
<td>The Deployment Editor Window</td>
<td>87</td>
</tr>
<tr>
<td>35</td>
<td>Configure the Command Set for Your Deployment File</td>
<td>88</td>
</tr>
<tr>
<td>36</td>
<td>The CapDump Window</td>
<td>90</td>
</tr>
<tr>
<td>37</td>
<td>CAP File Contents</td>
<td>92</td>
</tr>
<tr>
<td>38</td>
<td>Command and Response APDUs</td>
<td>95</td>
</tr>
<tr>
<td>39</td>
<td>Structure of Command and Response APDUs</td>
<td>96</td>
</tr>
<tr>
<td>40</td>
<td>Command APDU Formats</td>
<td>97</td>
</tr>
<tr>
<td>41</td>
<td>Response APDU Formats</td>
<td>98</td>
</tr>
<tr>
<td>42</td>
<td>The Control Panel Window (Windows 98)</td>
<td>126</td>
</tr>
</tbody>
</table>
List of Tables

Table 1 - JavaCard and OP Specification Compatibility ........................................ xii
Table 2 - Accessing GemXpresso RAD III Tools .................................................. 9
Table 3 - GSE Launch Methods ............................................................................. 61
Table 4 - Available GSE Simulation Modes ............................................................... 67
Table 5 - Default Card Manager AID Values for Virtual Mode ............................... 76
Table 6 - Default CPLC Values for the GSE ............................................................. 76
Table 7 - Command APDU Fields ......................................................................... 96
Table 8 - Response APDU Fields ........................................................................... 97
Table 9 - VISA Key Diversification Data ................................................................. 124
Table 10 - VISA2 Key Diversification Data ............................................................. 124
GemXpresso RAD III bridges the gap between smart card technology and the Java world. Smart cards, embedded with a chip containing a microprocessor and an input and output port, have become personal pieces of hardware providing persistent, portable, and secure data storage.

Previously, programming these smart cards was a specialist activity: no off-the-shelf development tools were available. However, with the release of the Java Card—a smart card with a built-in Java Virtual Machine capable of running platform-independent Java code—smart card applications can be written by any programmer familiar with Java. The standardization of the Java language used for smart cards, communication protocols and APIs ensures interoperability of Java Card applications at source code level.

Yet two main problems in developing applications for smart cards remained:

- The limited amount of memory space available on smart cards.
- The difficulty of debugging smart card-based applications.

Gemplus' GemXpresso RAD III resolves both these issues by offering Java Card developers components for:

- Ensuring efficient source code design of Java Card applets and client applications with project templates.
- Debugging applications from within popular software tools such as VisualCafé or JBuilder, or any other JDK1.2.2-compatible integrated development environment (IDE).
- Simulating an applet running in a smart card, using GemXpresso RAD III’s sophisticated card simulator, the Gemplus simulation environment (GSE).
In addition, GemXpresso RAD III provides tools for:

- Converting an applet’s Class files to the loadable CAP file format, targeting one or more cards simultaneously.
- Loading an applet onto a target card.
- Displaying the contents of a CAP file in a human-readable form.
- Estimating the amount of memory an applet will occupy on the card.
- Issuing commands to a card, including loading and installing applets in a card.

The GemXpresso RAD III environment has been developed to be compatible with a wide range of smart cards, including the GemXpresso family of multi-applet Java-OP cards. GemXpresso RAD III is fully compatibility with both the Java Card and Open Platform standards. The versions of the standards for each card profile is shown in the following table:

<table>
<thead>
<tr>
<th>Card Profile</th>
<th>JavaCard Spec Version</th>
<th>OP Spec Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>GemXpresso 211 V2 &amp; IS</td>
<td>2.1</td>
<td>2.0</td>
</tr>
<tr>
<td>GemXpresso 211 PK &amp; PK_IS</td>
<td>2.1.1</td>
<td>2.0.1</td>
</tr>
<tr>
<td>GemXpresso Lite Generic</td>
<td>2.1.1</td>
<td>2.0.1</td>
</tr>
<tr>
<td>GemXpresso Pro R3</td>
<td>2.1.1</td>
<td>2.0.1’</td>
</tr>
</tbody>
</table>

Table 1 - JavaCard and OP Specification Compatibility

**Prerequisites**

This book assumes that GemXpresso RAD III has already been successfully installed, as described in GemXpresso RAD III Version 3.2 Getting Started.

**Who Should Read This Book**

A basic knowledge of Java programming is assumed.

You need no specific knowledge about smart cards, but you are strongly recommended to make use of the standards and documents listed in “Appendix G - For More Information”. Fundamental information is contained in the introduction, appendices and the respective card reference manuals.

The cryptographic techniques cited in this document are detailed in the references contained in “Appendix G - For More Information”. 
Conventions Used in This Document

**bold** Command and window names are shown in bold. For example:

... the **JCardManager** window...

> Menu selection sequences are shown using the > symbol to link the selections. For example:

... select **Start > Programs > Gemplus Applications**...

**italics** Cross references and book titles are shown in **italics**.

**notation**
- By default, a numeric value is expressed in decimal notation.
- Whenever a value is expressed in binary, it is followed by the letter “b”. For example the decimal value 13 expressed in binary becomes **1101b**.
- A hexadecimal number is followed by the letter “h”, or preceded by “0x”. For example, the decimal value 13 expressed in hexadecimal becomes **0Dh** or **0x0D**.

**installdir** This variable is used throughout this book to indicate the directory in which GemXpresso RAD III is installed. For example, when the following directory is shown in the book:

```
installdir\bin
```

you may need to substitute the following directory:

```
c:\Gemplus\GemXpresso.rad3\bin
```

**cardname** This variable is used throughout this book to indicate a particular type of card. The possible card types are:

- GXP211_PK (GemXpresso 211 Compact PK)
- GXP211_PK_IS (GemXpresso 211 Compact PK with limited cryptography)
- GXP211v2 (GemXpresso 211 V2)
- GXP211v2_IS (GemXpresso 211 V2 with limited cryptography)
- GXP Lite-Generic (All GemXpresso Lite cards)
- GXPPro-R3

For example, if you are using a GemXpresso Pro R3 card, substitute “GXPPro-R3” for **cardname** wherever it appears in the text.
Contact Our Hotline

If you do not find the information you need in this manual, or if you find errors, contact the Gemplus hotline by phone, fax, or email. In your email, please include the document reference number, your job function, and the name of your company. (You will find the document reference number at the bottom of the legal notice on the inside front cover.)

**Corporate and EMEA**

Hotline: +33 (0)4 42 36 50 50  
Hot Fax: +33 (0)4 42 36 50 98  
Email: hotline@gemplus.com

**Americas**

Hotline: 1 (877) 436-7233  
Hot Fax: 1 (215) 390-1586  
Email: hotlineusa@gemplus.com

**From Our Web Site**

http://www.gemplus.com
Developing Project Code

Introduction

Before beginning to develop source code, you should read the GemXpresso RAD III Version 3.2 Getting Started document.

You can develop source code for an applet and its associated client application in several ways:

• Within an IDE (integrated development environment)
• Outside an IDE, using any JDK 1.2.2-compliant development environment

Working within an IDE provides many advantages: an IDE integrates a complete set of development and debugging tools that can be used without leaving the IDE. Furthermore, for two of the most popular Java-based IDEs, JBuilder and VisualCafé, GemXpresso RAD III provides the following tools:

• Ready-to-use templates for creating the most common project types. Each template uses a wizard to gather project-related information.
• A plug-in that adds a GemXpresso RAD menu item to the IDE’s menu bar, from which many of the GemXpresso RAD III tools can be launched directly.

If you are not using VisualCafé or JBuilder, the GemXpresso RAD III tools can either be started from within the JCardManager, or launched from the command line using supplied batch files (see “Method 3: Command Line Launch” on page 64 for details).
Developing a Project Using Templates

A template contains skeleton Java code and other components necessary for building Java Card applets and the corresponding client applications.

The following GemXpresso RAD templates are available:

- Java Card Applet
- Java Card Library
- Custom Java Card Project
- Import Java Card Project

Installing the Project Templates

The GemXpresso RAD III installation program automatically detects the presence of a compatible version of JBuilder or VisualCafé on the local workstation, copies the project templates to the appropriate directory, and configures the IDE to load the plug-in when started.

Creating a New Project Using a Project Template

The GemXpresso RAD III templates are accessed using the usual project creation procedure of the respective IDE.

Creating New Projects with JBuilder

To create a new Java Card project using JBuilder:

2. On the Object Gallery window, click the Gemplus tab to display the Object Gallery window:

   ![Figure 1 - JBuilder’s Object Gallery Window](image)

   Figure 1 - JBuilder’s Object Gallery Window
3. Select the project type to create and click OK. Depending on your selection, a wizard is launched to guide you through the project definition phase.
   To create a Java Card applet, see “Creating a Java Card Applet” on page 4.
   To create a Java Card library, see “Creating a Java Card Library” on page 6.
   To create a custom Java Card project, see “Creating a Custom Project” on page 7.
   To import a Java Card project, see “Importing a GemXpresso Project” on page 8.

**Creating New Projects with VisualCafé**

To create a new Java Card project using VisualCafé:

2. On the New Project window, select the template type:

![VisualCafé's New Project Window](image)

*Figure 2 - VisualCafé's New Project Window*

3. Click OK. A wizard is launched to guide you through the project definition phase.
Creating a Java Card Applet

When you choose to create a Java Card Applet:

1. The first page of the GemXpresso RAD III project creation wizard appears:

![Figure 3 - The Applet Project Wizard Window, Step 1](image)

Enter the following information:

- **Type.** Choose whether to create a basic Java Card applet (Java Card 2.1 compatible only), or a Java Card-OP applet.
- **Project.** Specify the directory in which to store the IDE's project file (.JPR or VEP) and other associated files, for example
  `installdir\examples\cardname\oppurse\oppurse.jpr`
- **Package name.** Enter the name of the package to which the applet is to belong, for example “com.gemplus.examples.oppurse”.

<table>
<thead>
<tr>
<th>TYPE</th>
<th>Java Card Open Platform Applet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project</td>
<td>C:\GemXpresso\GemXpresso.rad3\EXAMPLES\GXEP211_PK_61\OPPurse</td>
</tr>
<tr>
<td>Package name</td>
<td>com.gemplus.examples.oppurse</td>
</tr>
<tr>
<td>Package AID</td>
<td>A0 00 00 00 18 FF 00 00 00 00 00 03 00 00 01 01</td>
</tr>
<tr>
<td>Applet name</td>
<td>OPPurse</td>
</tr>
<tr>
<td>Applet AID</td>
<td>A0 00 00 00 18 FF 00 00 00 00 00 03 00 00 01 02</td>
</tr>
</tbody>
</table>
– **Package AID.** Enter the 5- to 16-byte AID of the package. Enter the value in either hexadecimal format or as a string enclosed in double quotes, for example, “6D 79 61 70 70 6C 65 74 5F 70” or “myapplet_p”.

**Tip:** To automatically convert from one format to another, select the AID, then right click and select Convert to string or Convert to hexadecimal.

– **Applet name.** Enter the name of the applet, for example “myapplet_a”.
– **Applet AID.** Enter the 5- to 16-byte AID of the applet. Enter the value in either hexadecimal format or as a string enclosed in double quotes, for example, “6D 79 61 70 70 6C 65 74 5F 61” or “myapplet_a”.

2. Click **Next** to display the second page of the project creation wizard:

![Figure 4 - The Applet Project Wizard Window, Step 2](image)

On this window, you specify the target card types onto which the applet is eventually to be loaded. Initially, the **Available cards** list shows all available target card types, depending on the version of GemXpresso RAD III you have purchased.

3. Select a card type from the **Available cards** list and click **Add** to move it to the **Selected cards** list. Repeat for all card types to be supported. Alternatively, click **Add all** to move all available card types to the **Selected cards** list.
4. In the **Output path** field, enter the root directory for converted packages that are ready to load. For example: "`installdir/examples/cardname/oppurse/oncard`". A directory is created automatically for each card type in the **Selected cards** list.

5. In the **GSE target list**, select one of the card types from the **Selected cards** list to be the default target when using the GSE GUI (card simulator) to perform interactive debugging of the applet within the IDE.

6. Click **Finish** to create the project.

### Creating a Java Card Library

A Java Card library is a set of classes, grouped in a package, that are available when building applets or client applications.

The library is created in the form of a JBuilder or VisualCafé project, to which you assign a package name and an AID. You then add the library classes to the project and convert the classes for the required target cards.

![Figure 5 - The Library Project Wizard Window, Step 1](image)

1. Specify:
   - The name of the JBuilder or VisualCafé **Project** file.
   - The **Package name** to which the library classes are to belong.
   - The **Package AID**. Enter the 5- to 16-byte AID of the package.
2. Click **Next** to display the second page of the wizard:

![Figure 6 - The Library Project Wizard Window, Step 2](image)

This window is the same as that of the applet creation wizard. See “Figure 4 - The Applet Project Wizard Window, Step 2” on page 5. Refer to steps 3 to 6, starting on page 5.

**Creating a Custom Project**

This project template creates an empty project into which you can insert your own components. The only information that needs to be specified is the project name. No skeleton code is generated. By default, the project is configured in the project settings to use the `gse_javacard_21.jar` library for interactive debugging of the applet. To debug the applet in the GSE with another card, change this setting accordingly. See “Launching the GSE” on page 60 for a full list of the simulation modes available.
Importing a GemXpresso Project

This project template allows you to create a new project in your IDE based on the information contained in a GemXpresso Project file (.GXP). GemXpresso Project files are created using the Project Editor (see “Chapter 2 - Using the Project Editor”). The following wizard window is displayed:

In the GemXpresso project file to import field, select the project file (GXP) that contains the information to import.

In the Java source files field, specify Java source files that are to be included in the project.

In the Library files field, add the libraries required for the card you are using to compile the project. These libraries are in the \installdir\resources\exportfiles\cardname directory.

Click Finish to create the JBuilder or VisualCafé project with the specified information.
Accessing Other GemXpresso RAD III Tools

When using VisualCafé or JBuilder, the GemXpresso RAD III tools are available from the GemXpresso RAD menu, displayed in the IDE’s menu bar:

![GemXpresso RAD Menu]

Figure 8 - The GemXpresso RAD Menu

<table>
<thead>
<tr>
<th>Menu Item</th>
<th>Refer to...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edit GemXpresso Project</td>
<td>“Chapter 2 - Using the Project Editor” on page 21.</td>
</tr>
<tr>
<td>Edit GemXpresso Deployment</td>
<td>“Chapter 7 - The Deployment Editor” on page 83.</td>
</tr>
<tr>
<td>Convert</td>
<td>“Chapter 3 - Using the GxpConverter” on page 29.</td>
</tr>
<tr>
<td>JCardManager</td>
<td>“Chapter 4 - Using the JCardManager” on page 37.</td>
</tr>
<tr>
<td>GSE</td>
<td>“Chapter 5 - Using the GSE” on page 57.</td>
</tr>
<tr>
<td>About</td>
<td>Displays version information for the plug-in.</td>
</tr>
</tbody>
</table>

Table 2 - Accessing GemXpresso RAD III Tools

The OpenCard Framework (OCF)

OCF is an open architecture and a set of APIs providing a common interface for card readers and smart card-based applications. OCF enables you to develop applets and client applications that work on a variety of different suppliers’ equipment and platforms.
The core OCF API features two main parts:

- The **CardTerminal** layer provides access to physical card terminals and inserted smart cards.
- The **CardService** layer deals with the wide variety of card operating systems in existence and the differing functions they offer.

The Gemplus implementation of OCF includes:

- An open platform (OP) **CardService** that implements the OP 2.0.1 specification, used for managing a Java Card-OP application.
- Separate **CardTerminal** packages for individual GemXpresso cards and the GSE.

“Figure 9 - Gemplus Services in the OpenCard Framework” illustrates the architecture of the Gemplus OCF components:

For more information about the OpenCard Framework, refer to [http://www.opencard.org](http://www.opencard.org) and in particular the “OpenCard Framework 1.2 Programmer’s Guide”, which can be downloaded from the site.
Configuring OCF

GemXpresso RAD III uses an OCF configuration file, opencard.properties, to activate and configure smart card readers and the GSE. The default opencard.properties file is located in:

`installdir/conf`

The GemXpresso RAD III tools read this file to determine the type of card reader installed and the communications port on which it is installed.

The GemXpresso RAD III installation program identifies your current card reader type and the communications port to which it is connected (COM1, COM2, COM3 or COM4), and updates the opencard.properties file appropriately.

Use a text editor to modify the opencard.properties file. When editing the file:

- Insert a hash symbol ‘#’ at the beginning of a line to comment out the line.
- Each entry has the form `OpenCard.option = value(s)`. To continue a line, for example, to specify several values, insert a backslash ‘\’ at the end of a line to indicate that the option continues onto the next line.

The most important entries in the opencard.properties file are as follows:

```
OpenCard.services = \\
com.gemplus.opencard.service.gse.GSECardServiceFactory \\
com.gemplus.opencard.service.gse.PSECardserviceFactory \\
com.gemplus.opencard.service.gse.PSECardserviceFactory \\
com.gemplus.opencard.service.op.vop.vop200.CardServiceVOP200Factory \\
com.gemplus.opencard.service.op.vop.vop211.CardServiceVOP211Factory
```

Necessary to obtain GSE-specific commands.

```
# - Terminals configuration - #

OpenCard.terminals = \\
com.gemplus.opencard.terminal.GemplusRadCardTerminalFactory |Simulator|SOCKETJC21SIMULATOR|127.0.0.1:5000 \\
com.gemplus.opencard.terminal.GemplusRadCardTerminalFactory |gcr410_com1|GemPC410|COM1
```

Note: The GemPlusCardTerminalFactory, Pscs10CardTerminalFactory, and CASECardTerminalFactory options are mutually exclusive. Comment out the options that do not apply.
Controls communications with a serial card reader using the COM API. In GemXpresso RAD III, applications communicate with OCF-compliant terminals by means of a general-purpose communications library, comm.jar.

# com.ibm.opencard.terminal.pcsc10.Pcsc10CardTerminalFactory|mygcr|GemPC410|COM1

Controls communications with PC/SC card readers (see “Accessing PC/SC Terminals” on page 13).

########################################
# - Only for Gemplus Card Terminal #
########################################
Uses_Standard_ISO_TPDU = false

Set to “false” to avoid an exception when a card sends a response containing no data during an APDU exchange that matches Case 4 of the ISO 7816-4 standard.

##############################################
# - TRACE configuration - #
##############################################
OpenCard.trace.eventsOnly = true

When set to “false”, detailed trace messages are sent to the console.

###################################################
# - Do NOT remove entry com.gemplus.tools.gemxpresso.pilot:6,
# it is necessary to obtain an APDU trace in the JCardManager.
###################################################
OpenCard.trace = com.gemplus.tools.gemxpresso.pilot:6

Trace the package com.gemplus.tools.gemxpresso.pilot:6. This package displays messages relating to card services in JCardManager’s Messages window and allows ATF and SCR scripts to be recorded. You can also trace messages from other packages (for example, remove the comment from the following line to trace the com.gemplus.opencard.service.op package).

# com.gemplus.opencard.service.op:8

**Example: Using a Card Reader and the GSE**

For example, if you have connected a GemPC410 card reader on the serial port COM2 and want to use the GSE as well as the card reader, the opencard.properties file would contain:

```
OpenCard.terminals = \
com.gemplus.opencard.terminal.GemplusCardTerminalFactory  
|Simulator|SOCKETJC21SIMULATOR|localhost:5000 \ 
com.gemplus.opencard.terminal.GemplusCardTerminalFactory  
gcr410_com1|GEMPC410|COM2
```

---

12
**Example: Using the GSE Only**

If you want to use only the GSE, comment out unwanted entries with a hash ("#") symbol. The opencard.properties file would then contain:

```properties
OpenCard.terminals = \\
com.gemplus.opencard.terminal.GemplusCardTerminalFactory
    |Simulator|SOCKETJC21SIMULATOR|localhost:5000
# com.gemplus.opencard.terminal.GemplusCardTerminalFactory
[gcr410_com1]GemPC410|COM2
```

**Accessing PC/SC Terminals**

PC/SC (personal computer/smart card) is a native layer for communication between an application and a smart card reader.

You can access PC/SC-compliant terminals in GemXpresso RAD III by using a special "bridge" between OCF and PC/SC. The PC/SC software bridge is made up of:

1. A Java library file, `pcsc_wrapper.jar`
2. A native Windows library, `Ocfpcsc1.dll`, which supports the T=0 and T=1 communications protocols.

**To verify that the PC/SC service is running:**

Under Redhat Linux and Sun Solaris, RAD III uses PC/SC Lite. For details on installation and configuration, go to the website [www.linuxnet.com](http://www.linuxnet.com).

For Windows platforms, proceed as follows:

1. Ensure that PC/SC is installed and running on the local workstation, that is, the PC/SC Smart Card Resources Manager service has been started.
2. Check that a PC/SC-compliant card reader has been correctly installed, that is, the PC/SC reader driver has been activated. The GemXpresso RAD III installation program detects whether PC/SC is running on the machine and automatically configures any PC/SC-compliant card readers it detects to run under PC/SC.

**Note:** For further details about how to perform steps 1 and 2 under Windows' platforms, refer to “Appendix E - Checking PC/SC Installation”.

3. Add the following line to the opencard.properties file:

```properties
OpenCard.terminals = \\
    com.ibm.opencard.terminal.pcsc10.
Pcsc10CardTerminalFactory|mygcr|GemPC410|mode
```

Where `mygcr` is your chosen identifier of the card reader, GCR410P is the card reader type, and `mode` can be either of the keywords SHARED or EXCLUSIVE.
If you do not know what type of card reader is being used, leave the line after `Pcsc10CardTerminalDFactory` empty. All installed card readers are then detected. For example:
```
OpenCard.terminals = \
```

**Locating `opencard.properties` in Client Application Code**

If you are using a batch file to launch your client application, the location of the `opencard.properties` file can be specified in the client application code using the `user.dir` Java property. For example:
```
// Define c:\gemplus\gemxpresso.rad2\custom
private String customDir = "c:" + File.separator + "gemplus" +
    File.separator + "gemxpresso.rad3" + File.separator + "conf";
\nprivate String customDir = "c:" + File.separator + "gemplus" +
    File.separator + "gemxpresso.rad3" + File.separator + "conf";
```

```
// default constructor
public OPPurseClient()
{
    // set the current directory to this sample one
    java.util.Properties p = System.getProperties();
    p.put("user.dir", customDir);
    System.setProperties(p);
}
```

**Developing Projects Without Templates**

If you are creating a custom Java Card project, are not using the GemXpresso RAD III project templates, or are not using an IDE at all, you need to set up the development environment for applets and client applications manually.

**Configuring the Applet Development Environment**

This involves specifying:
- The location of Class files
- The class containing the `main()` method

This is described in the sections that follow.
Class Files
To compile an applet, the following imported packages are required by the GemXpresso RAD III environment:

- The GemXpresso GSE library, held in installdir\lib. These files are named gse_cardname.jar
  Where cardname is the name of the target card profile you have purchased, for example: gse_gxp211_pk.jar for the GemXpresso 211 PK card profile, or gse_javacard_21.jar for the generic JavaCard 2.1 compatible card profile.
  This library contains the following packages:
  - javacard.framework
  - visa.openplatform
  - javacard.security
  - javacardx.crypto

- The GemXpresso Cryptography libraries, cryptix-jce-api.jar, cryptix-gemxpresso.jar (supporting full and limited cryptography), and cryptix-gemxpresso-is.jar (supporting limited cryptography only).

Main Class
An applet loaded in a card does not need a main class. To enable interactive debugging with the GSE, however, you can specify the name of the GSE’s main class in the project settings:

    com.gemplus.javacard.gse.Simulator

Note: For certain JDKs, you need to use the “-classic” compiler option to enable step-by-step debugging of the code.

See “Chapter 5 - Using the GSE” for details.

Configuring the Client Application Development Environment
Client applications can be developed in tandem with the applet, for example, as part of the same project. However, the client application’s classes must be in a separate package in order to verify, convert and load the correct Java Card applet package.

Communication between a client application and a Java Card applet must take place using opencard framework (OCF) protocols.
Imported Packages

To compile, run and debug a client application in the GemXpresso RAD III environment, the following imported packages are required:

- base-core.jar
- base-opt.jar
- gemplus-terminals.jar
- gemplus-rad-terminals.jar
- gemxpresso-filemanager.jar
- gemxpresso.jar
- gemplus-util.jar
- gxploader.jar
- gxputil.jar
- vopcardservice.jar
- cryptix-jce-api.jar
- cryptix-gemxpresso.jar (supports full and limited cryptography)
- cryptix-gemxpresso-is.jar (supports limited cryptography only)

**Note:** If you are using VisualCafé, the global CLASSPATH variables are included in the sc.ini file, which sets the CLASSPATH for every project. If this information is not included in the sc.ini file, CLASSPATH variables must be manually added to each new project. Note that in certain cases, this can cause anomalies in VisualCafé.

Main Class

The client application’s main class should be specified in the IDE’s project settings. This enables interactive debugging when the applet associated with the client application is loaded into the GSE. See “Chapter 5 - Using the GSE” for details. Refer to the GemXpresso RAD III Version 3.2 Getting Started for an example based on the supplied OPPurse client application and applet.
Development Guidelines

Java Language Restrictions

The memory constraints of Java Cards mean that only a subset of the Java language can be used when developing Java Card applets. For example:

- You can create dynamic objects, invoke methods, and use interfaces, but there are some restrictions on the use of data types and arrays. The Java Card VM does not, for example, support the char, float, double, and long data types, and only one-dimensional arrays can be used.
- The card’s EEPROM is used to store objects. By default, these objects are persistent—classes need only be loaded and initialized in the JCVM once. The contents of EEPROM are preserved when the card is without power, so the card can store data permanently.
- There are no garbage collection classes in Java Cards. An object for which memory is allocated persists in memory, therefore, even after references to it are dropped. Memory is only recovered when a package on the card is deleted.

For a complete description of the limitations of the Java language in Java Card programming, see Sun’s Java Card 2.1 Virtual Machine Specification.

Applet Class and Method Implementations

For complete information about developing Java Card applets, refer to the Java Card Technology for Smart Cards: Architecture and Programmer’s Guide from Sun Microsystems. You should also review the source code of the samples applets supplied with GemXpresso RAD III in the installdir\examples directory.

The most important points to consider are as follows:

- All your applet’s classes must belong to one package. For example:

  ```
  package com.gemplus.examples.oppurse;
  ```

- Java Card applets must import from the following package:

  ```
  import javacard.framework.*;
  ```

- Java Card-OP applets must additionally import the following package:

  ```
  import visa.openplatform.*;
  ```

- The following packages provide optional utility methods:

  ```
  import java.lang.*;
  import java.security.*;
  import javacardx.crypto.*;
  ```
- An applet must override the `javacard.framework.Applet` class:
  ```java
  public class OPPurse extends javacard.framework.Applet {
  }
  ```
  Refer to the HTML-based Javadoc documentation for a listing of the methods contained in this class.

- An applet must implement the `install()` method of the base Applet class:
  ```java
  public static void install(byte[] installparam, short offset, byte length) throws ISOException {
  }
  ```
  The `install()` method should call the applet's constructor to perform any necessary initialization:
  ```java
  public static void install(byte[] installparam, short offset, byte length) throws ISOException {
      // applet instance creation with the initial balance
      new OPPurse(installparam, offset, length);
  }
  ```

- The applet’s constructor should be a protected method to ensure it can only be called by the applet’s `install()` method:
  ```java
  protected OPPurse(byte[] buffer, short offset, byte length) {
      // initialization
  }
  ```

- An applet must also override the `process()` method of the base `Applet` class:
  ```java
  public void process(APDU apdu) throws ISOException {
  }
  ```
  Within this method, define all the APDU commands supported by the applet. Refer to “Appendix A - Sending and Receiving APDUs” for details on APDU commands and how to code APDU commands and responses.
Developing Project Code

- The base `Applet` class provides `select()` and `deselect()` methods that are called by the JCRE to select or deselect the applet, respectively. Override these methods to provide any necessary data initialization or cleanup whenever the applet is selected or deselected in this way. For example:

```java
public void deselect()
{
    // reset security if used.
    reset_security();
    return;
}
```

See “Appendix B - Sample Code” for a complete listing of the supplied OPPurse applet’s code.

Next Steps

Once successfully compiled and debugged, your applet’s Class files are ready to be converted to a format that can be loaded onto a card or the card simulator. The first step in this process is to use the Project Editor to create a project file that contains all the necessary conversion information. The Project Editor is described in “Chapter 2 - Using the Project Editor”.

Using the Project Editor

Overview

Converting an applet’s Class files into a format ready for loading onto a Java Card is a two-step process:

1. You first use GemXpresso RAD III’s **Project Editor** to gather together all the information necessary about the conversion environment, such as the location and identity of the Class files to convert, and the card types to generate files for. This information is saved in a project file (*.GXP).

2. You then pass the project file to GemXpresso RAD III’s **GxpConverter** tool, which generates loadable files for each targeted card type. This process is described in “Chapter 3 - Using the GxpConverter”.

“Figure 10 - The Project Editor and GxpConverter” illustrates the process:

![Figure 10 - The Project Editor and GxpConverter](image-url)
Starting the Project Editor

You can access the Project Editor in a number of ways:

- From within the JCardManager:
  Choose **File > Open Project**.

- From within the Gemplus plug-in for the VisualCafé or JBuilder IDE:
  Choose **GemXpresso RAD > Edit GemXpresso Project**.

The Project Editor Window

When you start the Project Editor, the following window is displayed:

![The Project Editor Window](image)

Figure 11 - The Project Editor Window
The Project Editor window is divided into three main areas:

- The upper area of the window is used to specify the general project environment applicable to all target card types, including the name of the package to convert.
- The Default Applet AIDs box allows you to choose which applets in the specified package are to be converted and to specify the default AID for each applet in the specified package.
- The Target cards for this conversion area of the window contains a property sheet for each target card type supported by the version of GemXpresso RAD III you have purchased, for example, GemXpresso 211 (GXP211) or GemXpresso 211 PK IS (GXP211_PK_IS).

Creating a New Project

Specifying Project Information

In the project environment area of the Project Editor window, enter the following information about the package containing the applet or applets to be converted:

- **Project Name.** The name of the project file that is to hold all the information specified in the Project Editor window. If you entered the Project Editor from the JBuilder or VisualCafé plug-ins, a default project file name is created for you, based on the name you chose for your project. If you entered the Project Editor by launching the batch file, you need to specify a project file name:
  - Click the browse button next to the **Project Name** field.
  - Select the directory in which to store the file. All other directories you specify on the Project Editor window will be written to the project file as relative paths from the directory you choose.
  - Type the name of the project file, without an extension.
  - Click **OK** to create the project file.

The name of the project file, with a “.GXP” extension, is displayed in the **Project name** field.

- **Class directory.** The root directory of the project’s Class files. Click the browse button and select the directory containing the root of the Class files, then click **OK**.

- **Package name.** The fully-qualified name of the package containing the applet or applets to convert. For example, com.gemplus.examples.OPPurse.
The following figure shows how to specify the **Class directory** and **Package name** for an example **Class file**:

![Figure 12 - Determining the Class Directory and Package Name](image)

The Project Editor inspects the Class files in the specified directory and displays information relating to any applets it finds in the **Default Applet AIDs** field. For example:

**Figure 13 - Default Applet AIDs**

- **Package AID**. Enter 5- to 16-bytes for the value of the package AID. You can separate each byte with a space or colon, or leave no space between bytes. Press Enter to confirm entry of the AID. The Project Editor automatically reformats the package AID to show a single space between each byte.

- **Version**. The major and minor version number of the package. The Java Card specification recommends that you assign the first implementation of a package a major version of 1 and a minor version of 0 and that you increment the major or minor version number by 1 whenever the package is modified.

The **Default AIDs** list displays the following information about each applet found in the specified package:

- Whether the **Class name** displayed contains an applet to be converted for the specified target card types.
- The name of the applet’s **Class file**.
- The default AID of the applet.
To specify a default AID for an applet:

Click in the Default applet AID field for that applet and type a 5- to 16-byte AID.

You can now proceed to configure conversion settings for each target card type.

Configuring Target Card Types

When you first start the Project Editor, a tab is displayed for each card type supported by your version of GemXpresso RAD III. To add or remove card types:

1. Click Select Cards. The Cards Selection window is displayed:

![Figure 14 - The Cards Selection Window](image)

2. Use the Add, Add All, Remove, and Remove All buttons to position only the required target card types in the Selected cards list. Card types you move into the Available cards list are not deleted but remain available for future selection.

3. Click OK. The Project Editor window displays a tab for each card type in the Selected cards list.
Configuring Conversion Settings for a Card Type

To configure conversion settings for a card type:

1. Click the card’s property sheet.
2. To temporarily disable conversion for this card type, select **Disable conversion for this target**.
3. Specify the export directories. These are the directories in which the converter subsequently places the export files (.EXP). Default card export directories are automatically added. Click **Add** to add a custom card directory to the **Linked Libraries paths** list. Click **Remove** to remove a directory from the list.
4. In **Output Path**, specify the name of the directory where the converted files for this card type are to be placed. For example "installdir\examples\cardname\OPPurse\oncard\". Click the browse button to select an existing directory. If you specify a directory that does not exist, the Project Editor creates the directory.
5. If the target card type supports the 32-bit Integer data type, select **Supports Integer Type**.

Generating the Project File

When you have configured all information on the Project Editor window, click **OK**. The following files and directories are created:

- A GemXpresso project file (.GXP).
- The specified output directories for each card type, if they do not already exist.

See “Chapter 3 - Using the GxpConverter” for details on how to generate the export files for each card type using the Project File you have created.

Opening an Existing Project

To work with a project file that already exists:

1. Click the browse button next to the **Project Name** field and browse to the directory containing the previously created project file.
2. Select the project file and click **OK**. The Project Editor window fields are filled with the previously stored values.

Modify any of the information displayed, as described in the next section. When you click **OK** to re-generate the project file, any values you have changed are automatically updated.
Next Steps

The Project Editor does not actually perform file conversion. For that, you use the GxpConverter utility, which takes the information contained in the Project file and generates any necessary files. See “Chapter 3 - Using the GxpConverter” for details.
Using the GxpConverter

Introduction

An applet’s Class files must be converted to a Java Card-compatible CAP file format before it can be loaded onto a card, or a SAP file format for loading into the GSE. Because CAP files must be formatted differently for each type of card for which conversion is planned, generating CAP files has previously been a somewhat lengthy and complex process.

GemXpresso RAD III introduced the GxpConverter tool to facilitate this conversion process. With just a single command, you can now generate all the load files necessary for all targeted card types, including the GSE.

The GxpConverter tool requires as input a Project file previously created using the Project Editor. See “Chapter 2 - Using the Project Editor” for details on creating a Project file. The Project file contains information about the location and identity of the Class files to convert as well as details about each of the card types for which CAP files are to be generated.

The GxpConverter calls Sun’s Converter utility for each targeted card type in turn, and generates a number of files as output:

- A Java archive (JAR) file containing the CAP files (.CAP)
- Export files (.EXP)
- SAP files (.SAP)
- JCA files (.JCA)

The purpose of these files is explained in the sections that follow.

Note: You can specify a converter other than Sun’s if you wish. See “Specifying a Different Converter” on page 33.
“Figure 15 - The Conversion Process with GxpConverter” illustrates the conversion process using GxpConverter.

**CAP Files**

A CAP file is a portable representation of a Java Card applet. CAP files are in a binary format capable of being loaded directly onto a Java Card, and are used by the card’s Java Card Virtual Machine to execute the applet.

For a complete description of the Java Card CAP File format, see Sun’s *Java Card 2.1 Virtual Machine Specification*.

You can use the CapDump utility (see “Chapter 8 - The CapDump Utility”) to view the contents of a CAP file.

**JAR Files**

Because a single package can generate many CAP files, the Java Card specification recommends that CAP files be stored in a Java Archive (JAR) file before being loaded onto a card. The JAR file is stored in a directory called `javacard` immediately below the package’s directory. For example, the JAR file for the package `com.gemplus.examples.OPPurse` is placed in the subdirectory `com\gemplus\examples\OPPurse\javacard`.

Figure 15 - The Conversion Process with GxpConverter
To view the contents of a JAR file, you can use tools provided for this purpose by IDEs such as JBuilder or VisualCafé. Because JAR files use the ZIP compression algorithm, you can also view their contents using a tool such as WinZip.

**EXP Files**

An export file contains the public interface information for a package of Classes. Export files are not actually loaded onto the card. However, the information in an export file is critical to the operation of the off-card portion of the Java Card Virtual Machine. They are used, for example, when adding classes to a package that has already been uploaded to a card.

For a complete description of the Java Card export file format, see Sun’s *Java Card 2.1 Virtual Machine Specification*.

**SAP Files**

SAP files are a Gemplus-proprietary version of the JAR file format containing CAP files. They cannot be uploaded to a real card, but once loaded into the GSE environment, they accurately simulate an applet in almost all other ways.

**JCA Files**

A JCA (Java card assembler) file is a textual representation of the contents of a CAP file. A JCA file can be useful for testing and debugging purposes, and you can subsequently convert the JCA file to a CAP file using a Java Card Development Kit utility. The GxpConverter tool generates a JCA file for each specified target card type.

**Starting the GxpConverter**

The GxpConverter can be launched:

- From within the JCardManager
- From within a plugin

**Running GxpConverter from within JCardManager**

To convert files from within the JCardManager:

1. Open the Project Editor window with **File > Open Project**.
2. Create or open the Project file, then click **OK**.
3. Choose **File > Convert** to call the GxpConverter tool to begin conversion.
Running GxpConverter from within a Plug-in

To convert files from within the JBuilder or VisualCafé plug-in:

1. Open the project containing the files to be converted.
2. Compile the project to ensure the Class files are up-to-date.
3. Choose GemXpresso RAD > Edit GemXpresso Project to open the Project Editor window and define the contents of the project file for this package. The Project Editor is described in “Chapter 2 - Using the Project Editor”.
4. Choose GemXpresso RAD > Convert. The Convert window is displayed:

![Convert Window](image)

**Figure 16 - The Convert Window**

5. For each applet in the current project, select **Convert** to specify that files are to be converted for this applet.
6. To calculate the number of bytes that the converted package will occupy on the card, select **Compute size of the package**. The calculated value is displayed in the IDE’s Messages window.
7. To select target card types for this conversion, click Targets. The following window is displayed:

![Figure 17 - Choosing Target Card Types](image)

Use of this window is described in “Configuring Target Card Types” on page 25. Click OK when all target card types are configured.

8. Click Convert to begin conversion. The output from the GxpConverter utility is displayed in the Messages window of the JBuilder or Visual Café IDE. Files are generated.

**Specifying a Different Converter**

By default, the GxpConverter uses Sun’s Java Card 2.1 Converter tool to convert files. You can change the converter used by modifying the card type’s corresponding card profile, then specifying this card profile in the GxpConverter command line.

**To specify a different converter:**

1. Edit the card profile for the card type you are converting. Card profiles are located in the `installdir\resources\cardprofile` directory. For example, `GXP211_PK.gsc` is the card profile for GemXpresso 211 PK cards.
2. A card profile is a structured XML file. Locate the following section of the file:

```xml
<ConversionSettings name="card">
  <Converter name="JC2.1 SUN Converter (version 1.0)"
    className="com.sun.javacard.converter.Converter"
    classPath="../../lib/converter/sun/converter.jar">
    <Parameter name="classes_directory" type="">-classdir </Parameter>
    <Parameter name="export_path" type="">-exportpath </Parameter>
    <Parameter name="output_directory" type="">-d </Parameter>
    <Parameter name="out_files" type="">-out JCA EXP </Parameter>
    <Parameter name="verbose" type="">-verbose </Parameter>
    <Parameter name="applet_aid" type="">-applet </Parameter>
    <Parameter name="applet_name" type=""></Parameter>
    <Parameter name="package_name" type=""></Parameter>
    <Parameter name="version" type=""></Parameter>
  </Converter>
</ConversionSettings>
```

3. Modify the **name**, **classname**, **classpath**, and **parameter** values as necessary and save the card profile.

4. Use the **-cardprofiledir** parameter when running the GxpConverter to specify the location of the modified file.

Results of Conversion

For each target card type identified in the Project file, the GxpConverter tool creates the following in the directory `outputpath\packagepath\javacard`:

- A **shortpackagename**.JAR file containing all the applet’s CAP files.
- A **shortpackagename**.JCA file.
- A **shortpackagename**.EXP file.
- A **shortpackagename**.SAP file

Where:

- `outputpath` is the directory you chose on the Project Editor window as the **Output Path** for the card type.
- `packagepath` is the fully-qualified path of the package being converted, for example, `com\gemplus\examples\oppurse`.
- `shortpackagename` is the name of the last part of the package name, for example “oppurse” for the package “com.gemplus.examples.oppurse”.
Next Steps

When converted files have been successfully created, you can:

- Upload the applet’s JAR file to a card using the JCardManager or the GxpLoader. See “Chapter 4 - Using the JCardManager” or “Chapter 6 - The GxpLoader” respectively.
- Upload the applet’s SAP file to the GSE using either the JCardManager or the GSE. See “Chapter 4 - Using the JCardManager” or “Chapter 5 - Using the GSE”.
- View and validate the contents of the generated CAP file using the CapDump utility. See “Chapter 8 - The CapDump Utility”.

Using the JCardManager

The JCardManager is a generic terminal-based client application. Using this tool, you can communicate with and test applets without needing to develop your own client application. The JCardManager can communicate with both real cards and the GSE.

The principal functions of the JCardManager are:

- Access to the GxpConverter to convert Class files to formats ready to be loaded onto a card or the GSE (card simulator).
- Managing the OCF communications layer.
- Access to the Deployment Editor to record and replay deployment files, allowing one-step uploading, installation, and selection of applets.
- Performing card/terminal authentication.
- Loading and installing applets into the card.
- Managing the cryptographic features of the card.
- Displaying and recording a trace of all activity between the program and the target.
- Displaying the contents of a card.
- Sending high-level APDU commands or user-defined commands to a target and observing the responses.
• Recording and play back of scripts to automate the issuing of sequences of commands.

![Figure 18 - The JCardManager](image)

**Launching the JCardManager**

The JCardManager is a GemXpresso RAD III tool that can be launched:

- From the Windows **Start** menu by selecting **Programs > Gemplus Applications > GemXpressoRAD III > JCardManager**.
- As a batch file, by executing **JCardManager.bat**
  
  This file is located in the following directory:
  
  `installdir\bin`

- From within the JBuilder or VisualCafé IDEs by choosing **GemXpresso RAD > JCardManager**.
The JCardManager Window

The main JCardManager window is divided into a number of areas as shown in “Figure 19 - The JCardManager Window”:

- The Cards list displays a list of all available target card types. Notice that, depending on the type of card you select, the appropriate list of supported commands is displayed in the Commands list.

- The Terminals list displays a list of all configured terminals in which cards are currently inserted. The GSE is included in the list if either the GSE or the GSE GUI is currently running. The list is updated dynamically whenever a card is inserted or removed from a terminal, and whenever the card simulator program is started or stopped.

- The Tools tab displays a list of available tools. This contains a utility for enciphering and deciphering data using either the 3DES or the DES algorithm.

- The Commands tabs display lists of commands supported by the currently selected target card.

- The parameters of the command currently selected in the Commands list are displayed in the Command Parameters area of the window.

- The Messages area of the window displays run-time and error messages, the output from commands and scripts, and status messages.

- The Script Toolbar contains icons allowing you to record and run (play back) scripts.
Managing Projects

You use GemXpresso RAD III’s Project Editor to gather together all the information needed to convert a package’s Class files to a format ready for uploading to a target card.

To create a new project:

Choose File > New Project

To open an existing project:

Choose File > Open Project or File > ReOpen Project (this option lists the more recently opened projects).

To edit a project:

Choose File > Edit Project

To close a project:

Choose File > Close Project

Refer to “Chapter 2 - Using the Project Editor” for full details on using the Project Editor.
Converting Class Files to CAP Files
To convert Class files identified using the Project Editor to one or more target card formats simultaneously, choose:

File > Convert
This command invokes the GxpConverter utility to generate CAP files for the card types selected using the Project Editor window. Refer to “Chapter 3 - Using the GxpConverter” for full details.

Managing the OCF Communications Interface
You can start, stop, and reset the OCF communications interface from within the JCardManager.

To start the OCF interface:
Choose Communication > Start OCF
To stop the OCF interface:
Choose Communication > Stop OCF
To reset the OCF interface:
Choose Communication > Reset OCF

Performing DES and 3DES Encryption and Decryption
With the Tools category selected, the available commands are DES and 3DES. These are tools for encrypting and decrypting data using the DES (Data Encryption Standard) and triple DES algorithms.

To perform DES or 3DES encryption/decryption:
1. Select the Tools tab on the JCardManager window.
2. Select DES or 3DES from the command list, depending on the algorithm that you want to use.
3. In Mode, select the action required: Cipher or Decipher.
4. In the Data in box, type the data to be ciphered or deciphered.
5. In the Chaining box, select the encryption/decryption algorithm:
   – ECB (electronic code book)
   – CBC (cipher block chaining)
If you choose CBC, type the eight-byte **ICV** (initial chaining vector) value required for CBC mode. When encrypting data, this should always be 8 bytes of binary zeros ("00").

6. In the **Key** box, for DES, type the eight-byte value for the **K1** secret key. In the case of triple DES, choose the number of keys (two or three) and enter the values for **K1**, **K2** and if applicable, **K3**. Enter values by typing them directly or by selecting them from the lists. Alternatively you can click the key button to the right of these boxes to generate random values.

**Note:** 3DES usually uses three different keys. The value of the **K3** key is assumed in this case to be equal to the **K1** key.

7. Click **Compute** to calculate the encrypted or decrypted value of the data.

The **Messages** area of the window displays the results, for example, when deciphering:

```
================================= 3DES
K1 = 12 34 56 78 90 12 34 56, K2 = 11 22 33 44 55 66 77 88, (K3=K1),
CBC mode : ICV = 11 22 33 44 55 66 77 88
data in = 44 35 9F D4 2C 8C DB 01 FF 29 22 91 60 3F F9 D5 5A AE 80 29 09 9E AA 02
data out = 55 6E 20 74 65 73 74 20 70 6C 75 73 20 6C 6F 6E 67 2E 2E 2E 80 00 00 00
```

**Creating a Key File Using the Key File Editor**

A set of files containing a default set of authentication keys are provided in the `installdir\resources\targets` directory. You can use GemXpresso RAD III’s Key File Editor to modify existing key files or to create your own custom key files.

**Launching the Key File Editor**

The Key File Editor is a tool that can be launched:

- As a batch file, by executing **KeyFileEditor.bat**
  
  This file is located in the following directory:
  ```
  installdir\bin
  ```

- From within the JBuilder or VisualCafé IDEs by choosing **GemXpresso RAD > Key File Editor**.
Using the JCardManager

The Key File Editor window appears, as shown in “Figure 20 - The Key File Editor Window”:

![Figure 20 - The Key File Editor Window](image)

**Using the Key File Editor**

To create a new key file:

1. Choose **File > New**.
2. Enter the 5- to 16-byte **Security Domain AID** of the target card’s default security domain (the Card Manager applet or a proprietary security domain applet). For example, enter `A0 00 00 00 18 43 4D 00` for the GemXpresso Lite Generic cards.
3. Enter the number of the **Default Key Set**.
4. Define the mother key.
5. Create the key sets.
6. Save the key file with **File > Save**.

To open an existing key file:

1. Choose **File > Open**.
2. Modify the key file as required.

To define the mother key:

1. Enter the **Name** of the mother key.
2. Choose the **Key Diversification** to derive static keys (NONE, VISA or VISA2). For more details on these options see “Appendix D - Key Diversification”.
3. Enter the Value of the mother key as a 16-byte hexadecimal number.

**Note:** You can also import a mother key from an existing key file by choosing **Mother Key > Import.**

**To create a key set:**
1. Choose **Key Set > Add.**
2. Enter the **Key Set Number.** It is important to ensure that all key sets within the same key file have unique key set numbers.
3. Click **OK.**
4. Click **Add** to create a new key value.
5. Choose the **Algorithm** to use, for example “DES-ECB”.
6. Type the **Key Number** and **Key Value**, then click **OK.**
7. Repeat the previous steps to add other keys to the key set.

**Note:** You can also import key sets from existing key files by choosing **Key Set > Import.**

---

**Creating a Key File Containing Custom Keys**

If you choose not to use the key file editor, you can also manually create your own key file and use this file to perform authentication.

**To create a key file:**
1. Create a new ASCII file using a text editor.
2. Enter the following information into the file:
   ```
   this_file_format=1
   aid.security_domain=hex/A0 00 00 00 03 00 00
   ```
   The value must be equal to 1 for all current versions of GemXpresso RAD.
   ```
   aid.security_domain=hex/A0 00 00 00 03 00 00
   ```
   The AID of the default Security Domain (the Card Manager applet or a proprietary security domain applet). You can enter the AID using either the hexadecimal format shown above, or in ASCII format (for example, “ascii/SYSTEM_A”).

---
Using the JCardManager

set_version=13
Where “13” is the default key set version number.

set_number=1
Specifies the number of key sets present in the file. The value set_number=0 indicates that the file contains no key sets (that is, authentication is by means of key diversification).

set1.version=13
Defines the index of the first key set version. If the value of set_number is greater than one, use “set2.version=” for the second key set, and so on.

set1.key_number=3
The number of keys in this key set.

set1.key1=des-ecb/CA CA CA CA CA CA CA CA 2D 2D 2D 2D 2D 2D 2D 2D
Value of the first key in the set.

set1.key2=des-ecb/2D 2D 2D 2D 2D 2D 2D 2D CA CA CA CA CA CA CA CA
Value of the second key in the set.

set1.key3=des-ecb/CA 2D CA 2D CA 2D CA 2D 2D CA 2D CA 2D CA 2D CA
Value of the third key in the set.

3. Save the file as installdir\resources\targets\filename.properties, where filename can be any name you choose.

Selecting a Target Card

Before beginning to send commands to a card, you need to select the terminal containing the card (an installed card reader or the GSE), and the card type:

1. Select the terminal containing the card from the Terminals list.

2. Select the target card type from the Cards list. If the list is empty, start the GSE (card simulator), or insert a card into the card reader.

   If you are using JCardManager with support for limited length cryptographic keys and the type of card you select supports full-length keys, a warning message to this effect appears when you attempt authentication on the card.

The list of commands supported by the selected card type is displayed in the Commands list.
Displaying the Contents of a Card

JCardManager allows you to display the current contents of a card or GSE context.

To display the contents of a card or the GSE:

1. Select the card type and terminal to use.
2. Perform authentication of the card or GSE by running the Authenticate command. See the GemXpresso RAD III Version 3.2 Command Reference guide for the selected card type, or the online help, for details.

The Card Explorer window is displayed:

The Card Explorer window displays two separate lists:

1. The Package View, containing:
   - The package AID of each package on the card.
   - For the currently selected package, the life cycle status and privileges granted for use of the package.
2. The Applet View, containing:
   - The applet AID of each applet installed on the card. The Card Manager applet is always displayed.
   - For the currently selected applet, the life cycle status and current privileges.

You can click the Refresh button at any time to update the display, for example, after having issued a command.
To copy the value of a package AID or applet AID, for example, to a field in the Command Parameters area, right-click a package or applet and select **Copy**, then paste the value into the required field.

**Sending Simulator Commands**

When the GSE is selected as the current target, the **Simulator Commands** tab displays a list of commands that you can send to the GSE. These commands, for example, request information about the current GSE environment or the current context loaded in the GSE.

**Note:** These commands operate at the Java Card level and are not Java Card-OP compatible. They can be used when the GSE is running in Java Card 2.1 “virtual” mode. See “Chapter 5 - Using the GSE” for details.

**GSE Get Card Type**

To obtain the current card type being simulated:

1. Ensure the **GSE** is selected as the current target.
2. Select the **GSE Get Card Type** command on the **Simulator commands** tab.
3. Click **Go**.

The type of card currently being simulated by the GSE is displayed in the Messages area of the window. For example:

```
=======================  Simulator Get Card Type
Current selected Simulator Type is GXP211
```

To change the type of card being simulated, refer to “Chapter 5 - Using the GSE”.

**GSE Java Card Load**

Before using this command, you must create a load file for the applet. For the GSE, this must be a .SAP file. Use the Project Editor and GxpConverter tools to create a SAP file.

This command can only be used with the GSE (card simulator) selected as the target—it cannot be used to load a file into a real card.

To send a Load command to the simulator using the Java Card protocol:

1. Ensure the **Simulator** is selected as the current target.
2. Select the **GSE Java Card Load** command on the **Simulator commands** tab.
3. Enter the **Applet AID** of the applet to be loaded.
4. Enter any **Install Parameters** of the applet to be loaded.
5. Click **Go**.
GSE Java Card Install

Before using this command, the applet to select must have been uploaded to the current simulator context, for example using the GSE Java Card Load command.

This command can only be used with the GSE (card simulator) selected as the target—it cannot be used to load a file into a real card.

To send an Install command to the simulator using the Java Card protocol:
1. Ensure the Simulator is selected as the current target.
2. Select the GSE Java Card Install command on the Simulator commands tab.
3. Click Go.

GSE Reset

This command resets all packages and applets on the current simulator context to their initial status.

To send a Reset command to the simulator:
1. Ensure the GSE is selected as the current target.
2. Select the GSE Reset command on the Simulator commands tab.
3. Click Go.

GSE Restore Context

This command loads a previously saved simulator context into the simulator. A simulator context (.SSD or .FSD) can be created using the GSE Save Context command or the GSE GUI.

To send a Restore Context command to the simulator:
1. Ensure the Simulator (GSE) is selected as the current target.
2. Select the GSE Restore Context command on the Simulator commands tab.
3. Select either Input FSD File to restore a full serialized data (*.FSD) context file, or Input SSD File to restore a system serialized data (*.SSD) context file, and enter or select the name of the context file to restore.
4. Click Go.
GSE Save Context

This command saves the current simulator context to a file. This makes it easy to preserve the contents of a simulated card and restore it later.

To save the current simulator context to file:
1. Ensure the Simulator (GSE) is selected as the current target.
2. Select the GSE Save Context command on the Simulator commands tab.
3. Select either Output FSD File to create a full serialized data (*.FSD) context file, or Output SSD File to create a system serialized data (*.SSD) context file, and enter or select the name of the context file to create.
4. Click Go.

Note: A simulator context file (.FSD or .SSD) can also be created using the GSE GUI.

GSE Select Applet

To select an applet that has previously been loaded and installed on the simulator:
1. Ensure the GSE is selected as the current target.
2. Select the GSE Select Applet command on the Simulator commands tab.
3. Enter the Applet AID of the applet to be selected.
4. Click Go.

Sending Commands to a Target

There are several ways of sending commands to a target:

- Using pre-defined commands available from the Command tabs. Each tab contains a category of commands supported by the currently selected card type. These commands are described in the online Command Reference guide for a particular card type.
- When the GSE is selected as the current target, using GSE-specific commands available from the Simulator Commands tab, as described earlier in this chapter.
- Issuing user-defined APDU commands.
- Running previously recorded scripts containing APDU commands. See “Using Scripts” on page 53.

The results of a command exchange are displayed in the Messages area of the window (runtime and error messages, or trace of an APDU exchange).
You can perform the following tasks using compatible commands from the **Commands** list (some commands are only supported by particular card types, and command parameters may differ from card to card):

- Perform card/terminal authentication
- Change or unblock the card’s PIN code
- Delete an applet from the card.
- Get data from an applet on the card
- Get the memory space of an applet on the card
- Install an applet on the card
- Store data on the card
- Store an encryption key on the card
- Select an applet on the card, using its AID
- Upload packages into the card
- Send an APDU command to the card

These commands are described in the *GemXpresso RAD III Version 3.2 Command Reference*.

**Managing Deployment Files**

A deployment file describes all the commands necessary, together with all associated parameters, to install an applet on one or more target card types. Running a deployment file simulates running the following commands in order:

1. **External Authenticate**
2. **Upload File into a Card**
3. **Install**
4. **Select**

Certain card types may not support all these commands, while the commands’ parameters may differ for other card types.

When creating a deployment file, you can choose to include only a subset of these commands (for example, if the applet is to be uploaded but not installed).
Creating and Editing Deployment Files

You create deployment files using the Deployment Editor tool. See “Chapter 7 - The Deployment Editor” for details. You can access the Deployment Editor directly from within the JCardManager.

To create a new deployment file:
Choose Deployment > New

To open an existing deployment file:
Choose Deployment > Open

To re-open a previously opened deployment file:
1. Choose Deployment > Reopen
2. Select one of the listed deployment files.

To edit an existing deployment file:
Choose Deployment > Edit deployment

To close the current deployment file:
Choose Deployment > Close deployment

Running Deployment Files

To open a deployment file and run the contents as a series of commands from within the JCardManager:
1. Choose Deployment > Deploy.
2. The Messages area of the JCardManager window displays the output from the commands described in the deployment file.

Note: You can also run a deployment file using the GxpLoader tool. See “Chapter 6 - The GxpLoader”.

To re-run the contents of a previously run deployment file:
1. Choose Deployment > Redeploy and choose one of the listed deployment files.
2. The Messages area of the JCardManager window displays the output from the commands described in the deployment file.
Managing the APDU Commands List

When you are using the Send APDU command, an Edit Command List button is displayed in the Command Parameters area of the window:

![Command Name](getBalance)

Figure 22 - The Edit Command List Button

Whenever you issue an APDU command name, the JCardManager automatically records the name of the command and its associated parameters and adds the information to a command names list.

These commands appear thereafter in the Command Name list. When you select a command from this list, the previously used parameters are automatically entered into the CLA, INS, P1, P2, Le and Lc fields.

If your applet accepts a number of custom APDU commands, you can save a lot of time by creating a custom APDU commands list.

To view the list of available APDU commands:

1. On the JCardManager window, click Edit Command List. The Edit Command List window is displayed:

![Edit Command List](GetBalance Debit Verify PIN Credit Wrong PIN)

Figure 23 - The Edit Command List Window
2. To delete any unwanted commands from the list, select a command and click **Remove**.
3. Click **Save AMF** to save the list of commands as a .AMF file.

**To subsequently reload a list of saved commands:**
1. On the JCardManager window, click **Edit Command List** to display the **Edit Command List** window.
2. Click **Load AMF** and select a List of Commands (.AMF) file.
3. Click **OK**.

**Using Scripts**

Scripts are an invaluable aid in using the JCardManager. Scripts let you record a sequence of commands issued and the results returned from the target in a file. You can then subsequently load and execute (play back) previously recorded scripts. Scripts can be executed incrementally, that is, one command at a time.

**Note:** For SCR and ATF scripts, command replay is not possible if a MAC (message authentication code) is being used for an APDU command, because MAC requires a secure channel to be in place. To prevent this, APDUs should be recorded without MAC for debugging purposes. Therefore:

- You must issue an **Authenticate** command before recording commands (which can then be replayed later).
- The **Authenticate** command that opens a secure channel must have the **No Security** parameter disabled.
You control the recording and playback of scripts using the Script toolbar:

![Script Toolbar](image)

**Figure 24 - The Script Toolbar**

### Supported Script Formats

The JCardManager is able to replay and store commands in the following file formats.

- The **SCR** format, records data sent to the card in an ASCII format defined by Sun Microsystems.
- The **ATF** (APDU trace format) format, which is a Gemplus format, contains the data sent to the card, the responses received from the card and the command name in an ASCII format.
- The **GXCF** format contains a list of stored commands in XML format. The aim is not to store all signatures and MAC information but to recalculate them each time.

**Note:** You can convert GCF scripts to GXCF format using a supplied conversion utility. See “Chapter 9 - The GCF to GXCF Converter” for details.

When recording a new script, you must manually add the extension corresponding to the script format you want to use to the file name. For example, to record a file in GXCF format, type `myscript.gxcf`.

When loading an existing script for playback, you can choose to load script files recorded in any of the above formats.

**Note:** You cannot use ATF or SCR scripts to record or play back the commands on the **GSM** property sheet of the JCardManager.
Recording a Script

To record a script:
1. Click the Record button on the Script toolbar. The Script Options window is displayed.
2. Type the name of the file to save the script in or click the browse button to search for an existing script file to use.
3. Click OK. If the name of the script file exists already, you are asked to confirm that you want to overwrite the file contents.
4. Recording begins. The message “Start recording...” is displayed in the Messages area of the window.
5. Set up the parameters for and issue any commands you wish to record in the script file. All commands and the commands’ parameters are recorded in the script file. To stop recording, click the Stop button on the toolbar.

Replaying a Script

To replay the contents of an existing script:
1. Click the Play button on the toolbar. The Script Options window is displayed.
2. Type the name of the script file to replay or click the browse button to search for the file.
3. If you recorded the script in ATF format, you can select Compare APDU Responses to compare the responses the target makes to APDU commands in the script with the responses recorded in the script itself. Select Stop if different responses found to stop playback of the script if a different response is found. For example, if the expected response to an APDU was 0x9000 (successful) but during playback of the script the card responds with an error code, playback of the script stops.
4. Click OK to start replaying the script. The message “Start playing...” is displayed in the Messages area of the window, followed by the commands contained in the script and the responses from the target.

Stepping Through a Script

During “normal” playback of a script file, all commands are replayed without interruption unless:

- One of the commands fails to complete successfully.
- The Compare APDU Responses and Stop if Different Responses Found options are selected and the response code received does not match the response code recorded in the script. This is only possible if the script was recorded in ATF format.
To observe the target card’s responses to each command before the next command is issued:

1. Load the script as before, but click the Step by Step button on the toolbar. Proceed as for normal replay of a script.
2. The first command in the script is issued and the response displayed in the Messages area of the window. Press any key to issue the next command and display the corresponding response. Continue like this until all commands have been issued, or click the Stop button to stop playback.

Resetting a Script

When a script has completed playback, you can replay the script without needing to re-open the script file.

To reset a script:

1. Click the Rewind button on the toolbar.
2. Click the Play or Step by Step button to begin playback of the script.

Recording a Trace File

The JCardManager writes all commands issued and responses received to the Messages area of the window. You can save the contents of the Messages area as a Trace file to allow you to subsequently review the exact sequence of events at some later time.

To record a trace file:

1. Click the Erase current trace button on the script toolbar to the right of the Messages area of the window, or choose Tools > Trace > Erase the trace. This clears the Messages area of the window of any existing text.
2. Perform whatever actions you want to record in the trace file: issue commands, run script files, etc.
3. When finished, click the Save current trace button on the toolbar, or choose Tools > Trace > Save the current trace.
4. Specify the name of the trace file. You are free to choose any name and extension. The contents of the Messages window are written to the trace file.
Introduction

The Gemplus simulation environment (GSE) is a tool for simulating smart cards. Unlike “generic” card simulators, GemXpresso RAD III’s GSE allows you to simulate specific types of smart cards, including the GemXpresso card families. For test purposes, the GSE is also capable of simulating the JavaCard 2.1 standard in a special “virtual” simulation mode.

The GSE provides a much faster and simpler way of loading and installing applets than a real card, and is therefore particularly useful during the development and test phases of projects. Once testing is complete, the applet can be loaded into a real card for final testing and qualification. No changes to the code are necessary when switching from using the GSE to a real card.
GSE Architecture

The GSE architecture and components are shown in “Figure 25 - GSE Architecture and Environment”:

To ensure applet portability and interoperability, a standard Java VM, fully compliant with the JDK 1.2.2 from Sun Microsystems, forms the runtime basis of the GSE. This allows the GSE to be launched from within any JDK 1.2.2-compatible programming environment or IDE.

Client applications can be the JCardManager (a generic client application) or custom-made client applications.

Applets are loaded into the GSE using standard Java Card or Java Card-OP installation techniques, for example, using the JCardManager to upload, install, and select an applet.

All communications between client applications and applets loaded in the GSE are then achieved using APDU commands, which are processed by the intermediate Open Card Framework (OCF) software layer.
The GSE software components consist of:

- A specific socket server on TCP/IP (transparent to the user), which is the communication entry point for loading applets and APDU commands, through the software components of the OpenCard Framework.
- The Simulator Java card runtime environment (JCRE) is the simulator core, implementing the API functions.
- The Java Card 2.1 and OP 2.0.1 APIs are the standard API layers used by Java Card and Java Card-OP applets.
- The GSE GUI provides an easy-to-use graphical user interface for viewing details of the loaded packages and applets as well as tracing the communications between client applications and loaded applets.
- The Open API is a Gemplus-proprietary API used to communicate with the graphical user interface component of the GSE.

**Test Procedure Using the GSE**

The process of testing within the GSE consists of the following major steps:

1. **Preparation and Development**
   Applets and client applications must first be prepared, consisting of:
   - Project creation.
   - Developing code and compiling applets within the established project.
   This is described in “Chapter 1 - Developing Project Code”.

2. **Configuring OCF**
   The OCF software layer handles all communications between client applications and applets. Configuration of this software layer is achieved by editing the OCF configuration file, `opencard.properties`. This is described in “Configuring the OCF opencard.properties File” on page 60.

3. **Launching the GSE**
   You can launch the GSE using one of the possible methods. The choice of method affects the subsequent debugging possibilities that are available. Details of the different launch methods are given in “Launching the GSE” on page 60.

4. **Loading the Applet**
   Load the applet(s) into the GSE. This is described in “Loading Applets into the GSE” on page 67.
5. **Debugging**

Use the debugging capabilities of an IDE, such as JBuilder or VisualCafé, in conjunction with the GSE (for the modes that allow debugging).

### Configuring the OCF `opencard.properties` File

The GSE requires a specific OCF `CardTerminal` package, which implements a GSE-proprietary protocol. The following declaration must be included in the `opencard.properties` file:

```
OpenCard.terminals = Factory|Simulator_name|Type|Network_address
```

Where:

- **Factory**
  Is `com.gemplus.opencard.terminal.GemplusRadCardTerminalFactory`

- **Simulator_name**
  Is a user-defined name for the simulator. You can change this to any name (without a space in the name). The same name must be declared in the client application code (see “Method 1: Client Application Code Launch” on page 62).

- **Type**
  Is `SOCKETJC21SIMULATOR`, which cannot be changed.

- **Network_address**
  Is the network address in the form `Address:Port`, for example: “127.0.0.1:5000”. You can edit this entry, for example, if the port 5000 is already in use on your system. The same port number must be declared in the client application code (see “Method 1: Client Application Code Launch” on page 62).

For example:

```
OpenCard.terminals=\com.gemplus.opencard.terminal.GemplusRadCardTerminalFactory|Simulator|SOCKETJC21SIMULATOR|127.0.0.1:5000
```

### Launching the GSE

The GSE can be launched in various ways:

- **Method 1: Client Application Code Launch**
  Code contained in the client application launches the GSE. Both the client application and applet can be run in interactive debug mode from within an IDE. The client application’s main class is specified in the IDE.

  For more details, see “Method 1: Client Application Code Launch” on page 62.
• **Method 2: IDE Launch**

The IDE’s project parameters are modified to specify the GSE as the main class. The applet can then be run in interactive debug mode from within the IDE. The client application can be a user-supplied client application, or the JCardManager acting as a generic client application.

For more details, see “Method 2: IDE Launch” on page 62.

• **Method 3: Command Line Launch**

The GSE is launched manually by a batch file or command line. The client application is also started using a batch file or from the command line. Interactive applet or client application debugging from within an IDE is not available using this technique.

For more details, see “Method 3: Command Line Launch” on page 64.

• **Method 4: Mixed IDE and Command Line Launch**

The GSE is first launched manually from outside the IDE using a batch file. The client application is then run in debug mode from within the IDE. Interactive applet debugging from within an IDE is not available using this technique, but you can interactively debug the client application.

For more details, see “Method 4: Mixed IDE and Command Line Launch” on page 66.

“Table 3 - GSE Launch Methods” summarizes these techniques:

<table>
<thead>
<tr>
<th>GSE Launch Methods</th>
<th>Method Available with Templates?</th>
<th>Interactive Client Application Debugging?</th>
<th>Interactive Applet Debugging?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method 1: Client application code launch</td>
<td>Yes, if client application is developed with the applet code</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Method 2: IDE launch</td>
<td>Yes, if applet is developed alone</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Method 3: Command Line launch</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Method 4: Mixed launch</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 3 - GSE Launch Methods
Method 1: Client Application Code Launch

The following code can be included in the client application to launch the GSE directly:

```java
/* ------ imports ------ */
import com.gemplus.javacard.gse.Simulator;
import com.gemplus.javacard.gse.util.APDUPrinter;

/* simulator creation and start */
Simulator gse = new Simulator();
gse.start(5000, "GXP211_PK");
gse.printVersion();
```

Where the options used in the `gse.start()` call are:

- The port number ("5000") defining the listening socket port. This must exactly match the port number specified in the `opencard.properties` file.
- "GXP211_PK" runs the simulator in GemXpresso 211 PK mode. Each card type supported by GemXpresso RAD III has its own identifier. The Java Archive (JAR) library corresponding to this identifier must be present in the `installdir\lib` directory. See Table 4, "Available GSE Simulation Modes", on page 67 for a list of available identifiers and the corresponding Java archive files, then check that the version of GemXpresso RAD III you have purchased includes the necessary library file.

The `gse.printVersion();` line, to print version information in the IDE’s Message window, is optional.

**Note:** It is not possible to launch the GSE in this way from applet code.

Method 2: IDE Launch

The GSE can be launched from within the IDE by specifying the name of the class containing the GSE’s `main()` method, together with GSE start parameters, within the IDE’s project environment.

**Note:** It is not recommended to launch the GSE GUI from within the IDE, as this can cause serious screen refresh and other problems related to the graphical user interface.

Once an applet has been converted to a loadable format, uploaded, and installed in the GSE, the applet can be interactively debugged using the debugging facilities of the IDE, for example:

- Stepping through the applet code to view the applet’s responses.
- Setting breakpoints in the source code that are triggered when the corresponding line of code in the loaded applet is reached.
- Viewing and evaluating expressions.
For VisualCafé

Proceed as follows:

1. Open the project in VisualCafé.
2. Select Options from the Project menu and display the Project tab to display the project configuration options.
3. Specify:
   - **Main class**: com.gemplus.javacard.gse.Simulator
   - **Program arguments**: Enter the GSE options described previously in “Method 4: Mixed IDE and Command Line Launch”. For example: `-port 5000 -card gxp211_pk -version`
   
   If you created the project using one of the supplied templates, these values are already configured.
4. Add one of the following library files:
   - `installdir\lib\gse\gse_cardname.jar`
   - `installdir\lib\gse\gse_javacard_21.jar`
   
   Then add the following file:
   - `cryptix-jce-api.jar`
   
   Finally, add one of the following, depending on the version of GemXpresso RAD III you have purchased:
   - `cryptix-gemxpresso.jar` (supports full and limited cryptography)
   - `cryptix-gemxpresso-is.jar` (supports limited cryptography only).
5. Click OK.

For JBuilder

Proceed as follows:

1. Open the project in JBuilder.
2. Select Project > Project Properties to display the project configuration options.
3. Add one of the following libraries (Project > Project Properties > Required Libraries > Add) to the project:
   - `installdir\lib\gse\gse_cardname.jar`
   - `installdir\lib\gse\gse_javacard_21.jar`
   
   Then add the following file:
   - `cryptix-jce-api.jar`
Finally, add one of the following, depending on the version of GemXpresso RAD III you have purchased:

cryptix-gemxpresso.jar (supports full and limited cryptography)
cryptix-gemxpresso-is.jar (supports limited cryptography only).

4. Click the Run tab, then click Set and enter the following Class name:
   com.gemplus.javacard.gse.Simulator

5. In the Application parameters field, enter the GSE options described previously in “Method 4: Mixed IDE and Command Line Launch”. For example:
   -port 5000 -card GXP211_PK -version
   If you created the project using one of the supplied templates, these values are already configured.
   See Table 4, “Available GSE Simulation Modes”, on page 67 for a list of the simulation modes available with the -card parameter.

6. Click OK.

**Method 3: Command Line Launch**

There are a number of ways of launching the GSE from the command line:

- Launch GSE in a particular simulation mode by running a supplied batch file.
- Launch the GSE GUI component using a supplied batch file. When GSE GUI starts, you can choose the simulation mode to use.
- Launch the GSE GUI by first including the JAR file library for the simulation mode to be used in the CLASSPATH, then running the com.gemplus.javacard.gse.gui.GSE class from the command line.
- Launch the GSE GUI by running the com.gemplus.javacard.gse.gui.GSEGuiLauncher class from the command line. You can then choose the simulation mode to use from a dynamically constructed list of available targets.

**Note:** If you are using simulator files (SAP) generated using GemXpresso RAD version 1, the CLASSPATH must include all applets and classes that the client application needs to load into the GSE before you launch the GSE.

**Launching GSE from a Batch File**

The GSE can be launched from the following batch file:

```bash
RunGSE_cardname.bat
```
For example: RunGSE_GXP211_PK.bat
The command line options are:

-help
  Displays command help.

-port number
  Defines the listening socket port. The default value is “5000”. If you specify a
different value, update the opencard.properties file accordingly.

-card simmode
  The simulation mode to use. See “Table 4 - Available GSE Simulation Modes” for a
  list of the available simulation modes.

-atr ATRvalue
  Initialize the card with a custom value of ATR. The specified ATRvalue must be ISO-
  compliant or the OCF software layer may crash.

-motherkey customkey
  Use a custom mother key value for the card. customkey must be a valid 16-byte
  authentication key.

-serial customserialno
  Use a custom serial number for the card. customserialno must be a valid 16-byte serial
  number value.

-version
  Displays the GSE version and date.

Launching GSE GUI from a Batch File
The GSE GUI can be launched from the following batch file:
  RunGSE_GUI.bat
This batch file is in the installdir\bin directory.

Launching GSE GUI in a Specific Simulation Mode
To launch the GSE GUI from the command line in a specific simulation mode, you must
first include the JAR files for the GSE GUI and for the simulation mode to use in the
CLASSPATH. For example:

  set CLASSPATH=%CLASSPATH%;installdir\lib\gse\gse-gui.jar
  set CLASSPATH=%CLASSPATH%;installdir\lib\gse\gse_cardname.jar
Where \textit{cardname} is the type of card to simulate. See Table 4, “Available GSE Simulation Modes”, on page 67 for a complete list of available simulation modes.

Then run the following command line:

\begin{verbatim}
java com.gemplus.javacard.gse.gui.GSEApp options
\end{verbatim}

The command line \texttt{options} are described in “Method 2: IDE Launch” on page 62.

\textbf{Example}

\begin{verbatim}
set CLASSPATH=%CLASSPATH%;\ installer\lib\gse\gse-gui.jar
set CLASSPATH=%CLASSPATH%;\ installer\lib\gse\gse_gxp211_pk.jar
java com.gemplus.javacard.gse.gui.GSEApp -port 5000
-card GXP211_PK -version
\end{verbatim}

This command launches the simulator using port 5000, simulating a GemXpresso 211 PK card, and prints version information.

\textbf{Launching GSE GUI in Dynamic Simulation Mode}

To launch the GSE GUI from the command line and choose the simulation mode from a dynamically constructed list of available targets when GSE GUI starts, run the following command line:

\begin{verbatim}
java com.gemplus.javacard.gse.gui.GSEGuiLauncher
\end{verbatim}

The command line \texttt{options} are described in “Method 2: IDE Launch” on page 62.

GSE GUI inspects the libraries available in the \texttt{installer\lib} directory and dynamically builds a list of available simulation modes.

\textbf{Example}

\begin{verbatim}
java com.gemplus.javacard.gse.gui.GSEGuiLauncher
\end{verbatim}

\textbf{Method 4: Mixed IDE and Command Line Launch}

This method is similar to Method 1, except that you launch the GSE manually from the command line, then specify the main method of the GSE from within the IDE. The IDE’s main method is the client application, which communicates with the GSE without launching it.
Simulation Modes

The following table lists the simulation mode identifiers and the corresponding Java Archive (JAR) file names to use when launching the GSE.

<table>
<thead>
<tr>
<th>Identifiers (not case sensitive)</th>
<th>Corresponding JAR File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JAVACARD_21</td>
<td>gse_javacard_21.jar</td>
<td>Java Card 2.1 “virtual” simulation mode—no real card is being simulated</td>
</tr>
<tr>
<td>JAVACARD_21_IS</td>
<td></td>
<td>Java Card 2.1 “virtual” simulation mode, with limited key length</td>
</tr>
<tr>
<td>GXP211V2</td>
<td>gse_gxp211v2.jar</td>
<td>GemXpresso 211 V2 cards</td>
</tr>
<tr>
<td>GXP211V2_IS</td>
<td></td>
<td>GemXpresso 211 V2 cards with limited key length</td>
</tr>
<tr>
<td>GXP211_PK</td>
<td>gse_gxp211_pk.jar</td>
<td>GemXpresso 211 Compact PK cards</td>
</tr>
<tr>
<td>GXP211_PK_IS</td>
<td></td>
<td>GemXpresso 211 Compact PK cards with limited key length</td>
</tr>
<tr>
<td>GXPLITE-GENERIC</td>
<td>gse_gxplite-generic.jar</td>
<td>All GemXpresso Lite cards</td>
</tr>
<tr>
<td>GXPPro-R3</td>
<td>gse_gxppro-r3.jar</td>
<td>GemXpresso Pro R3</td>
</tr>
</tbody>
</table>

Table 4 - Available GSE Simulation Modes

**Note:** When in “virtual” Java Card 2.1 simulation mode, certain GSE-specific JavaCard 2.1 commands (for example, Upload File and Install) can be issued using the JCardManager. See “Chapter 4 - Using the JCardManager” for details.

Loading Applets into the GSE

Java Card applets are loaded into the GSE in exactly the same way as onto real cards. The only difference is the format of the executable load file (package): CAP files for real cards, and SAP files for the GSE.

When using the Java Card 2.1 “virtual” simulation mode, the JCardManager provides a basic set of commands for loading and installing applets. See “Sending Simulator Commands” on page 47.

To load applets into the GSE following the open platform (OP) standard, GemXpresso RAD III provides the JCardManager tool. The JCardManager is compatible with the OCF framework and supports the GSE as if it were a real Java Card-OP compatible card. Proceed as follows:
1. Launch the JCardManager and the GSE (or GSE GUI), in any order.
2. In JCardManager, select Simulator as the target and select the card type.
3. In JCardManager, issue the following commands in turn:
   - **Authenticate**
   - **Upload File into a Card**
     If the Upload command is successful, the package appears in the Package View of the GSE GUI window.
   - **Install**
     If the Install command is successful, the applet appears in the Applet View of the GSE GUI window.
   - **Select**
     If the Select command is successful, APDU commands can be sent to the applet and responses are displayed in the Messages area of the window.

See “Chapter 4 - Using the JCardManager” for details of how to load an applet into the GSE.

See the *GemXpresso RAD III Version 3.2 Getting Started* for a complete example of installing an applet in the GSE.

Applets can also be loaded from within a user-supplied client application by using the GemXpresso OP card service.

**Using the GSE GUI**

The GSE GUI provides a user-friendly tool with which to view the simulated contents of cards. The GSE GUI allows you to visualize the contents of a card and to trace the commands and responses exchanged between a card and a client application, such as the JCardManager.
Launching the GSE GUI

1. Choose Start > Programs > Gemplus Applications > GemXpresso RAD III > GemXpresso Simulator to open the Gemplus Simulation Environment Launcher window as shown in the following figure:

![Image of the Gemplus Simulation Environment Launcher Window]

Figure 26 - The Gemplus Simulation Environment Launcher Window

2. Modify any of the following default values:

   **Port**
   The port number defining the listening socket port. The default value is “5000”. This must exactly match the value used as the address port in the OCF opencard.properties file.

   **Card type**
   The type of card to simulate. Select a card type from the list. Only cards types compatible with the version of GemXpresso RAD III you are using are displayed. In addition, GemXpresso 211 V1 cards are not supported by the GSE, so are never listed.

   **Options**
   Select Options and enter any startup options. See “Method 3: Command Line Launch” on page 64 for a complete list of the available options.

   **Load a context at startup**
   Optionally, select the check box, choose either Full or System, and enter the name of a context file to load at startup. Click the browse button to search for an existing file. See “Working with Contexts” on page 73 for information on contexts.

3. Click Start to continue.

   **Note:** To permanently change the default values that are displayed, see “Setting Preferences” on page 72.
The GSE GUI Window

The Gemplus Simulation Environment window appears:

![The Gemplus Simulation Environment Window](image)

Figure 27 - The Gemplus Simulation Environment Window

The status information area of the window displays a tab for each environment supported by the chosen card—**Java Card** only for Java Card, or **Java Card** and **Open Platform** tabs for a Java Card-OP card—and a property sheet for each applet installed in the card. The property sheets display the following status information:

- **Open Platform**:
  - CM AID. The AID of the Card Manager.
  - CM State. The current life cycle status of the Card Manager.

Refer to “Differences Between the GSE and Real Cards” on page 75 for details.

- **Java Card**: the AIDs of the default and currently selected applets (initially, the Card Manager).

- **Applet** property sheet: private data values extracted from the applet code.
Using the GSE

The **Package View** displays a list of all packages installed on the card. When first started, the default system packages for the chosen card type are displayed. Double-click an entry in the package view to display either the package AID, or a list of applets contained in the package. Double-click an applet icon to display the applet AID.

Package and applet names and AIDs are color-coded. See “Setting Preferences” on page 72 for details. Packages or applets that cannot be deleted, such as the system packages and the Card Manager applet, are displayed with the [X] icon.

The **Applet View** displays a list of all applets installed on the card. The Card Manager applet is always displayed. Double-click an applet icon to display the applet AID.

To the right of the **Package View** and **Applet View**, status windows show standard OP-specific information relating to the selected package or applet.

The **APDU trace window** records in real time commands and responses exchanged between the card simulator and the client application.

The **Log and Error Trace** window records in real time and log and error messages that occur.

Finally, the **Status Line** at the bottom of the card simulator window displays the following information:

| 5000 | gsp211_ski_js | Simulator state: Ok |

**Figure 28 - GSE GUI Status Line**

**Starting, Stopping andResetting the GSE**

You can start, stop and reset the GSE software layer without leaving the GSE GUI. This allows you, for example, to quickly change the card type being simulated without needing to exit and restart the GSE GUI.

**To reset the GSE:**

Click the **Reset** button on the GSE GUI toolbar, choose **Simulator > Reset**, or press **F6**.
To stop the GSE:

Click the Stop button on the GSE GUI toolbar, choose Simulator > Stop, or press F7.

To start the GSE:

Click the Start button on the GSE GUI toolbar, choose Simulator > Start, or press F9. The Start Gemplus Simulation Environment window is displayed (see “Figure 26 - The Gemplus Simulation Environment Launcher Window” on page 69), where you can change the port number used by GSE GUI, the type of card to simulate, or the startup options to use.

Setting Preferences

To set preferences and default settings for GSE GUI, choose Edit > Preferences. The Default Preferences window is displayed:

![Figure 29 - The Default Preferences Window](image)

You can change any of the following preferences:

**General Property Sheet.** Enter the default System Context file name, or click the browse button to search for the file.

Enter the default Full Context file name, or click the browse button to search for the file.

If you select Always confirm overwriting, the GSE GUI will always prompt before overwriting an existing context file.

See “Working with Contexts” on page 73 for information on contexts.
**Simulator Property Sheet.** Specify the default value of **Port**, **Card Type** and **Options** that appear on the **Start Gemplus Simulation Environment** window when you start the card simulator (see “Figure 26 - The Gemplus Simulation Environment Launcher Window” on page 69).

**Short Cut Configuration Property Sheet.** Configure the keyboard shortcuts available to the card simulator’s menu items. You can alter the default shortcuts by assigning any function key, or combination of Alt+Function key, Ctrl+Function key, or Shift+Function key, to a menu item.

**Color Configuration Property Sheet.** This property sheet allows you to configure default colors used to display certain types of information on the GSE GUI window. Select an item from the list and click **Other** to select a different color to use. Click **OK** to return to the Gemplus Simulation Environment window.

**Note:** All preferences are recorded in the file \( \text{installdir}/\text{conf}/\text{gse-gui.properties} \).

---

**Working with Contexts**

To facilitate simulating personalized card contents, GemXpresso RAD III uses the concept of a **context**. A context is an image of the current contents of a card, including the AID and life cycle status of each item. Loading a context into the simulator is therefore equivalent to inserting a personalized smart card into a card reader.

GemXpresso RAD III supports the following context types:

- A **system context** includes full details of all system packages and applets on the card.
- A **full context** includes additionally all custom packages and applets on the card, for example, packages that you have uploaded and installed.

Contexts allow you to:

- Switch quickly between contexts to test your client application against different card types without having to stop and start the card simulator each time.
- Save a card’s contents at a particular point in the development cycle and restore the contents at a later time without having to re-load and install packages and applets.

Each context is stored in a file. Full and system contexts are stored in full serialized data (*.FSD) or system serialized data (*.SSD) files, respectively. Context files are stored in a Gemplus-proprietary binary format.
Saving a Full Context
To save the full context of the current card:
1. Choose **File > Save Full Context**.
2. Type the full path and file name of the full serialized data (FSD) file to be created, or
   click the browse button and select the directory and file name.
3. Click **OK** to save the full context to the specified file.

You are asked whether you want to save the current full context whenever you quit the
card simulator.

Saving a System Context
To save the system context of the current card:
1. Choose **File > Save System Context**.
2. Type the full path and file name of the system serialized data (SSD) file to be created,
   or click the browse button and select the directory and file name.

Click **OK** to save the system context to the specified file.

Loading a Full Context
To load a previously saved full serialized data file:
1. Choose **File > Load Full Context**.
2. Type the full path and file name of the full serialized data (FSD) file to be loaded, or
   click the browse button and select the directory and file name.

Click **OK** to load the full context file. Any error or warning messages encountered while
importing the file are displayed in the Log and Error Trace window.

Loading a System Context
To load a previously saved System Serialized Data file:
1. Choose **File > Load System Context**.
2. Type the full path and file name of the system serialized data (SSD) file to be loaded,
   or click the browse button and select the directory and file name.

Click **OK** to load the system context file. Any error or warning messages encountered while
importing the file are displayed in the Log and Error Trace window.
GSE Restrictions

The GSE does not completely simulate the reaction of the card during applet execution. The following constraints apply:

- **Memory behavior**
  The GSE does not simulate the card memory use, because it runs under the workstation-based Java VM, which is not capable of simulating the card-based VM’s memory use. Therefore, the size of applets in the simulator is unlimited.

- **Java operator restriction**
  The unsigned right shift >>> cannot be used in the Java source code.

- **Transaction behavior**
  The transaction behavior is not implemented by default, but is supported in the simulation API.

- **Firewalls**
  The applet firewall does not exist, but the applet context is detected and checked when necessary.

- **Persistence**
  When the simulator is terminated, all the applet’s data is lost. Persistence can be simulated by saving and restoring card contexts.

- **Reset**
  The power on/off is not equivalent to a card reset.

Differences Between the GSE and Real Cards

The GSE uses default values for a number of parameters, which may differ from those used by a real card:

- The card production life cycle (CPLC) data structure.
- The card’s serial number. The default value can be changed with the -serial startup option.
- Key set values. Although the default values match those of the corresponding card types, mass-produced cards may use different values. You can specify a different default value using the -motherkey startup option.
- Answer to reset (ATR) values. The default value can be changed with the -atr startup option.
- Memory management (see “GSE Restrictions”).

These values may be used or returned by certain APDU commands.
The values used are displayed on the **Open Platform** tab of the Status Information area of the GSE GUI window when it starts. The following example is for a GemXpresso 211 card:

![Figure 30 - Default GSE Values](image)

### Default Card Manager AID Values

When simulating a card, the same value as that of the real card is used. When used in JavaCard 2.1 “virtual” simulation mode, the GSE uses the following default AID values:

<table>
<thead>
<tr>
<th>AID Name</th>
<th>ASCII Value</th>
<th>Hex Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Package</td>
<td>“SYSTEM_P”</td>
<td>53 59 53 54 45 40 5F 50</td>
</tr>
<tr>
<td>System Applet</td>
<td>“SYSTEM_A”</td>
<td>53 59 53 54 45 40 5F 41</td>
</tr>
</tbody>
</table>

*Table 5 - Default Card Manager AID Values for Virtual Mode*

### Default Card Manager Status

The GSE uses the value 07 (INITIALIZED) for the Card Manager’s initial state.

### CPLC Data Structure

The CPLC data structure for the GSE is as follows:

<table>
<thead>
<tr>
<th>Data Element</th>
<th>Hex value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IC fabricator</td>
<td>00 04</td>
</tr>
<tr>
<td>IC type</td>
<td>00 15</td>
</tr>
<tr>
<td>Operating System identifier</td>
<td>00 04</td>
</tr>
<tr>
<td>Operating System release date</td>
<td>12 99</td>
</tr>
<tr>
<td>Operating System release level</td>
<td>32 30</td>
</tr>
</tbody>
</table>

*Table 6 - Default CPLC Values for the GSE*

You can obtain the current CPLC data structure using the **Get Data** APDU.
Default Serial Number

The following card serial number value is appended to the CPLC data structure.
01h 02h 03h 04h 05h 06h 07h 08h
09h 0Ah 0Bh 0Ch 0Dh 0Eh 0Fh 10h // CHIP_SERIAL_NUMBER

You can obtain the current serial number using the Get Data APDU, and set a different value using the -serial startup option.

Key Set Values

The GSE performs key diversification. At startup, it contains the following static keys:

Key Set Version: 13 (decimal)

Mother Key Value: The default mother key value is the same as that of the card delivered with the version of GemXpresso RAD III you have purchased. You can change the default value used with the -motherkey startup option.

Answer To Reset Value

The answer to reset (ATR) is the initial information emitted by a smart card upon being reset.

The ATR is a string of bytes, interpreted by the terminal, which consists of two parts:

- The protocol bytes, which come first, provide information about the communication protocols supported by the card (for example: protocol type, baud). These values cannot be altered after card issuance.
- The historical bytes, which follow, provide information about the type of card. These are normally used to identify the type of card.

The GSE returns its own answer to reset (ATR), which differs from that returned by a real GemXpresso 211 card:

- The ATR used by the GSE for GemXpresso V2 cards is:
  3Bh 0Dh 80h 31h 80h B0h 03h 01h 01h F0h 83h 00h 90h 00h
  The last nine bytes are the historical bytes.
- The ATR used by the GSE for GemXpresso PK cards is:
  3Bh 0Dh 80h 31h 80h 65h B0h 05h 01h 02h F0h 83h 00h 90h 00h

You can change the default ATR value using the -atr startup option.

Refer to the OpenCard Framework Programmer’s Guide for details on how to program a client application to read and interpret the ATR bytes.
Status and Error Codes Returned by GSE

In Java Card only mode, the status codes returned by the GSE are ISO codes defined in the javacard.framework.ISO7816 class of the OpenCard Framework API.

In Java Card-OP mode, the status codes returned by the GSE are the OP error status. The OP specification sometimes uses ISO codes with a different interpretation. See the OP command descriptions in the appropriate GemXpresso card reference manual for more details.

In the GSE, some ISO status codes can be used in certain cases, for example:

- Load with an invalid SAP file.
- Unpredictable events such as a null reference due to an incorrect install command or unavailable components.

The 6F00 (unknown error) ISO status code can be the result of a non-ISO exception in the GSE (for example, null pointer exceptions or ArrayOutOf Bounds). In case of non-ISO exceptions, a description appears on the GSE console or in the Messages area of the GSE GUI window describing the problem.
The GxpLoader

The GxpLoader tool allows you to load, and install files into real cards or the GSE in one step.

Running the GxpLoader

The GxpLoader is a command-line only utility. You can run the tool in two ways:

1. By typing the command and all parameters on the command line.
2. By using a deployment file that contains all the required commands and command parameters. In this case, pass the name of this deployment file as a parameter to the GxpLoader. See “Chapter 7 - The Deployment Editor” for details on creating deployment files.

Running GxpLoader from the Command Line

The name of the GxpLoader package is:

```
com.gemplus.tools.gxploader
```

This package is situated within the gxploader.jar library file. Before running the GxpLoader utility from the command line, you may need to update the CLASSPATH accordingly. For example:

```
set CLASSPATH=%CLASSPATH%;c:\gemplus\gemxpressorad3\lib\loader\gxploader.jar
```

Then run the tool as follows:

```
java com.gemplus.tools.gxploader.GxpLoader parameters
```

Where the parameters are described in the next section.
Using GxpLoader

Configuring Parameters for the GSE

The command line for launching the GxpLoader using a deployment file created using the Deployment Editor and with the GSE as target is:

```
java -Dgemplus.gemxpresso.rad.home=radhomedirectory
    com.gemplus.tools.gxploader.GxpLoader
    -gse targetType
    -deploymentfile deploymentfilename
    -terminal terminalreadename
    [-scriptname scriptname]
    [-ocf opencardpropsfilename]
```

The parameters are:

- `-Dgemplus.gemxpresso.rad.home=radhomedirectory`
  Mandatory. Specifies the root directory of your GemXpresso RAD III installation, for example `c:\Gemplus\GemXpresso.rad3`.

- `-gse`
  Mandatory. `targetType` specifies the simulated card type for which the GxpLoader command is run.

- `-deploymentfile`
  Mandatory. Specifies the name of the deployment file to use.

- `-terminal`
  Mandatory. Name of the terminal.

- `-scriptname`
  If no script name is specified, all scripts found in the current directory targeting the specified card (`-card`) and the GSE are executed.

- `-ocf opencardpropsfilename`
  Specify the full path name of the OCF opencard.properties file.

Example

```
java -Dgemplus.gemxpresso.rad.home=c:\gemplus\gemxpresso.rad3
    com.gemplus.tools.gxploader.GxpLoader
    -gse Gxp211v2
    -deploymentfile installdir\examples\cardname\OPPurse\OPPurseToGSE.gxd
    -terminal Simulator
    -ocf c:\temp\opencard.properties
```
Configuring Parameters for a Card

```
java -Dgemplus.gemxpresso.rad.home=radhomedirectory
com.gemplus.tools.gxploader.GxpLoader
-card targetType
-deploymentfile deploymentfilename
-terminal terminalreadername
[-scriptname scriptname]
[-ocf opencardpropsfilename]
[-cardprofile cardprofilefilename]
```

The parameters are:

- `-Dgemplus.gemxpresso.rad.home=radhomedirectory`
  Mandatory. Specifies the root directory of your GemXpresso RAD III installation, for example C:\Gemplus\GemXpresso.rad3.

- `-card`
  Mandatory. `targetType` specifies the card type for which the GxpLoader command is run.

- `-deploymentfile`
  Mandatory. Specifies the name of the deployment file to use.

- `-terminal`
  Mandatory. Name of the terminal.

- `-scriptname`
  Name of the script to execute. If no script name is specified, all scripts found in the current directory targeting the specified card (-card) and terminal (-terminal) are executed.

- `-ocf opencardpropsfilename`
  Specify the full path name of the OCF opencard.properties file to use.

- `-cardprofile cardprofilefilename`
  Specify the full path and file name of the card profile to use. By default, this is installdir\resources\cardprofile.
Example

java -Dgemplus.gemxpresso.rad.home=c:\gemplus\gemxpresso.rad3
com.gemplus.tools.gxploader.GxpLoader
-card Gxp211v2
-deploymentfile installdir\examples\cardname\OPPurse\OPPurseToCard.gxd
-terminal gcr410_com1
-ocf c:\temp\opencard.properties
-cardprofile installdir\resources\cardprofile\GXP211v2.gsc
The Deployment Editor

Introduction

The Deployment Editor is a graphical utility that you use to create deployment files. A deployment file is a structured XML file containing details of the commands and command parameters necessary to load and install an applet in a card.

You create a deployment file by using the graphical user interface of the Deployment Editor to specify parameters for the following default commands used during the applet load and install process:

- Authenticate
- Upload File into a Card
- Install
- Select

You then save the settings as a deployment file. This deployment file can subsequently be used with:

- The JCardManager. See “Chapter 4 - Using the JCardManager”.
- The GxpLoader. See “Chapter 6 - The GxpLoader”.

Running the Deployment Editor

You can run the Deployment Editor:

1. From the GemXpresso RAD menu of one of the IDE plug-ins. See “Chapter 1 - Developing Project Code”.
2. From within the JCardManager. Refer to “Chapter 4 - Using the JCardManager” for details.
Using the Deployment Editor

You can create a new deployment file or open an existing one.

Creating a New Deployment File

To create a new deployment file:

1. In the JCardManager, choose Deployment > New. This opens the following window:

![New Deployment window](image)

   **Figure 31 - Naming the New Deployment File**

2. In **Deployment file name**, enter the name of the new deployment file, with a .GXD extension. You can use the browse button to select the directory for the file.

3. In **Card profile directory**, browse to the directory containing the card profiles if it is not the one that is already displayed.

4. Click **Next** to continue. This displays the window shown in “Figure 32 - Selecting Card Profiles” on page 85.

5. From this window you specify the card profiles that the deployment file will support. You can choose individual card profiles or all the card profiles specified in a .gxp file.
To support all the files in a .gxp file:
Click **Select a gxp file** and browse to the file.

To choose individual card profiles:
Click **Select card profile**, then highlight a profile in **Available cards** that you want to select and click **Add**. The card profile appears in **Selected cards**. Repeat for each profile that you want to select. You can click **Add All** to select all the profiles in the list. If you want to deselect a profile from the **Selected cards list**, highlight it and click **Remove**. **Remove All** deselects all the currently selected card profiles.

---

**Note:** The profiles that appear in **Available cards** are those found in the directory that you chose for **Card profile directory** in the preceding window.

6. Click **Finish**. This action displays the **Deployment Editor** window, shown in “**Figure 34 - The Deployment Editor Window**” on page 87.
Opening an Existing Deployment File

To open an existing deployment file:

1. In the JCardManager, choose Deployment > Open. This opens the following window:

   ![Figure 33 - Selecting a Deployment File to Open](image)

2. In Deployment file name, enter the name of the existing deployment file, with a .GXD extension. You can use the browse button to select the directory for the file.

3. Click Open. Nothing appears; you must perform an Edit Deployment action to display the file that you have opened.

4. In the JCardManager, choose Deployment > Edit deployment. This opens the Deployment Editor window.
The Deployment Editor Window

This window displays:

- A property sheet for each of the Supported card types that you chose to support.
- A list of the Commands to support for this card type.
- The Command Parameters area of the window, where you select the parameters to use for each command.
- The name of the Current script.

Click a Supported card types property sheet.

To add or remove commands from the list of Commands:

1. Click the Plus button. This action displays the Add command window, as shown in “Figure 35 - Configure the Command Set for Your Deployment File” on page 88.
2. The right pane displays the full set of commands, the left pane displays the supported commands. To add a command, select it in the left pane and click **Plus**. To remove a command, select it in the right pane and click **Minus**.

3. Click **OK** to exit the **Add command** window.

**To define the parameters for a supported command:**

1. In the **Deployment Editor** window, click a command in the **Commands to support** list to display the command’s parameters in the **Command Parameters** area of the window. Configure the parameter values as required. The layout of the Command Parameters area of the window is identical to that of the Command Parameters area of the JCardManager window. Refer to the *GemXpresso RAD III Version 3.2 Command Reference* for details on configuring the parameters for each command.

2. Configure all the commands displayed in the **Commands to Support** area.

3. Repeat this procedure for all supported card types.

When you have finished configuring all command parameters for all supported card types, click **OK** to generate/save the deployment file.
The CapDump Utility

The CapDump utility is an aid to diagnosing and testing the contents of an applet’s CAP files. CapDump:

- Parses CAP files and displays any errors or warnings.
- Displays the contents of CAP files in a readable format.
- Allows you to save the readable form of CAP files as ASCII text files for further analysis.
- Provides a Search facility to allow you to search CAP files for specific text strings.

Launching the CapDump Utility

You can start the CapDump utility in a number of ways:

- From within the JCardManager:
  Choose Tools > CapDump
- From the command prompt:
  a) Open a command prompt window and browse to the installdir\bin directory.
  b) Run the CapDump.bat batch file:
    
    Capdump
Using the CapDump Utility

When the CapDump utility starts, the following window is displayed:

![The CapDump Window](image)

To display the contents of a CAP file:

1. Enter the name of the CAP file to open, or click the browse button and search for the file. You can select either a CAP file or a JAR file that contains the CAP file.
2. Select Dump the file.
3. Click Dump. The selected CAP file is opened and its contents displayed on the screen.

Any error messages encountered by the utility while opening the file are displayed in the Errors area at the bottom of the window.
To save the readable form of a CAP file:
1. Open the CAP file as described previously.
2. Select Save to File.
3. By default, the Output field displays the name of the selected CAP file with a “.TXT” extension. To save the displayed output to a different file, modify the Output file name. You can also click the browse button and select a different directory in which to store the file.
4. Click Dump. The readable form of the CAP file is written to the specified file.

Any error messages encountered by the utility while reading or writing the file are displayed in the Errors area at the bottom of the window.

To copy a part or all of the displayed contents to the clipboard:
1. Open the CAP file as described previously.
2. Select the lines to copy.
3. Click the Copy icon.

To search for a string in the displayed contents:
1. Open the CAP file as described previously.
2. Click the Search icon.
3. Enter the string to search for and click Find. Optionally, select Match Case to perform a case-sensitive search.
Diagnosing the CAP File Contents

“Figure 37 - CAP File Contents” highlights some of the information in a CAP file that may be useful for diagnosing CAP files and applets. Refer to Sun’s Java Card 2.1 Virtual Machine Specification for full details.

```
header = {
  magic             : decaffed
  minor_version     : 1
  major_version     : 2
  flags             : 4
  pkg_minor_version : 0
  pkg_major_version : 1
  pkg_AID_length    : 5
  pkg_AID           : 00.00.00.00.00
}

ConstantPool = {
/* 0000,   0 */CONSTANT_InstanceFieldRef : field 1 of class 0x0000
/* 0004,   1 */CONSTANT_InstanceFieldRef : field 0 of class 0x0000
/* 0008,   2 */CONSTANT_InstanceFieldRef : field 3 of class 0x0000
/* 000c,   3 */CONSTANT_InstanceFieldRef : field 4 of class 0x0000
/* 0010,   4 */CONSTANT_InstanceFieldRef : field 5 of class 0x0000
/* 0014,   5 */CONSTANT_InstanceFieldRef : field 2 of class 0x0000
/* 0018,   6 */CONSTANT_StaticMethodRef  : external: 0x80,0x7,0x1
/*0033*/ L0:   aload_0
/*0034*/       invokespecial   7
/*0037*/     /aload_0
/*0038*/     sconstant_0
/*0039*/     putfield_b      0
/*003b*/     /aload_0
/*003c*/     aconst_null
```

Projected size of the CAP file’s image on the card (not including memory allocated for instantiating objects).

The projected amount of card memory required to load the file.

Package AID, as specified on the Project Editor window.

Major and minor package version.

Values of constants.

Byte code (as interpreted by the card’s Virtual Machine).

Figure 37 - CAP File Contents
The GCF to GXCF Converter

The GCF to GXCF utility converts command scripts recorded using previous versions of GemXpresso RAD to the new GXCF format used by GemXpresso RAD III.

Running the GCF to GXCF Script Converter

The GCF to GXCF script converter is a command-line only utility. You can run the tool in several ways:

- Editing a supplied batch file and running the batch file.
- Typing a command from the command line.

Running the Script Converter with a Batch File

The batch file is:

```
installdir\bin\GcfScriptToGxcfScriptConversion.bat
```

Copy and rename this batch file, then configure the parameters as appropriate.

Running the Script Converter from the Command Line

The name of the script conversion package is:

```
com.gemplus.tools.gcf_gxcf_converter.ConvertGcfToGxcf
```

This package is situated within the gcf2gxcfconverter.jar library file.

Run the tool as follows:

```
java com.gemplus.tools.gcf_gxcf_converter.ConvertGcfToGxcf parameters
```

Where the parameters are described in the next section.
**Configuring the Script Converter**

The command line for launching the script converter is as follows:

```java
java com.gemplus.tools.gcf_gxcf_converter.ConvertGcfToGxcf
    -gcf scripttoconvert
    -cardservice cardservicename
```

The parameters are:

- **-gcf scripttoconvert**
  The name of the GCF script to be converted to GXCF format.

- **-cardservice cardservicename**
  The name of the card service that will be used to replay the commands contained in the script file. The available OCF card services are listed in the `OpenCard.services` section of the `opencard.properties` file, for example:

  ```
  “com.gemplus.opencard.service.op.vop.vop200.CardServiceVOP200Factory”.
  ```

The utility displays warning messages if it encounters potentially ambiguous commands or other conversion problems during conversion.

If successful, a converted script in GXCF format (a well-formed XML file) is placed in the directory from where the script was run.

**Example**

```java
java com.gemplus.tools.gcf_gxcf_converter(ConvertGcfToGxcf
    -gcf c:\gemplus\gemxpresso.rad3\scripts\Simple.gcf
    -cardservice
        com.gemplus.opencard.service.op.vop.vop211.CardServiceVOP200
```
Sending and Receiving APDUs

This chapter describes how commands are passed between a client application and a Java Card applet.

Introduction

Once an applet has been successfully installed in a Java Card, it can only communicate externally using application protocol data unit (APDU) commands. The ISO standard ISO 7816-4: Interindustry Commands for Interchange describes the possible formats of APDUs. The standard defines two types of APDU:

- **Command APDUs**, sent by a client application to a card.
- **Response APDUs**, returned by a card to a client application.

![Figure 38 - Command and Response APDUs](image-url)
“Figure 39 - Structure of Command and Response APDUs” shows the structure of each type of APDU:

**Command APDU**

<table>
<thead>
<tr>
<th>Field</th>
<th>Length in bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLA</td>
<td>1</td>
<td>Class byte: category of APDU command.</td>
</tr>
<tr>
<td>INS</td>
<td>1</td>
<td>Instruction code: Uniquely identifies the command APDU.</td>
</tr>
<tr>
<td>P1</td>
<td>1</td>
<td>Parameter 1. Used to pass additional parameter information.</td>
</tr>
<tr>
<td>P2</td>
<td>1</td>
<td>Parameter 2. Used to pass additional parameter information.</td>
</tr>
<tr>
<td>Lc</td>
<td>1</td>
<td>Length of the data field in bytes. (GemXpresso RAD III tools calculate this value automatically).</td>
</tr>
<tr>
<td>Data</td>
<td>1-n</td>
<td>Data to be sent to the card.</td>
</tr>
<tr>
<td>Le</td>
<td>1</td>
<td>The number of bytes expected in the response APDU.</td>
</tr>
</tbody>
</table>

**Table 7 - Command APDU Fields**

**Response APDU**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW1</td>
<td></td>
</tr>
<tr>
<td>SW2</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 39 - Structure of Command and Response APDUs**
Since both the Body and Data field of a command APDU are optional, four formats of command APDU are possible:

<table>
<thead>
<tr>
<th>Case 1</th>
<th>Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 2</td>
<td>Header Le</td>
</tr>
<tr>
<td>Case 3</td>
<td>Header Lc Data</td>
</tr>
<tr>
<td>Case 4</td>
<td>Header Lc Data Le</td>
</tr>
</tbody>
</table>

In **Case 1**, the command contains no data and expects no data to be returned in response.

In **Case 2**, the command contains no data, but expects data in response (the value of Le indicates the number of response bytes expected).

In **Case 3**, the command contains data, but expects no data in response (the value of Lc indicates how many bytes of data follow).

In **Case 4**, the command contains data (the value of Lc indicates how many bytes of data follow) and expects data in response (the value of Le indicating the number of response bytes expected).
Response APDU Formats

The data field of a response APDU is optional, since some commands do not require any data to be returned. Therefore, a response APDU has two possible formats:

<table>
<thead>
<tr>
<th>Case 1</th>
<th></th>
<th>SW1</th>
<th>SW2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Case 2</td>
<td>SW1</td>
<td>SW2</td>
<td></td>
</tr>
</tbody>
</table>

Figure 41 - Response APDU Formats

In Case 1, the number of data bytes is determined by the value of Le in the command APDU. To these data bytes, two status bytes, SW1 and SW2, are appended.

In Case 2, only the SW1 and SW2 status bytes are returned.

Sending and Receiving APDUs

The easiest way to send a command APDU to an installed applet is to use the JCardManager. The JCardManager’s Messages area displays the contents of any response APDUs received from the applet.

Sending Command APDUs

To issue a command APDU:

1. Start the JCardManager.
2. Either start the GSE GUI and load the context containing the installed applet, or insert the card containing the applet into a configured card reader.
3. In the JCardManager, select the target (GSE or card reader) and the card type.
4. From the list of commands, select the Send APDU command.
5. Enter the parameters of the command APDU, including field values and any data bytes.
6. Click Go to issue the command.
7. Study the Messages area of the JCardManager window to view the value of status words and data fields in the returned response APDU.

See “Chapter 4 - Using the JCardManager” for full details.
Sending and Receiving APDUs

Viewing Response APDUs

The JCardManager displays the results returned in response APDUs in the Messages area of the window.

If the command was successful, the following results are displayed:

Case 1 (data and status bytes returned):

\[ \begin{array}{c}
00 & 64 & 90 & 00 \\
\end{array} \]

Status bytes (SW1 = 90, SW2 = 00)

Two data bytes

Case 2 (status bytes only returned):

\[ \begin{array}{c}
90 & 00 \\
\end{array} \]

Status bytes (SW1 = 90, SW2 = 00)

If the command was not successful, the applet must return an error code other than 0x9000 in the status bytes SW1 and SW2. Although specific error codes can be issued by the applet itself—for example, that a command completed but a problem exists—the interface ISO7816 includes a range of ISO 7816-defined error codes that should be used wherever applicable because they help make the code ISO 7816-4 compatible.

Applets should throw an ISOException exception whenever an error occurs, passing an appropriate reason code. These reason codes are returned as status bytes in the response APDU in the same way as other error codes.

Any other type of error causes 0x6F00 to be returned.

Coding Command and Response APDUs

Coding Command APDUs

From an applet’s perspective, incoming command APDUs are handled by the \texttt{process()} method:

```java
public void process(APDU apdu) throws ISOException {
    ... 
}
```
The applet’s first task is to identify the command. This is done by inspecting the contents of the APDU buffer. The APDU buffer is a byte array containing the parameters and data of the command APDU.

The `getBuffer()` method determines the length of the APDU buffer:

```java
byte[] apduBuffer = apdu.getBuffer();
```

The ISO7816 interface defines a number of useful constants that allow the individual fields of a command APDU to be extracted from the buffer based on their offset from the start of the buffer. For example:

```java
switch(apduBuffer[ISO7816.OFFSET_CLA])
{
    case INS_VERIFY_PIN:
        //process the verifyPIN command APDU
        verifyPIN(apdu);
        break;
    case INS_GET_BALANCE:
        //process the getBalance command APDU
        getBalance(apdu);
        break;
    case INS_EXTERNAL_AUTHENTICATE:
        if(apduBuffer[ISO7816.OFFSET_CLA]==CLA_EXTERNAL_AUTHENTICATE)
            // call external authenticate security method
            external_authenticate(apdu);
        break;
}
```

**OFFSET_CLA** Offset to the CLA field

**OFFSET_INS** Offset to the INS field

**OFFSET_P1** Offset to the P1 field

**OFFSET_P2** Offset to the P2 field

In the following code example, a switch/case construct is used to test the value of the INS field (which identifies the command), or of both the INS and the CLA field (which identifies the category of OP 2.0.1 commands such as External Authenticate):

```java
private final static byte INS_GET_BALANCE = (byte)0x30;
private final static byte INS_VERIFY_PIN = (byte)0x33;
```

```java
switch(apduBuffer[ISO7816.OFFSET_INS])
{
    case INS_VERIFY_PIN:
        //process the verifyPIN command APDU
        verifyPIN(apdu);
        break;
    case INS_GET_BALANCE:
        //process the getBalance command APDU
        getBalance(apdu);
        break;
    case INS_EXTERNAL_AUTHENTICATE:
        if(apduBuffer[ISO7816.OFFSET_CLA]==CLA_EXTERNAL_AUTHENTICATE)
            // call external authenticate security method
            external_authenticate(apdu);
        break;
}
```

// Define 1-byte APDU constants for custom APDUs

```java
// Define 1-byte APDU constants for custom APDUs
private final static byte INS_GET_BALANCE = (byte)0x30;
private final static byte INS_VERIFY_PIN = (byte)0x33;
```
In each case, an APDU handler routine is called, the APDU buffer being passed as a parameter so that the data or other parameters of the command can be extracted.

If the APDU instruction code is not recognized, an exception is thrown:

```
default:
    // The INS code is not recognized
    ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
    break;
```

An APDU handler routine typically first verifies that the data field of the APDU buffer contains the expected number of bytes. This can be done by testing the fifth byte of the APDU buffer, the Lc field, and then calling the `setIncomingAndReceive()` method of the APDU class to extract the data, which returns the number of bytes read.

```java
private void credit(APDU apdu) throws ISOException {
    // get the APDU buffer
    byte[] apduBuffer = apdu.getBuffer();

    // get the number of bytes received
    if(apduBuffer[4] != 2 || apdu.setIncomingAndReceive() != 2)
        throw new ISOException(ISO7816.SW_WRONG_LENGTH);

    // The data can now be read into a variable, starting from the sixth byte of the APDU buffer, for example:
    short amount = (short)(((apduBuffer[6]) & (short)0x000000FF)
                        | ((apduBuffer[5] << 8) & (short)0x0000FF00));
```

**Coding Response APDUs**

A command APDU requires a response APDU to be returned to the client application if it contains a body in which the Le field has a non-zero value (cases 2 and 4 in “Figure 40 - Command APDU Formats” on page 97). The value of the Le field indicates the number of bytes expected in the response.

To minimize memory usage, response data is best stored in the same APDU buffer structure in which the incoming command APDU data was received. For example:

```java
private void getBalance( APDU apdu ) {
    // get the APDU buffer
    byte[] apduBuffer = apdu.getBuffer();
```
// write the balance into the APDU buffer
apduBuffer[5] = (byte)(balance >> 8) ;
apduBuffer[6] = (byte)balance ;

To send a response APDU, the setOutgoing() method is called to indicate to the communications layer that the applet wishes to send a response APDU:

    apdu.setOutgoing() ;

The setOutgoingLength() method is then called to fix the number of data bytes that are being returned (not including the SW1 and SW2 status bytes):

    // 2 bytes to return
    apdu.setOutgoingLength((short)2) ;

Finally, the sendBytes() method is called to send the response APDU, passing the offset into the APDU buffer and number of bytes to send as parameters. In the following example, the offset is set to 5 because the data was stored in the buffer starting at offset 5, and two data types are returned:

    // offset and length of bytes to return in the APDU buffer
    apdu.sendBytes((short)5, (short)2) ;
This appendix includes code listings for the OPPurse Java Card applet.

**The OPPurse Applet**

```java
package com.gemplus.examples.oppurse;

import javacard.framework *
import visa.openplatform *

public class OPPurse extends javacard.framework.Applet {{
    private final static byte CLA_OPPURSE = (byte)0x90;
```
private final static byte CLA_OFFURSE_SM = (byte)0x94;
private final static byte INS_GET_BALANCE = (byte)0x10;
private final static byte INS_DEBIT = (byte)0x12;
private final static byte INS_CREDIT = (byte)0x14;
private final static byte INS_VERIFY_PIN = (byte)0x16;
// the OP/VOP specific instruction set for mutual authentication
private final static byte CLA_INIT_UPDATE           = (byte)0x80;
private final static byte INS_INIT_UPDATE           = (byte)0x50;
private final static byte CLA_EXTERNAL_AUTHENTICATE = (byte)0x84;
private final static byte INS_EXTERNAL_AUTHENTICATE = (byte)0x82;

// the PIN validity flag
private boolean validPIN = false;

// SW bytes for PIN Failed condition
// the last nibble is replaced with the number of remaining tries
private final static short SW_PIN_FAILED = (short)0x63C0;

// the illegal amount value for the exceptions.
private final static short ILLEGAL_AMOUNT = 1;

// the maximum balance in this purse.
private static final short maximumBalance = 10000;

// the current balance in this purse.
private short balance;

/* Security part of declarations */

// the Security Object necessary to credit the purse
private ProviderSecurityDomain securityObject = null;

// the security channel number
byte secureChannel = (byte)0xFF;

// the authentication status
private boolean authenticationDone = false;

// authentication Enciphered of Maced ?
private boolean enciphered = false;

// the secure channel status
private boolean channelOpened = false;

// the previous working state
private byte previousState = OPSystem.APPLET_PERSONALIZED;
/**
 * Only this class's install method should create the applet object.
 */
protected OPPurse(byte[] buffer, short offset, byte length)
{
    // data offset is used for application specific parameter.
    // initialization with default offset (AID offset).
    short dataOffset = offset;
    byte V2 = 1;

    if(length > 9) {
        // Install parameter detail. Compliant with OP 2.0.1.

        // | size | content
        // |-----|---------------------------
        // | 1   | [AID_Length]
        // | 5-16| [AID_Bytes]
        // | 1   | [Privilege_Length]
        // | 1-n | [Privilege_Bytes] (normally 1Byte)
        // | 1   | [Application_Proprietary_Length]
        // | 0-m | [Application_Proprietary_Bytes]

        // shift to privilege offset
        dataOffset += (short)( 1 + buffer[offset]);
        // finally shift to Application specific offset
        dataOffset += (short)( 1 + buffer[dataOffset]);
        // checks wrong data length
        if(buffer[dataOffset] != 2)
            // return received proprietary data length in the reason
            ISOException.throwIt((short)(ISO7816.SW_WRONG_LENGTH + offset + length -
            dataOffset));

        // go to proprietary data
        dataOffset++;
        V2 = 2;
    }
    else {
        // Install parameter compliant with OP 2.0.
        if(length != 2)
            ISOException.throwIt((short)(ISO7816.SW_WRONG_LENGTH + length));
    }
    // retrieve the balance value from the APDU buffer
    short value = (short)(((buffer[(short)(dataOffset + 1)]) & 0xFF)
    | ((buffer[dataOffset] & 0xFF) << 8));
// checks initial balance value
if(value > maximumBalance)
    ISOException.throwIt(ISO7816.SW_DATA_INVALID);

// initializes the balance with the APDU buffer contents
balance = value;

// register this instance as an installed Applet
if(V2 == 2)
    register(buffer, (short)(offset + 1), (byte)buffer[offset]);
else
    register();

// ask the system for the Security Object associated to the Applet
securityObject = OPSystem.getSecurityDomain();

// applet is personalized and its state can change
// OPSystem.setCardContentState(OPSystem.APPLET_PERSONALIZED);

// build the new ATR historical bytes
byte[] newATRHistory = new byte[]
{
    // put "OPPurse" in historical bytes.
    (byte)0x4F, (byte)0x50, (byte)0x50, (byte)0x75, (byte)0x72, (byte)0x73, (byte)0x65
};
// !!! ACTIVATED IF INSTALL PRIVILEGE IS "Default Selected" (0x04). !!!
// !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
// change the default ATR to a personalized's one
// OPSystem.setATRHistBytes(newATRHistory, (short)0, (byte)newATRHistory.length);

/**
 * Method installing the applet.
 * @param installparam the array containing installation parameters
 */
```
* @param offset the starting offset in installparam
* @param length the length in bytes of the data parameter in installparam
* />
public static void install(byte[] installparam, short offset, byte length )
throws ISOException
{

    // applet instance creation with the initial balance
    new OPPurse(installparam, offset, length );
}
/**
* Select method returning true if applet selection is supported.
* @return boolean status of selection.
*/
public boolean select()
{
    // reset security if used.
    // In case of reset deselect is not called
    reset_security();
    // return status of selection
    return true;
}
/**
* Deselect method.
*/
public void deselect()
{
    // reset security if used.
    reset_security();
    return;
}
/**
* Method processing an incoming APDU.
* @see APDU
* @param apdu the incoming APDU
* @exception ISOException with the response bytes defined by ISO 7816-4
*/
public void process(APDU apdu) throws ISOException
{
    // get the APDU buffer
    // the APDU data is available in ‘apduBuffer’
    byte[] apduBuffer = apdu.getBuffer();
if( authenticationDone &&
    ((apduBuffer[ISO7816.OFFSET_CLA] & 0x0F) == 4))
    enciphered = true;
else
    enciphered = false;

// the "try" is mandatory because the debit method
// can throw a javacard.framework.UserException
try
{
    switch(apduBuffer[ISO7816.OFFSET_INS])
    {
    case INS_VERIFY_PIN :
        if( apduBuffer[ISO7816.OFFSET_CLA] == CLA_OPPURSE ||
            apduBuffer[ISO7816.OFFSET_CLA] == CLA_OPPURSE_SM)
            verifyPIN(apdu);
        else
            ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
        break ;
    case INS_GET_BALANCE :
        if( apduBuffer[ISO7816.OFFSET_CLA] == CLA_OPPURSE ||
            apduBuffer[ISO7816.OFFSET_CLA] == CLA_OPPURSE_SM)
            getBalance(apdu) ;
        else
            ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
        break ;
    case INS_DEBIT :
        if( apduBuffer[ISO7816.OFFSET_CLA] == CLA_OPPURSE ||
            apduBuffer[ISO7816.OFFSET_CLA] == CLA_OPPURSE_SM)
            debit(apdu) ;
        else
            ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
        break ;
    case INS_CREDIT :
        if( apduBuffer[ISO7816.OFFSET_CLA] == CLA_OPPURSE ||
            apduBuffer[ISO7816.OFFSET_CLA] == CLA_OPPURSE_SM)
            credit(apdu) ;
        else
            ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
        break ;
    case INS_INIT_UPDATE :
if(apduBuffer[ISO7816.OFFSET_CLA] == CLA_INIT_UPDATE)
  // call initialize/update security method
  init_update(apdu);
else
  // wrong CLA received
  ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
break;

case INS_EXTERNAL_AUTHENTICATE:
  if(apduBuffer[ISO7816.OFFSET_CLA] == CLA_EXTERNAL_AUTHENTICATE)
    // call external/authenticate security method
    external_authenticate(apdu);
  else
    // wrong CLA received
    ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
break;

case ISO7816.INS_SELECT:
  break;

default:
  // The INS code is not supported by the dispatcher
  ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
  break;
  } // end of the switch
} // end of the try
catch(UserException e)
{
  // translates the UserException in an ISOException.
  if(e.getReason() == ILLEGAL_AMOUNT)
    throw new ISOException(ISO7816.SW_DATA_INVALID);
  }

 private void verifyPIN(APDU apdu)
{
  // get APDU data
}
apdu.setIncomingAndReceive();

// Decrypt APDU buffer
unwrapAPDU(apdu);
// get APDU buffer
byte[] apduBuffer = apdu.getBuffer();
// check that the PIN is not blocked
if((OPSystem.getTriesRemaining() == 0)
&& (OPSystem.getCardContentState() != OPSystem.APPLET_BLOCKED)) {
    previousState = OPSystem.getCardContentState();
    OPSystem.setCardContentState(OPSystem.APPLET_BLOCKED);
}
// Pin format for OP specification
//
//  |type(2),length|nibble(1),nibble(2)|nibble(3),nibble(4)|...|nibble(n-1),nibble(n)|
//
// get Pin length
byte length = (byte)(apduBuffer[ISO7816.OFFSET_LC] & 0x0F);
// pad the PIN ASCII value
for(byte i=length; i<0x0E; i++)
{
    // only low nibble of padding is used
    apduBuffer[ISO7816.OFFSET_CDATA + i] = 0x3F;
}
// fill header TAG
apduBuffer[0] = (byte)((0x02 << 4) | length);
// parse ASCII Pin code
for(byte i=0; i<0x0E; i++)
{
    // fill bytes with ASCII Pin nibbles
    if((i & 0x01) == 0)
        // high nibble
        apduBuffer[(i >> 1)+1] = (byte)(apduBuffer[ISO7816.OFFSET_CDATA + i] & 0x0F) << 4);
    else
        // low nibble
        apduBuffer[(i >> 1)+1] |= (byte)(apduBuffer[ISO7816.OFFSET_CDATA + i] & 0x0F);
}
// verify the received PIN
// !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
// !!! WARNING PIN HAS TO BE INITIALIZED BEFORE USE !!!
// !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
if(OPSystem.verifyPin(apdu, (byte)0))
{
    // set PIN validity flag

validPIN = true;
// if applet state is BLOCKED then restore previous state (PERSONALIZED)
if(OPSystem.getCardContentState() == OPSystem.APPLET_BLOCKED)
  OPSystem.setCardContentState(previousState);
return;

// the last nibble of returned code is the number of remaining tries
ISOException.throwIt((short)(SW_PIN_FAILED + OPSystem.getTriesRemaining()));
}

/**
 * Performs the "getBalance" operation on this counter.
 * @param apdu The APDU to process.
 */
private void getBalance( APDU apdu )
{
  // check valid Applet state
  if(OPSystem.getCardContentState() == OPSystem.APPLET_BLOCKED)
    ISOException.throwIt(ISO7816.SW_COMMAND_NOT_ALLOWED);

  // get the APDU buffer
  byte[] apduBuffer = apdu.getBuffer();
  if(apduBuffer[ISO7816.OFFSET_CLA] == CLA_OPPURSE_SM)
    apdu.setIncomingAndReceive();

  // Decrypt APDU buffer
  unwrapAPDU(apdu);

  // writes the balance into the APDU buffer after the APDU command part
  apduBuffer[5] = (byte)(balance >> 8) ;
  apduBuffer[6] = (byte)balance ;

  // sends the APDU response
  // switches to output mode
  apdu.setOutgoing() ;
  // 2 bytes to return
  apdu.setOutgoingLength((short)2) ;
  // offset and length of bytes to return in the APDU buffer
  apdu.sendBytes((short)5, (short)2) ;
}

/**
 * Performs the "debit" operation on this counter.
 *
// check valid Applet state
if(OPSystem.getCardContentState() == OPSystem.APPLET_BLOCKED)
   ISOException.throwIt(ISO7816.SW_COMMAND_NOT_ALLOWED);

// the operation is allowed only if master pin is validated
if(!validPIN)
   ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);

// get the APDU buffer
byte[] apduBuffer = apdu.getBuffer();
apdu.setIncomingAndReceive();

// Decrypt APDU buffer
unwrapAPDU(apdu);

// Gets the length of bytes to received from the terminal and receives them
// If does not receive 4 bytes throws an ISO.SW_WRONG_LENGTH exception
if(apduBuffer[4] != 2) {
   ISOException.throwIt(ISO7816.SW_WRONG_LENGTH) ;
}

// Reads the debit amount from the APDU buffer
// Starts at offset 5 in the APDU buffer since the 5 first bytes
// are used by the APDU command part
short amount = (short)(((apduBuffer[6]) & (short)0x000000FF)
   | ((apduBuffer[5] << 8 ) & (short)0x0000FF00));

// tests if the debit is valid
if((balance >= amount) && (amount > 0)) {
   // does the debit operation
   balance -= amount ;

   // writes the new balance into the APDU buffer
   // (writes after the debit amount in the APDU buffer)
apduBuffer[7] = (byte)(balance >> 8) ;
apduBuffer[8] = (byte)balance ;

   // sends the APDU response
   apdu.setOutgoing() ; // Switches to output mode
apdu.setOutgoingLength((short)2); // 2 bytes to return
// offset and length of bytes to return in the APDU buffer
apdu.sendBytes((short)7, (short)2);
}
else
   // throw a UserException with illegal amount as reason
   throw new UserException(ILLEGAL_AMOUNT);
}

/**
 * Performs the "credit" operation on this counter. The operation is allowed only
 * if master pin is validated
 *
 * @param apdu       The APDU to process.
 * @exception ISOException If the APDU is invalid or if the amount to credit
 *                         is invalid.
 *
 * private void credit(APDU apdu) throws ISOException
 {   // check valid Applet state
   if(OPSystem.getCardContentState() == OPSystem.APPLET_BLOCKED)
     ISOException.throwIt(ISO7816.SW_COMMAND_NOT_ALLOWED);

   // the operation is allowed only if master pin is validated and authentication is done
   if (!validPIN || !authenticationDone)
     ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);

   // get the APDU buffer
   byte[] apduBuffer = apdu.getBuffer();
apdu.setIncomingAndReceive();

   // Decrypt APDU buffer
   unwrapAPDU(apdu);

   // gets the length of bytes to received from the terminal and receives them
   // if does not receive 2 bytes throws an ISO.SW_WRONG_LENGTH exception
   if(apduBuffer[4] != 2)
     throw new ISOException(ISO7816.SW_WRONG_LENGTH);

   // reads the credit amount from the APDU buffer
   // starts at offset 5 in the APDU buffer since the 5 first bytes
   // are used by the APDU command part
   short amount = (short)(((apduBuffer[6]) & (short)0x000000FF) |
                          ((apduBuffer[5] << 8) & (short)0x0000FF00));

   // tests if the credit is valid

113
if(((short)(balance + amount) > maximumBalance) || (amount <= (short)0))
    throw new ISOException(ISO7816.SW_DATA_INVALID) ;
else
    // does the credit operation
    balance += amount ;

    // writes the new balance into the APDU buffer
    // (writes after the credit amount in the APDU buffer)
    apduBuffer[7] = (byte)(balance >> 8) ;
    apduBuffer[8] = (byte)balance ;

    // sends the APDU response
    apdu.setOutgoing() ; // Switches to output mode
    apdu.setOutgoingLength((short)2) ; // 2 bytes to return
    // offset and length of bytes to return in the APDU buffer
    apdu.sendBytes((short)7, (short)2) ;
}

/**
* Performs the "init_update" security operation.
* @param apdu The APDU to process.
*/
private void init_update( APDU apdu )
{
    // receives data
    apdu.setIncomingAndReceive();
    // checks for existing active secure channel
    if(channelOpened)
    {
        // close the opened security channel
        try
        {
            securityObject.closeSecureChannel(secureChannel);
        }
        catch(CardRuntimeException cre2)
        {
            // channel number is invalid. this case is ignored
        }
        // set the channel flag to close
        channelOpened = false;
    }
    try
    {
        // open a new security channel
        secureChannel = securityObject.openSecureChannel(apdu);
    }
}
// set the channel flag to open
channelOpened = true;
// get expected length
short expected = apdu.setOutgoing();
// send authentication result
// expected length forced to 0x1C
apdu.setOutgoingLength((byte)0x1C);
apdu.sendBytes(ISO7816.OFFSET_CDATA, (byte)0x1c);
}
catch(CardRuntimeException cre)
{
    // no available channel or APDU is invalid
    ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);
}
}

/**
 * Performs the "external_authenticate" security operation.
 *
 * @param apdu The APDU to process.
 */
private void external_authenticate( APDU apdu )
{
    // receives data
    apdu.setIncomingAndReceive();
    // checks for existing active secure channel
    if(channelOpened)
        try
            {
                // try to authenticate the client
                securityObject.verifyExternalAuthenticate(secureChannel, apdu);
                // authentication succeed
                authenticationDone = true;
            }
        catch(CardRuntimeException cre)
            {
                // authentication fails
                authenticationDone = false;
                // close the opened security channel
                try{
                    securityObject.closeSecureChannel(secureChannel);
                } catch(CardRuntime Exception cre2) {
                    // channel number is invalid. this case is ignored
                }
            }
// set the channel flag to close
channelOpened = false;

// send authentication result
ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);

// send authentication result
ISOException.throwIt(ISO7816.SW_NO_ERROR);

else

ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);

/**
 * The "reset_security" method close an opened secure channel if exist.
 * @return void.
 */
public void reset_security()
{
  // close the secure channel if opened.
  if(secureChannel != (byte)0xFF)
  {
    try
    {
      // close the opened security channel
      securityObject.closeSecureChannel(secureChannel);
    }
    catch(CardRuntimeException cre2)
    {
      // channel number is invalid. this case is ignored
    }

    // reset security parameters
    secureChannel = (byte)0xFF;
    channelOpened = false;
    // set validity flags to false
    validPIN = authenticationDone = false;
  }
  return;
}

/**
 * The "unwrapAPDU" method decrypt APDU buffer if needed (bit 0x04 up in instruction
 * class).
 * @param APDU The apdu to decrypt
 */
return void.
*/
public void unwrapAPDU(APDU apdu) {
    byte[] apduBuffer = apdu.getBuffer();
    if (authenticationDone && (apduBuffer[ISO7816.OFFSET_CLA] == CLA_OPPURSE_SM))
        securityObject.unwrap(secureChannel, apdu);
}
Using Sun’s Converter Tool

Although the work of converting an applet’s Class files to CAP files is best done using the GxpConverter tool, described in “Chapter 3 - Using the GxpConverter”, you can use also Sun’s Converter tool. The major advantage of GxpConverter over Sun’s Converter is that it generates load files for multiple target card types in sequence, whereas Sun’s Converter can only generate files for a single card.

The Converter can only convert an applet’s classes—a client application’s code must not be used as input to the Converter.

Launching the Converter from the Command Line

The command line is:

```
java com.sun.javacard.converter.Converter [options] package_name 
package_aid major_version.minor_version
```

Where options are:

- `-classdir rootdirectory`
  
  `rootdirectory` specifies the root directory in which the Converter looks for Class files. If this option is not specified, the Converter uses the current directory.

- `-i`
  
  Instructs the Converter to support the 32-bit integer type.

- `-exportpath listofdirectories`
  
  The directories in which the Converter looks for export files. The separator character for multiple paths is platform dependent. If this option is not specified, the Converter attempts to load the export file from the CLASSPATH.

- `-exportfile`
  
  Instructs the Converter to use the token mapping from the predefined export file of the package being converted. The Converter looks for the export file in the `exportpath`. 
-applet  AID  class_name
    Sets the applet AID and the class that contains the install() method for the applet.
    If the package contains multiple applets, this option must be repeated for each applet.

-d  outputdirectory
    Sets the directory in which to place the output files.

-out  [JCA]  [EXP]  [CAP]
    Instructs the Converter to output the JCA file, the Export File, and/or the CAP file. By
default (if this option is not specified), the Converter outputs the CAP file and (if the
package is exportable) the Export file.

-v, -verbose
    Prints progress and warning messages.

-profile
    Profiles the CAP file.

-V, -version
    Prints the Converter version string.

-help
    Prints this parameter description.

-debug
    Generates debugging information.

-nowarn
    Instructs the Converter not to report warning messages.

-mask
    Indicates this package is for mask.

The arguments are:

package_name
    Fully-qualified name of the package to convert.

package_aid
    The 5- to 16-byte AID of the package. Each byte must be separated by a colon (":")

major_version.minor_version
    User-defined version of the package.
Using Sun’s Converter Tool

**Input Files.** The files input to the Converter are Java class files named with the `.class` suffix. There can be several class files making up a package. All the class files for a package must be located in the same directory under the root directory, following the Java naming conventions. The root directory of the class hierarchy can be set from the command line using the `-classdir` option. If this option is not specified, the converter uses the current user directory as the root.

**Output Files.** The name of the Export file and the JCA file must be the last portion of the package specification followed by the extensions `.exp` and `.jca` respectively. By default, the files output from the converter are written to a directory called JavaCard, a subdirectory of the input package’s directory.

**Example.** For a GXPPro–R3 target (this needs the GXPPro–R3 card profile):

```
java -classpath C:\gemplus\gemxpresso.rad3\lib\converter\sun\1.2\converter.jar; com.sun.javacard.converter.Converter -classdir "C:\gemplus\GemXpressoRAD3\examples\GXPPRO-R3\Purse\out" -exportpath "C:\gemplus\gemxpresso.rad3\resources\exportfiles\GXPPRO-R3" -d "C:\gemplus\gemxpresso.rad3\examples\Purse\oncard" -out EXP JCA -applet 0xA0:0x00:0x00:0x00:0x18:0xFF:0x00:0x00:0x00:0x00:0x00:0x00:0x00:0x00:0x01 1.0
```

**Note:** In the previous example, there are no spaces before or after the colons (:).
Key Diversification

When using the key file editor to define Card Manager keys, you must choose a diversification option. See “Using the Key File Editor” on page 43. The three options are:

- None
- VISA
- VISA2

For the first option, the key is not diversified. For the VISA and VISA2 options, the key is diversified using the same method but with different diversification data.

Diversification Method

The Card Manager daughter keys (KDC\textsubscript{AUTH/ENC}, KDC\textsubscript{MAC}, and KDC\textsubscript{KEK}) are diversified from the mother key (KMC). The diversification method uses 3DES\_ECB with a 16-byte key (KMC) and 16 bytes of diversification data, D.

The data diversification method is as follows:

\[ D = D_{\text{left}} \parallel D_{\text{right}} \]

\[ K_D = K_{D_{\text{left}}} \parallel K_{D_{\text{right}}} \]

\[ K_{D_{\text{left}}} = 3\text{DES}(D_{\text{left}}, \text{KMC}) \]

\[ K_{D_{\text{right}}} = 3\text{DES}(D_{\text{right}}, \text{KMC}) \]
Diversification Data

The following tables show the diversification data for the Card Manager keys for the VISA and VISA2 options.

<table>
<thead>
<tr>
<th>Key</th>
<th>Diversification data</th>
</tr>
</thead>
<tbody>
<tr>
<td>$KDC_{AUTH/ENC}$</td>
<td>FFh FFh $</td>
</tr>
<tr>
<td>$KDC_{MAC}$</td>
<td>00h 00h $</td>
</tr>
<tr>
<td>$KDC_{KEK}$</td>
<td>F0h F0h $</td>
</tr>
</tbody>
</table>

Table 9 - VISA Key Diversification Data

Where:

card serial number is 8 bytes taken from the CPLC data.

<table>
<thead>
<tr>
<th>Key</th>
<th>Diversification data</th>
</tr>
</thead>
<tbody>
<tr>
<td>$KDC_{AUTH/ENC}$</td>
<td>xxh xxh $</td>
</tr>
<tr>
<td>$KDC_{MAC}$</td>
<td>xxh xxh $</td>
</tr>
<tr>
<td>$KDC_{KEK}$</td>
<td>xxh xxh $</td>
</tr>
</tbody>
</table>

Table 10 - VISA2 Key Diversification Data

Where:

xxh xxh is the last (rightmost) two bytes of the Card Manager AID.

IC Serial Number is taken from the CPLC data.
Checking PC/SC Installation

If you want to access PC/SC–compliant terminals, you can do this by using the special “bridge” described in “Accessing PC/SC Terminals” on page 13. This appendix describes how to configure this bridge under Windows platforms. If you are using the RedHat Linux a or Sun Solaris platforms, go to the website: www.linuxnet.com.

The steps in configuring the bridge are:

1. Check that a PC/SC-compliant card reader has been correctly installed, that is, the PC/SC reader driver has been activated.

2. Ensure that PC/SC is installed and running on the local workstation, that is, the PC/SC Smart Card Resources Manager service has been started.

3. Edit the opencard.properties file.

The procedure for steps 1 and 2 depends on the version of Windows. Read the section corresponding to your platform. Step 3 is described in “Accessing PC/SC Terminals” on page 13.

Note: For the latest version of the PC/SC reader driver and additional information about PC/SC, visit the Gemplus site www.gemplus.com/developers.
Checking the PC/SC Reader Driver Installation

Windows 98

1. Select Start>Settings>Control Panel to open the Control Panel window.

2. From the Control Panel, double-click the System icon. This opens the System Properties window.

3. Click the Device Manager tab.
4. Select your reader and click **Properties**. This action displays the following window:

![Smart Card Reader Properties Window](image)

**Figure 44 - Smart Card Reader Properties Window**

5. Clear the box **Disable in this hardware profile**.

6. Click **OK** to close the window.
Windows NT4

1. Select Start>Settings>Control Panel to open the Control Panel window.

2. From the Control Panel, double-click the Devices icon. This opens the Devices window:

3. From the list, select GrSerial.
4. If the status is not “started”, click **Start**.
5. Click **Close** to close the window.

**Windows 2000**

1. Select **Start>Settings>Control Panel** to open the **Control Panel** window as shown in the following window:

![Figure 47 - The Control Panel Window (Windows 2000)](image)

2. From the **Control Panel**, double-click the **System** icon.
3. In the **System Properties** window, click the **Hardware** tab.
4. Click **Device Manager**.
5. In the **Device Manager** window, expand the Smart card readers folder and select your PC/SC reader from the list (for example Gemplus GemPC410 Serial Smart Card Reader).
6. Right-click and select **Properties**. This opens the **Reader Properties** window as shown.

![Reader Properties Window](image)

**Figure 48 - Reader Properties Window**

7. In **Device usage**, select **Use this device (enable)** and click **OK** to return to the Control Panel.

**Starting the PC/SC Smart Card Resources Manager**

**Windows 98**

In Windows 98 you do not need to start the Smart Card Resource Manager,
Checking PC/SC Installation

Windows NT4

1. From Control Panel, double-click the Services icon. This displays the following window:

   ![Figure 49 - The Services Window]

2. From the list, select Smart Card Resource Manager.
3. If the status is not “started”, click Start.
4. Click Close to close the window.

Windows 2000

1. From the Control Panel, double-click the Administrative Tools icon.
2. Double-click the Services icon. This opens the Services window as shown:

   ![Figure 50 - Services Window (Windows 2000)]
3. If Smart Card does not say “started”, either double-click **Smart Card** in the list, or right-click it and select **Properties**. Both actions open the **Smart Card Properties** window as shown:

![Smart Card Properties Window](image)

**Figure 51 - Smart Card Properties Window**

4. Click **Start**, then **OK**.
# Terminology

## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AID</td>
<td>application identifier</td>
</tr>
<tr>
<td>APDU</td>
<td>application protocol data unit</td>
</tr>
<tr>
<td>API</td>
<td>application programming interface</td>
</tr>
<tr>
<td>ATF</td>
<td>APDU trace format</td>
</tr>
<tr>
<td>ATR</td>
<td>answer to reset</td>
</tr>
<tr>
<td>CAD</td>
<td>card acceptance device</td>
</tr>
<tr>
<td>CAP</td>
<td>card applet package</td>
</tr>
<tr>
<td>CBC</td>
<td>cipher block chaining</td>
</tr>
<tr>
<td>CLA</td>
<td>class byte</td>
</tr>
<tr>
<td>CPLC</td>
<td>card production life cycle</td>
</tr>
<tr>
<td>DES</td>
<td>data encryption standard</td>
</tr>
<tr>
<td>ECB</td>
<td>electronic code book</td>
</tr>
<tr>
<td>EXP</td>
<td>Java card export file</td>
</tr>
<tr>
<td>FSD</td>
<td>full serialized data</td>
</tr>
<tr>
<td>GSE</td>
<td>Gemplus simulation environment</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>GSM</td>
<td>global system for mobile communications</td>
</tr>
<tr>
<td>GXCF</td>
<td>XML-based script format</td>
</tr>
<tr>
<td>ICV</td>
<td>initial chaining vector</td>
</tr>
<tr>
<td>IDE</td>
<td>integrated development environment</td>
</tr>
<tr>
<td>INS</td>
<td>instruction byte</td>
</tr>
<tr>
<td>IS</td>
<td>international sample (card version)</td>
</tr>
<tr>
<td>ISO</td>
<td>International Standards Organization</td>
</tr>
<tr>
<td>JAR</td>
<td>Java archive file</td>
</tr>
<tr>
<td>JCA</td>
<td>Java card assembler</td>
</tr>
<tr>
<td>JCRE</td>
<td>Java card runtime environment</td>
</tr>
<tr>
<td>JCVM</td>
<td>Java card virtual machine</td>
</tr>
<tr>
<td>JDK</td>
<td>Java development kit</td>
</tr>
<tr>
<td>JVM</td>
<td>Java virtual machine</td>
</tr>
<tr>
<td>Lc</td>
<td>data length</td>
</tr>
<tr>
<td>Le</td>
<td>expected length of data to be returned</td>
</tr>
<tr>
<td>MAC</td>
<td>message authentication code</td>
</tr>
<tr>
<td>OCF</td>
<td>OpenCard framework</td>
</tr>
<tr>
<td>OP</td>
<td>open platform</td>
</tr>
<tr>
<td>PC/SC</td>
<td>personal computer/smart card</td>
</tr>
<tr>
<td>PIX</td>
<td>proprietary identifier extension</td>
</tr>
<tr>
<td>PK</td>
<td>public key</td>
</tr>
<tr>
<td>RAD</td>
<td>rapid application development</td>
</tr>
<tr>
<td>RID</td>
<td>registered identifier</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>-------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>SAP</td>
<td>simulator applet package (GSE-proprietary load format)</td>
</tr>
<tr>
<td>SSD</td>
<td>system serialized data</td>
</tr>
<tr>
<td>TCP/IP</td>
<td>transmission control protocol/internet protocol</td>
</tr>
<tr>
<td>VM</td>
<td>virtual machine</td>
</tr>
</tbody>
</table>
Glossary

Applet
In Java Card terminology, a Java Card applet is an independent Java application loaded into a Java Card.

Application Identifier
A string of between 5 and 16 bytes that identifies a package or an application in a card and which corresponds to the naming scheme defined in ISO7816-5. It may contain a registered application provider number. If it contains no application provider number, then this identification may be ambiguous.

Application Protocol Data Units (APDU)
Standard communication messaging protocol between a card acceptance device and a smart card.

Application Provider
The entity that owns an application and is responsible for the application’s behavior.

ATF file
A Gemplus APDU trace format script file.

ATR
When a card is inserted into a card reader, it stimulates a contact, which provokes the terminal to reset itself by sending a reset signal to the card. The card’s response is called an answer to reset (ATR).

The ATR is described in two standards:
ISO 7816-3: Electronic signals and transmission protocols, which defines the two possible low-level communication (or transport) protocols between the terminal and the card. It is strongly advised to refer to this standard for details.
ISO 7816-4: Interindustry commands for interchange, which defines a set of standard commands for smart cards, as well as a hierarchical file system structure for cards. These commands are the basis of most existing card protocols.

Bytecode
A VM instruction code as a sequence of binary bytes.

CAP files
A file format that can be loaded into a card. CAP files are generated by the GxpConverter tool.

Class files
A compiled Java code file.

Client application
An application that requests services from a server application—typically an applet in the Java Card environment.
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context</td>
<td>An image, stored in a file, of the current contents of a card, including the AID and life cycle status of each package and applet on the card. Loading a context into the GSE (card simulator) is equivalent to inserting a personalized smart card into a card reader. See also “Full context” and “System context”.</td>
</tr>
<tr>
<td>Conversion</td>
<td>A post-compilation step necessary to convert Java Class bytes into a form (.CAP files) understood by the card.</td>
</tr>
<tr>
<td>Converter</td>
<td>A utility provided by Sun to convert Java Class files into a form understood by the card.</td>
</tr>
<tr>
<td>Full context</td>
<td>An image, stored in a file, of system packages and applets, custom packages and applets stored on a card, for example, packages that you have uploaded and installed. See also “Context”.</td>
</tr>
<tr>
<td>GSE</td>
<td>Gemplus simulation environment. A software environment that simulates a Java Card context. Used for testing applet functionality before loading the applet into a card.</td>
</tr>
<tr>
<td>JAR file</td>
<td>Java archive file. A compressed file containing Java Class files, or CAP load files. Used by the GxpLoader, for example.</td>
</tr>
<tr>
<td>Java Card-OP applet</td>
<td>An applet that is fully compliant with the Java Card 2.1.1 and Visa’s Open Platform 2.0.1 specifications.</td>
</tr>
<tr>
<td>Java-OP Card</td>
<td>Smart cards that are fully compliant with both the Java Card 2.1.1 and Visa’s Open Platform 2.0.1 specifications.</td>
</tr>
<tr>
<td>JCA file</td>
<td>A Java card assembler file, generated by the GxpConverter or Sun’s Converter tool.</td>
</tr>
<tr>
<td>JCardManager</td>
<td>A GemXpresso RAD III tool that simulates a client application, enabling a user to exchange commands with an applet installed in either a card or the GSE.</td>
</tr>
<tr>
<td>Key index</td>
<td>Identifies a key within a key set.</td>
</tr>
<tr>
<td>Key set</td>
<td>A set of encryption keys used to authenticate the owner.</td>
</tr>
<tr>
<td>Key version</td>
<td>A number between 0 and 127 identifying the key version to use.</td>
</tr>
<tr>
<td>Load File</td>
<td>The physical data files that are uploaded to Java-OP cards to modify the card’s contents.</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MAC</td>
<td>A symmetric cryptographic transformation of data that provides data origin authentication and data integrity.</td>
</tr>
<tr>
<td>GxpLoader</td>
<td>A GemXpresso RAD III tool used to convert, load, and install files into target cards or the GSE (card simulator) in one step.</td>
</tr>
<tr>
<td>Package</td>
<td>A Java term to describe a collection of related classes and interfaces. A package can contain, for example, a client application, or several applets.</td>
</tr>
<tr>
<td>PC/SC</td>
<td>The PC/SC (personal computer/smart card) Workgroup was formed in May 1996 to resolve interoperability issues between PCs and smart cards. Members include Gemplus, IBM, Sun Microsystems, Toshiba, Groupe Bull, Hewlett-Packard, Microsoft, Schlumberger, and Siemens Nixdorf.</td>
</tr>
<tr>
<td>Project</td>
<td>A collection of one or more applets, together with the necessary system classes, with or without libraries. Also used to describe an IDE (VisualCafé or JBuilder) project.</td>
</tr>
<tr>
<td>RID</td>
<td>The first five bytes of an AID, registered with ISO, indicating the Application Provider.</td>
</tr>
<tr>
<td>SCR file</td>
<td>A Sun script file.</td>
</tr>
<tr>
<td>Scripts</td>
<td>A list of commands. Gemplus scripts (.atf) or Sun scripts (.scr) can be exchanged with a card or the GSE.</td>
</tr>
<tr>
<td>Size tool</td>
<td>A tool used to calculate the amount of memory that an applet occupies in a card. Provided by Sun as a command-line utility, in GemXpresso RAD III the Size tool is integrated in the GxpConverter utility.</td>
</tr>
<tr>
<td>System context</td>
<td>An image, stored in a file, containing details of all system packages and applets stored on a card. See also “Context”.</td>
</tr>
<tr>
<td>Target</td>
<td>The entity in which a Java Card applet or package is eventually installed. The target can be either the GSE (Gemplus simulation environment) or a specific type of card in a card reader.</td>
</tr>
<tr>
<td>Terminology</td>
<td>Definition</td>
</tr>
<tr>
<td>-------------</td>
<td>------------</td>
</tr>
<tr>
<td>Target file</td>
<td>The target file contains the characteristics of the card. It is required by the GxpLoader tool.</td>
</tr>
<tr>
<td>Template</td>
<td>A predefined project type provided by GemXpresso RAD III for the JBuilder and VisualCafé IDEs. Each template provides a wizard to help you specify project-related information. Some templates generate ready-to-compile skeleton Java code, to which the developer need only add the applet or client application's functionality.</td>
</tr>
</tbody>
</table>
For More Information

Standards and Specifications

References are made to the following standards and documents:
Documents marked with an asterisk (*) are included on the GemXpresso RAD III CD ROM, depending on the card profile(s) installed.

For GemXpresso 211 V2 and GemXpresso 211V2_IS:
- Sun’s Java Card 2.1 Virtual Machine Specification*.
- Sun’s Java Card 2.1 Runtime Environment Specification*.
- Open Platform Card Specification 2.0 from Visa.

For GemXpresso 211 PK, GemXpresso 211 PK_IS, GemXpresso Lite Generic and GemXpresso Pro R3:
- Sun’s Java Card 2.1.1 Virtual Machine Specification*.
- Sun’s Java Card 2.1.1 Runtime Environment Specification*.
- Open Platform Card Specification 2.0.1 from Visa.
- Open Platform Card Specification 2.0.1’ from Visa.

For all GemXpresso cards:
- ISO 7816-3: Electronic signals and transmission protocols.
  Defines the characteristics of the electronic signals exchanged between the card and the terminal, and the two possible low-level communication protocols that can be used. (The T=0 protocol is an asynchronous half-duplex character transmission protocol; the T=1 protocol is an asynchronous half-duplex block transmission protocol).
- ISO 7816-4: Interindustry commands for interchange.
  Defines a set of standard commands for smart cards, as well as a hierarchical file
system structure for cards. These commands are the basis of most existing card protocols.

- ISO 7816-5: Numbering system and registration procedure for application identifiers. This standard defines an unambiguous naming system for applets.

Recommended Reading

GemXpresso RAD III Documentation

GemXpresso RAD III provides a suite of documents that complement this manual:


- GemXpresso RAD III Version 3.2 Command Reference, which lists the commands available for communicating with a GemXpresso card. A different version of this document is available for each supported card type:
  - GemXpresso 211 v2 (DOC105185)
  - GemXpresso 211 v2 IS (DOC105186)
  - GemXpresso 211 PK (DOC105187)
  - GemXpresso 211 PK IS (DOC105188)
  - GemXpresso Lite Generic (DOC108064)
  - GemXpresso Pro R3 (DOC107568)

Applet Development

For more information about Java applet development for smart cards, refer to:


Java: The Language and the Platform

Bruce Eckel. *Thinking in Java*. MindView, Inc.
Available as a free download at [http://www.mindview.net/Books](http://www.mindview.net/Books).

This book is a very good reference concerning the practical use of Java, and comes with many examples. It covers all aspects of the Java language, and discusses object technology in depth.


A comprehensive presentation of the Java language and API, written in an accessible style.


This is the reference book about the Java language itself. This book is not intended for beginners, but rather as the undisputed reference for Java programmers. It covers all aspects of the Java language in distinct technical language. It comes with an extensive index.


This is the reference book on the Java Virtual Machine. Required reading for anybody who wishes to implement a Java VM.

Smart Card and Secure Programming


This book is a comprehensive guide to developing applications with Java Card technology. It introduces you to the Java Card platform and features detailed discussions of programming concepts. It also provides a step-by-step Java Card applet development guide to get you up and running.


Another comprehensive reference guide, this one originally written in German. Topics covered include smart card architecture and operating systems, data transfer and transmission protocols, data encryption techniques and security mechanisms, European and international standards, testing, and commercial applications with Europay, Visa, MasterCard, and Intersector Electronic Purse.
Cryptography


A major reference about cryptography. Contains an introduction to all the important concepts in cryptography, as well as a description of the most widely used algorithms, from Enigma to PGP. Plus source code for all of the algorithms!

Web References

The JavaSoft Home Page (http://www.java.sun.com)

The home page for the latest developments in Java. Maintained by Sun.

The Java Card home page (http://java.sun.com/products/javacard)

The Java Card home page, maintained by Sun. The right place to get the latest version of the API specification, at all times.

JavaWorld (http://www.javaworld.com)

Definitely a good resource for articles about Java. They have recently added a column about programming with Java Card. Their tutorials and short presentations of Java features are often of a very good quality.

Global Platform (http://www.globalplatform.org)

Site contains interesting documents for downloading including an overview of the Open Platform Card Specification.
Index

Numerics
3DES
   encryption/decryption  41

A
AMF file (APDU commands list)
   loading  53
   saving  53
answer to reset (ATR)  77
APDU
   comparing responses  55
   introduction  95
   response format  98
API (application programming interface)
   GSE  59
   Java Card 2.1  59
   OCF  10
   Open Platform  59
applet
   creating a new  4
   development  142
   selecting in simulator  49
applet AID
   default  24
   defining for project template  5
applets
   installing in GSE  48
   loading into GSE  47
   viewing contents of with GSE GUI  71
authentication
   creating custom key sets  42

B
base-core package  16
base-opt package  16

calculating package size  32
CAP files
   creating from Class files  41
   diagnosing with CapDump  92
CapDump
   introducing  89
   launching  89
   using  90
   viewing CAP file contents with  92
card
displaying contents of  46
Card Explorer  46
card manager
  AID in card simulator  70
  viewing life cycle state of  70
card production life cycle
  see CPLC
card simulator
  GxpLoader parameters, with deployment file  80
      resetting  48, 71
      selecting applet in  49
      starting and stopping  71
CardService layer  10
CardTerminal layer  10
CBC (cipher block chaining)  41
Class directory, determining  24
Class files
  converting to CAP format  41
  converting to loadable format  21
  specifying in Project Editor  24
CLASSPATH
  setting in VisualCafé  16
client application
  launching GSE from  62
command APDUs  95
commands
  authenticate  53
  Get Card Type (GSE)  47
  Java Card Install (GSE)  48
  Java Card Load (GSE)  47
  list of JCardManager  50
  Reset (GSE)  48
  Restore Context (GSE)  48
  Save Context (GSE)  49
  Select Applet  49
  sending to target card  49
  supported by deployment file  87
  contexts
  full  73
  restoring GSE  48
  saving GSE with JCardManager  49
  system  73
Converter (Sun)  119
converting Class files  32
CPLC  124
CPLC data structure  76
custom projects, creating  7

D
debugging, interactive  62
decryption, performing DES and 3DES  41
default
  applet AID  24
  CPLC status (for GSE)  76
default key set  43
deployment editor
  introduction  83
  running  83
deployment files
  creating  84
  creating with JCardManager  51
  editing with JCardManager  51
  opening  86
  running with JCardManager  51
  use with GxpLoader  80, 81
DES
  encryption/decryption  41
directories
  export file  26
diversification, of static keys  43
diversifying keys  123

E
ECB (electronic code book)  41
EEPROM  17
encryption, performing DES and 3DES  41
export files (.EXP)
   specifying directories for 26
   with GxpConverter 30

F
firewalls 75
formats
   of command APDUs 97
   of response APDUs 98
full context 73

G
GCF scripts, converting to GXCF 93
GCF to GXCF script converter 93
GemXpresso Project files (.GXP) 8
gemxpresso.jar package 16
Get Card Type 47
GSE
   architecture 58
   default key set 77
   launching 60
   launching client applications from 62
   launching from IDE 62
   restrictions 75
   test procedure using 59
GSE GUI 59
gse_gxp211_pk.jar 15
GXCF scripts 93
GxpConverter
   accessing from within JCardManager 31
   accessing from within plug-in 32
   converting Class files 32
   files generated by 29
   overview 29
   starting 31
GxpLoader
   parameters with a card and deployment file 81
   running 79
   with deployment file for card simulator 80

I
IDE
   launching GSE from 62
   using to develop applications 1
   integrated development environment
      see IDE
   interactive debugging 62
   ISO 7816-3 141
   ISO 7816-4 12, 95, 141
   ISO 7816-5 142

J
Jar file (.JAR) 30
Java archive file
   see Jar file
Java Card Assembler (JCA) file 31
Java Card Install command 48
Java Card library 6
Java Card Load command 47
Java Card Virtual Machine 30
Java classes
   main 15, 16
   Java language, restrictions 17
   Java packages
base-core 16
base-opt 16
CardTerminal 10
comm.jar 12
cryptix-gemxpresso.jar 16
cryptix-gemxpresso_is.jar 16
cryptix-jce-api.jar 16
gemplus-rad-terminals 16
gemplus-services 16
gemplus-terminals 16
gemplus-util.jar 16
gemxpresso.jar 16
gse_gxp211_pk.jar 15
javacard.framework 15
javacard.security 15
javacardx.crypto 15
pcsc_wrapper.jar 13
vopcardservice.jar 16
Java VM 58
javacard.framework 15
javacard.security 15
javacardx.crypto 15
JCard Manager
  running 42
JCardManager
  as client application 37
  commands 50
  creating a new project 40
  creating and editing deployment files 51
  list of main functions 37
  opening a new project 40
  overview 37
  running 38
  running deployment files 51
  using scripts 53
JCRE (Java card runtime environment) 59
JDK 1.2 58

K
key
  diversification 123
  key diversification 43
  key file editor 43
  key number 44
  key set, default 43
  key sets
    creating custom 42
    default (GSE) 77
  key sets, creating 44

L
library
  creating a Java Card 6
  life cycle status 70
  limitations of GSE 75

M
MAC 53
main class 15, 16
  in GSE 60
  memory persistence
    see persistence

O
OCF
  API structure 10
  CardService 10
  CardTerminal 10
  configuring 59
  packages 16
  Ocfpcsc1.dll 13
Open Platform
  OCF CardService 10
OpenCard Framework
see OCF
opencard.properties
   configuring for GSE   60
   configuring to access PC/SC terminals   13
   configuring to access PCSC terminals   125

P
package
   specifying version number   24
package AID   5,   24
package name
   specifying in Project Editor   24
packages
   calculating size of   32
   viewing contents of with card simulator   71
persistence
   overview xi
   with card simulator   75
plug-in   1
project editor
   creating new project   23
   overview   21
   starting   22
project file (.GXP)   21
ProjectEditor.bat   22
projects
   creating custom   7
   creating with JBuilder   2
   creating with VisualCafé   3
   developing without templates   14
   programming guidelines   17
protocols
   T=0   13
   T=1   13

R
recording
a trace file   56
   scripts with JCardManager   55
replaying scripts with JCard Manager   55
Reset command   48
resetting
   card simulator   71
   card simulator with JCard Manager   48
response APDUs   95
   comparing   55
   Restore Context command   48
restrictions of GSE   75

S
SAP Converter   31
SAP files
   generated by gxpConverter   31
   with JCardManager   47
Save Context command   49
sc.ini file   16
script converter
   configuring   94
   running   93
scripts
   recording   55
   replaying   55
   supported formats   54
   using with JCardManager   53
security domain   43
Select Applet command   49
serial number
   default (GSE)   77
simulator
   main class   15
   see card simulator
   simulator commands
Get Card Type  47
Java Card Install  48
Java Card Load  47
Reset  48
Restore Context  48
Save Context  49
Select Applet  49
sending with JCardManager  47
Sun’s Converter  119
system context  73

T
T=0 protocol  13
T=1 protocol  13
target files, creating custom  42
targets
   selecting  45
   selecting for conversion  25
   sending commands to  49
trace file, recording  56
triple DES  41

V
version number (of package)  24
virtual machine
   see Java Card Virtual Machine
VisualCafé
   sc.ini  16
   setting CLASSPATH with  16