



DeLaRue

DE LA RUE
CARD SYSTEMS



GalactIC
Version 1

User Manual

GalactIC
User Manual

© 1998, De La Rue Cartes et Systèmes. All rights reserved.

The information contained in this publication is accurate to the best of De La Rue Cartes et Systèmes' knowledge. However, De La Rue Cartes et Systèmes disclaims any liability resulting from the use of this information and reserves the right to make changes without notice.

Manual references:

PE: 993-099

12NCT: 4311 240 26362

Publication date: 15 December 1998.

CONTENTS

PREFACE	v
Presentation of this guide	v
GalactIC: A definition	v
Purpose	v
Audience.....	v
Structure of this guide	v
Related documentation	v
INSTALLATION	7
Package contents.....	7
Technical requirements.....	7
Hardware	7
Software	7
Installation procedure.....	8
Connecting the reader.....	8
Installing the kit.....	9
THE CONVERTER	11
Product Description.....	11
Environment	11
Functions.....	12
MultiClass Applications	12
Package Linking	12
Variable Types	13
Bytecode Verifier	13
Launching the Converter.....	14
Command Line.....	14
Input files.....	14
Output files.....	14
Conversion Errors.....	15
Conversion Warnings	15
THE LOADER	17
Product Description.....	17
Environment	17
Product Functions.....	17
Loading	18
Sending a file to the smart card	18
Loader*.ini	18
Initialisation	19
De La Rue Java Loader	19
Example – Filling in text fields	19
Other Fields.....	20
Loading	21
Reference.....	22
Fields	22
Flags	22
DEBUGGING ENVIRONMENT (PCOM32)	23
Product Description.....	23
Technical Requirements	23
Launching PCOM32.....	23

Contents

User Interface	24
Main Window.....	24
Example - Command File.....	26
Erasing card data.....	27
Reuse.cmd	27
Command Files.....	28
Introduction	28
Typographic conventions	29
Card commands.....	29
Expected return status	30
Expected return data.....	30
Special characters	31
Comment character	31
Line continuation character.....	31
Indentation.....	31
Card reader directives	32
.INSERT.....	32
.EJECT	32
.POWER_ON.....	32
.POWER_OFF	32
Setup directives	33
.LIST_ON	33
.LIST_OFF.....	33
.STEP_ON.....	33
.STEP_OFF	33
.ERROR_BEEP_ON.....	33
.ERROR_BEEP_OFF	33
.END_BEEP_ON	33
.END_BEEP_OFF	33
.SET_TIME_OUT.....	34
.READER.....	34
File management directives.....	35
.CALL	35
.EXECUTE	36
Loop management directives	37
Buffer management directives	38
The PCOM32 buffers	38
The directives	39
Constant management directives	43
APPENDIX A – ERROR STATUS	45
Reader error status.....	45
Card error status.....	46

PREFACE

Presentation of this guide

GalactIC: A definition

GalactIC is a Java™ powered smart card, optimised to store and run a range of applications written in Java. This application range covers solutions for debit/credit, electronic purse, electronic commerce, loyalty, access control, pay TV, healthcare, identification, mass transit and gambling. The Galactic User Manual includes these tools:

- The Converter
- The Loader
- Debugging environment (PCOM32)

Purpose

The purpose of this GalactIC User Manual is to introduce and explain the three tools comprising the Galactic solution.

Audience

This guide is destined for programmers who know the basics of Java language and are familiar with the basic principles of smart cards.

Structure of this guide

- Chapter 1: Installation
- Chapter 2: The Converter
- Chapter 3: The Loader
- Chapter 4: Debugging environment (PCOM32)

Related documentation

For further information on GalactIC, refer to the following De La Rue guides:

- GalactIC Operating System – Ref. PE 993-098
- Smart Card Reader Java API Reference Guide – Ref. PE 993-097

P r e f a c e

For information on specifications, refer to the following:

ISO/IEC 7816-3 (1989)	Identification cards - Integrated circuit(s) cards with contacts Part 3: Electronic signal and transmission protocols.
ISO/IEC 7816-4 (1995)	Identification cards - Integrated circuit(s) cards with contacts Part 4: Inter industry commands for interchange.
ISO/IEC 7816-5 (1992)	Numbering system and registration procedure for application identifiers.
ISO 9564-1 (1991)	Banking - Personal Identification Number management and security.
ANSI X9.19 (1986)	Financial Institution Retail Message authentication.
EMV (1996)	Integrated Circuit Card Specifications for Payment Systems: Part 1: Electromechanical Characteristics, Logical Interface, and Transmission Protocols (version 3.0). Part 2: Data Elements and Commands (version 3.0). Part 3: Application Selection (version 3.0). Part 4: Security Aspects (Version 3.0).
SUN	The Java Virtual Machine specification v1.0.
SUN	Java Card 2.0 - Programming concepts
SUN	Java Card 2.0 - Language Subset and Virtual Machine Specification
SUN	Java Card 2.0 - Application Programming Interfaces
VISA (optionally)	Visa Open Platform 1.0 (Visa proprietary)

INSTALLATION

Package contents

In this package you will find:

- 1 CD-ROM
- 1 Card Reader with cables
- 3 test cards

Technical requirements

Hardware

Minimal requirements you need to run GalactIC:

- PC Pentium with a screen resolution of 800x600 and 256 colours and 32 Mo of RAM
- Operating System: Windows[®] 95 or Windows[®] NT 4.0

Software

To develop applications, you need a Java development environment such as Symantec[™]Visual Café for Java, PDE V2.0.

To help debugging applications, you also need the Sun Reference Implementation Code. You can download it at the following address:

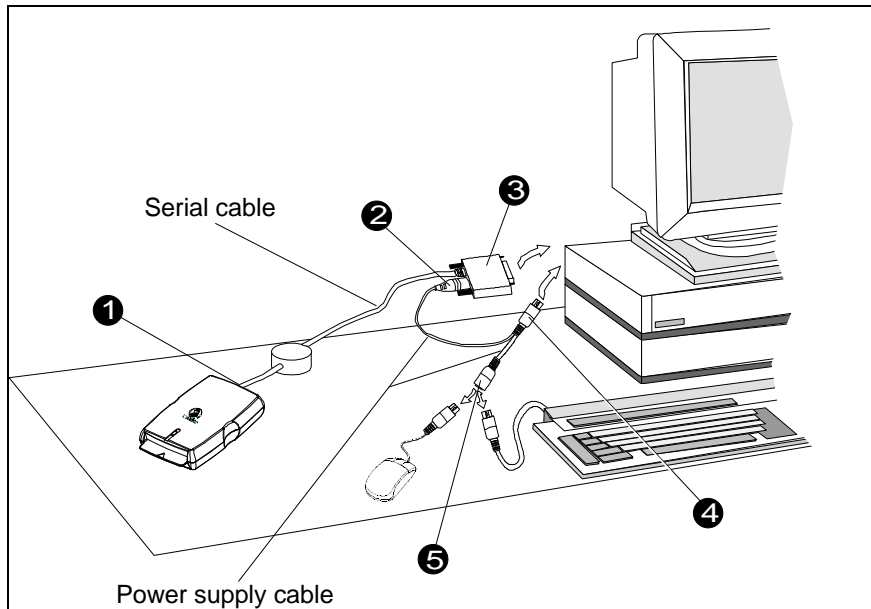
<http://java.sun.com/products/javacard/index.html#spec>

To facilitate the compilation of applets, a set of empty classes is supplied in the kit. These classes are contained in the file `cardclass.zip`. However, they are not needed if the Sun Reference Implementation Code is used instead.

Installation procedure

Before installing the CD-ROM, you must first connect the reader to your PC.

Connecting the reader



- Turn your PC off.
- Plug the RJ45 connector ① into the reader.
- Plug the DB9 part of the connector ③ into the serial port of the PC, tightening the screws.
- Disconnect *either* the PC keyboard *or* mouse. Connect it to part ⑤ of the power supply cable.
- Connect the mini-jack ② to the jack socket of the connector ③.
- Connect part ④ of the power supply cable to the PC mouse *or* keyboard port.

For information on how to configure the reader, see Installing the kit, step 3, on page 9.

Installing the kit

To install the GalactIC kit, proceed as follows:

Step	Action				
1.	<p>Insert the GalactIC CD-ROM in your drive and run setup.exe.</p> <p><u>Result:</u> The installation wizard starts up.</p>				
2.	<p>Follow the instructions displayed, particularly to indicate the destination directory (by default C:\De La Rue\GalactIC) and choose the installation type:</p> <ol style="list-style-type: none"> 1. The "Typical" installation installs all the GalactIC components (Converter, Java Loader, PCOM 32 Debugging Tool, user documentation, samples). The typical installation requires 15 M of free memory. 2. The "Compact" installation only installs the software components. It requires 4 M of free memory. 3. The "Custom" installation allows you to choose the components you want to install. <p><u>Result:</u> The GalactIC kit is installed. The Loader and Debugging Tool icons are created in the task bar. The Smart Card Readers icon is added to the Control Panel.</p>				
3.	<p>Open the Control Panel and double-click on the Smart Card Readers icon to configure the reader.</p>				
4.	<p>Click on the Add... button to create the new reader.</p> <p><u>Result:</u> The Add Logical Connection dialogue box is displayed with three tabs enabling you to set the various parameters.</p>				
5.	<p>Set the following parameters on the Reader tab.</p> <table border="0"> <tr> <td>Logical connection name</td> <td>Name to be assigned to the reader (see the instructions on screen)</td> </tr> <tr> <td>Reader type</td> <td>Type of the reader connected to the PC (De La Rue reader name)</td> </tr> </table>	Logical connection name	Name to be assigned to the reader (see the instructions on screen)	Reader type	Type of the reader connected to the PC (De La Rue reader name)
Logical connection name	Name to be assigned to the reader (see the instructions on screen)				
Reader type	Type of the reader connected to the PC (De La Rue reader name)				
6.	<p>Set the following parameter on the Protocol tab.</p> <table border="0"> <tr> <td>Protocol</td> <td>Protocol to be used when communicating with the reader (in this case Fastnet).</td> </tr> </table>	Protocol	Protocol to be used when communicating with the reader (in this case Fastnet).		
Protocol	Protocol to be used when communicating with the reader (in this case Fastnet).				
7.	<p>Set the following parameter on the Port tab.</p> <table border="0"> <tr> <td>Port</td> <td>Port to which the reader is connected on the PC.</td> </tr> </table>	Port	Port to which the reader is connected on the PC.		
Port	Port to which the reader is connected on the PC.				

THE CONVERTER

Product Description

This section describes the De La Rue Java Card (DLRJC) converter.

The De La Rue Java Card converter takes several class files, created by any Java compiler, and converts them into a single DLRJC install file. The resulting install file is run on a smart card by the De La Rue implementation of the Java Virtual Machine (32-bit MultiClass version).

In addition to creating the install file, the DLRJC has two functions:

- | | |
|-----------------|---|
| Verify | The DLRJC converter verifies that the class files contain only bytecodes included in the subset of the Java language defined for the Javacard. |
| Optimise | In order to comply with smart card technology constraints, the DLRJC converter must reduce the size of the DLRJC install file. Using improved class management and new bytecodes, this file is reduced to between 20% and 30% of its original size. |

Environment

The converter is designed to run on any 32-bit version of Windows.

The converter is a console application and must be launched from a Windows command line.

The Java compiler used must conform to the Java Virtual Machine Specification (Sun Microsystems).

Functions

MultiClass Applications

The converter integrates several class files into a single `.dlrjar` file. This allows applications to create and use objects defined by the user or any objects already defined in the Java Card API.

If, for example, the class `MyApplet` explicitly uses the classes `Class1`, `Class2` and `OwnerPIN`, passing `MyApplet` via the command line to the converter results in a `.dlrjar` file containing the conversion of the class files `MyApplet`, `Class1` and `Class2`.

The class `OwnerPIN` is not included since it belongs to the Java Card API.

Note : The converter looks for the class files in the current directory. If they are not found, the converter looks in the class files package directory. For example, the converter searches class `myPackage.utils.myClass` in the initial directory, then in `.\myPackage\utils\`.

Package Linking

This is the procedure used to instantiate classes (package2) that have been declared in another package (package1).

- Package1 must be compiled.
- The converter creates an export file (user `.api` file).
- Package2 uses the `.api` file during conversion.

For example, `Pack1` declares `class1` and `class2`, its AID is `C`, it generates `C1.API`.

`Pack2` declares `class3` and `class4`, its AID is `C2`, it generates `C2.API`.

`Pack3` instantiates `class1` and `class4`. The user does not have the sources of `Pack1` or `Pack2`, but they have the files `C1.API` and `C2.API`. By using the text file `javaconv.apis`, the user can link packages.

From the position of `Pack3` :

1. Create the text file `javaconv.apis`, for example `[C1.api], [C2.api]`.
2. Enter the text lines containing the full path of `C1.API` and `C2.API`.
3. Convert `Pack3`.

The converter automatically finds the descriptions of `Pack1` and `Pack2`.

Variable Types

The converter allows the use of the following variable types:

- byte
- short
- int (optional) – see Command Line on page 14.
- boolean
- strings
- objects from the Java Card 2.0 API, or user-defined
- arrays of the above variable types

The converter refuses other variable types such as:

- float
- long
- double

Multiarrays and arrays of arrays are not allowed.

Bytecode Verifier

The first step performed by the converter is to verify that the bytecode generated for the virtual machine is valid. The converter checks for:

- potential stack overflow
- type mismatches between actual and formal parameters
- use of unsupported data types
- transfer of control outside the current method

Launching the Converter

Command Line

The converter can be launched from:

- MS DOS or
- Postbuild for users of Visual j++

Enter the command for the converter as follows:

```
JavaConv options < file >
```

<file> is a .class file(s) and the options are:

```

C:\Galactic>javaconv
Usage: JavaConv -a<aid> [-p] [-i] [-l] <classfile> [<classfiles>...]
    -a specify AID
    -i allows usage of type 'int'
    -l generates associated .api file
    -p pauses output between each page
C:\Galactic>
    
```

Option	Description
-a<AID>	Specifies the application AID
-i	Allows use of 'int' type
-l	Generates associated .api file (see Package Linking on page 12)
-p	Pauses output between each page.

Input files

Below are the files necessary to launch the converter:

- The .class files of the application. Any further .class files used by the application are automatically converted.
- The file javaconv.apis and files declared within it. These files are optional (see Package Linking on page 12).

Output files

Conversion results in:

- a .dlrjar file, which is the package file.

and some or all of the following:

- a .cmd file
- a .api file

Note: If an error occurs no output files are created.

Conversion Errors

When an error is encountered the converter does not generate an output file.

The converter may stop execution after several errors, when an error introduces exceptions.

Error	Description
Too many methods in class.	A class cannot have more than 256 methods.
Not compiled in debug mode : 'class'.	The class file has not been compiled in debug mode.
Type 'type' not supported.	The data type is not supported, see command line reference to 'int' data types.
Unsupported bytecode 'bytecode'.	The converter encountered a bytecode not supported by Java Card 2.0.
Can't find enough space for interface methods!	Too many methods have been defined. The limit is 256.
Native methods not supported!	Native methods cannot be used in Java Card applications.
API entry not found.	A method not found in the API entries.
Packages not found.	The accessed package is not in the API or is not a user-defined package.
The <bytecode/macro> verifier crashed!	Internal error.
Different stack on the same location.	Bytecode verification detects stack error.

Conversion Warnings

This section gives a list of warnings given by the converter.

Warning	Description
Stack is not empty before return.	Bytecode verification found that there is no empty stack at the end of the function.
Pop no value from the stack.	Stack underflow in verifier operation.
Pop value type different from bytecode type!	Stack data type control error.

THE LOADER

Product Description

By default the GalactIC card supports a subset of the Visa Open Platform specification. This subset does not support security domains and hence encryption and signature of a file is not supported. This loading mechanism is present in all the test cards supplied with the kit, and can be extended with the Visa mechanism on initialisation, when the card is manufactured by De La Rue Card Systems.

This extension is a configurable option, which supports the mechanisms as defined by Visa International, through the Visa Open Platform specification. This specification is the property of Visa International.

The Loader consists of two parts:

JAVLOAD.EXE	This is an executable file representing the user interface to send commands to the card.
PCOMOP.DLL	This is the dynamic link library which includes the functions and the interface between the reader and the smart card.

Environment

The De La Rue Loader runs under MS – Windows NT. It is designed as a menu–mouse application.

It uses De La Rue Smart Cards drivers and libraries.

To use the Loader you need:

- a physical and logical smart card reader.
- all configuration files installed.

Note: The CD-ROM automatically runs the file Setup.exe (the configuration file). If it does not then you must run Setup.exe manually before using the Loader.

Product Functions

The file to be written to the smart card can be loaded using different functions. These functions conform to the Visa Open Platform specification and are available as configurable options during card initialisation.

These functions are designed to increase the level of security and management of the target file.

Loading

Sending a file to the smart card

The Loader cuts the file into several blocks APDU (Application Protocol Data Unit) and then sends it to the smart card. These blocks can be sent directly or secured.

The following table shows the options available and the domain keys to use.

File	Keys
Sent directly	None
Ciphered and signed	ENC card domain MAC card domain

Loader*.ini

This file contains information used by the Loader to load code onto the smart card. The GalactIC card supports, as a configurable option during initialisation, the Visa Open Platform specification.

Loader*.ini must be created and/or edited using an external editor such as Windows Notepad. Two of the parameters, AID and SID, are automatically updated by the Loader.

There are four fields in Loader*.ini.

1. [Common]

AIDCardDomain=47 61 6C 61 63 74 49 43

This is the AID of the Card Domain and the value is defined before the loading. The user cannot modify it.

2. [KeySet1]

Name="GalactIC demo keys"

KeyIndex=0

KeySetIdentifier=0

KeyMACCardDomain=22334455667788113344556677881122

KeyENCCardDomain=11223344556677882233445566778811

KeyMACSecurityDomain=22334455667788113344556677881122

KeyENCSecurityDomain=11223344556677882233445566778811

This field contains the set of keys defined in the Card Domain and Security Domain. The keys are hexadecimal triple DES keys of 16 bytes. The values are fixed during the Card Domain loading and cannot be modified. The keys are used for authentication and the functions are defined by the Visa Open Platform specification.

3. [AID]

AID1=A0 00 00 00 03 10 10

AID2=A0 00 00 00 03 90 10

AID3=A0 00 00 00 03 60 10

AID4=A0 00 00 00 03 60 20

AID5=A0 00 00 00 03 60 30

4. [SID]

SID1=00 00 00 00

This field contains the Security Domain AID. It is defined by the Visa Open Platform specification.

Initialisation

De La Rue Java Loader

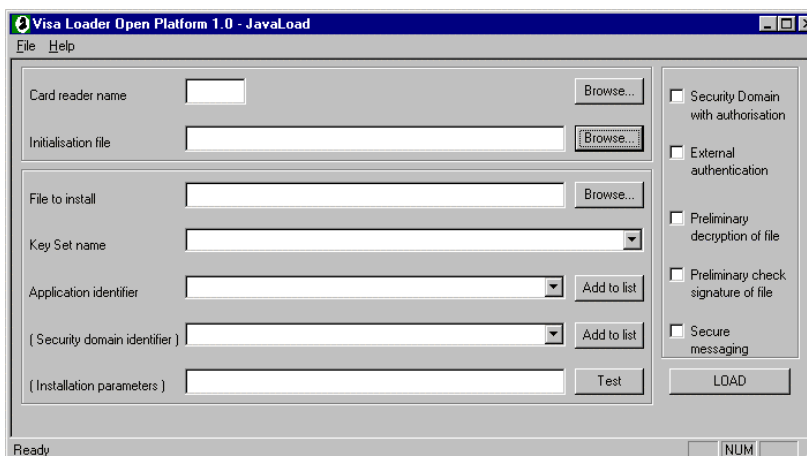
Launching

Two files are necessary:

- the initialisation file (Loader*.ini).
- the file to install (*.dlrjar).

Run Javaload.exe.

The following window opens.



Example – Filling in text fields

The following procedure is an example of filling in the text fields.

1. Use *Browse* to enter Card reader name.

By default this is LC0, which is created during the Setup.exe.



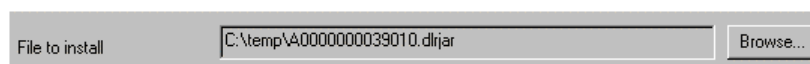
2. Use *Browse* to enter path and name of the initialisation file.

This file is Loader*.ini. The maximum number of characters is 256.



3. Use *Browse* to enter path and name of the file to install onto card.

This is the *.dlrjar file. The maximum number of characters is 256.



4. Use the scroll bar to select *KeySet* section of the initialisation file.

A screenshot of a software interface showing a dropdown menu for 'Key Set name'. The selected option is 'GalactIC demo keys'. The dropdown arrow is visible on the right side of the menu.

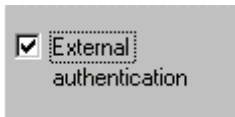
5. Use scroll bar to select *Application* identifier from the list.

This can be from 2 to 32 hexadecimal, even length characters.

A new value can be added to the list by using *Add to list*.

A screenshot of a software interface showing a dropdown menu for 'Application identifier'. The selected option is 'A0000000039010'. To the right of the dropdown is a button labeled 'Add to list'.

6. Select the flag *External authentication*.

A screenshot of a software interface showing a checkbox labeled 'External authentication'. The checkbox is checked, and the text 'External authentication' is displayed next to it.

Now you can install the file onto the smart card (see Loading on page 21).

Other Fields

The following fields are **NOT** used by the GalactIC loader when using the cards from the kit. They are used only if cards support Visa Open Platform specifications (offered as an option).

A screenshot of a software interface showing a dropdown menu for '(Security domain identifier)'. To the right of the dropdown is a button labeled 'Add to list'.A screenshot of a software interface showing a dropdown menu for '(Installation parameters)'. To the right of the dropdown is a button labeled 'Test'.

For a complete description of the fields and flags, see Reference on page 22.

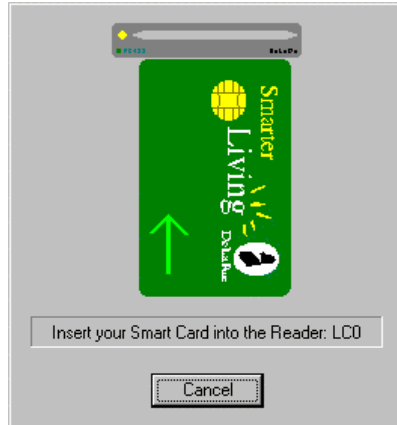
Loading

When all mandatory fields have been filled you can load the smart card.

- 1. Click Load.**

If the smart card is already in the reader the *Applet loading* window appears and the file loads automatically. Go to step 3.

If the smart card is not already in the reader the following windows appears:



- 2. Insert the smart card.**

The file loads automatically.

- 3. When the file finishes loading, click OK.**

- 4. Remove card.**

The loading is complete.

Now you can load another file or exit the application.

Reference

Fields

Field Name	Mandatory	Min. char. Length	Max. char. Length	Character type
Card reader name	Yes	3	3	2 letters, 1 number
Initialisation file	Yes	N/A	64	N/A
File to install	Yes	N/A	64	N/A
Key Set name	Yes	1	32	N/A
Application Identifier	Yes	2	32	Hexadecimal, even length
Security domain identifier	No	2	32	Hexadecimal, even length
Installation parameters	No	2	64	Hexadecimal, even length

Flags

<input type="checkbox"/> Security Domain with authorisation	Allows authentication of the Security Domain with the key KeyMACSecurityDomain before applet loading. Defined by the Visa Open Platform specification and available as an option for GalactIC.
<input type="checkbox"/> External authentication	Mandatory for GalactIC. Protects the card during loading.
<input type="checkbox"/> Preliminary decryption of file	Allows the loader to decipher the applet before loading. Defined by the Visa Open Platform and available as an option for GalactIC.
<input type="checkbox"/> Preliminary check signature of file	Allows verification of the MAC (Message Authentication Code) before applet loading. Defined by the Visa Open Platform and available as an option for GalactIC.
<input type="checkbox"/> Secure messaging	Allows loading of applet with ciphering and signature.

DEBUGGING ENVIRONMENT (PCOM32)

Product Description

The debugging environment consists of :

- the standard Java Development Kit (Symantec Café)
- De La Rue PCOM32 Card Command Processor

PCOM32 sends commands to a smart card via a card-reader interface, and logs the responses to these commands in a text file. The tool allows commands to be sent one at a time under manual control. Alternatively, an entire sequence of commands can be sent as a single operation. In addition there are control constructs that allow the user to loop through command sequences, and to insert break points. Responses to commands are logged in a text file, which is directly accessible from the PCOM32 window.

Technical Requirements

Hardware

You need:

- an IBM-compatible PC connected to at least one De La Rue smart card reader.

Software

You need:

- Windows 95 or NT operating systems.
- a smart card reader library.

Launching PCOM32

PCOM32 can be launched from a command line or Explorer.

1. From a command line, the syntax is:

- PCOM32 or
- PCOM32 <commandfile_name>

`commandfile_name` is the full name of the command file with extension. This parameter is optional.

2. From Explorer:

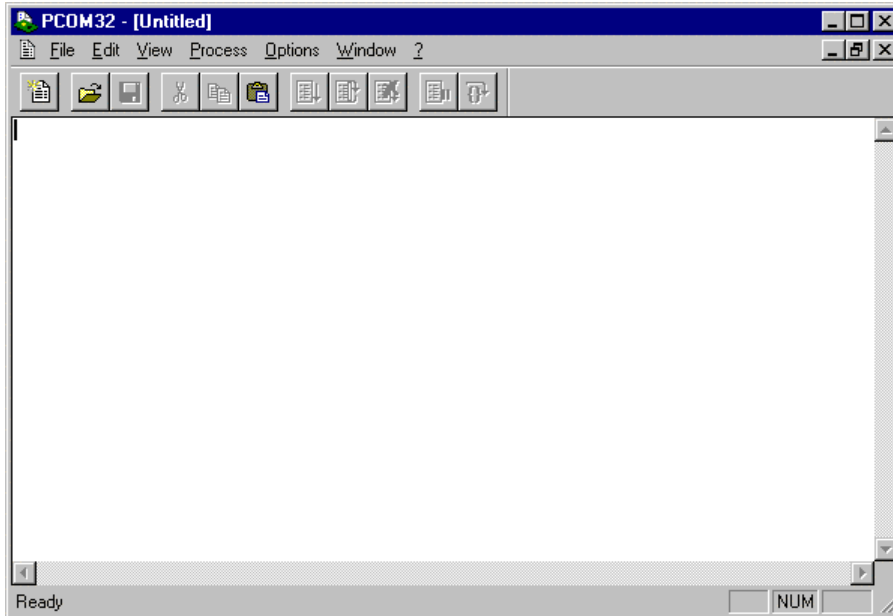
- Click on PCOM32.

User Interface

Main Window

Launch the Debug Tool in the Tools folder.




The following window appears.






The following menus are available:

- File
- Edit
- View
- Process
- Options
- Window
- About

File

Icon	Menu Item	Control Key	Description
	New	Ctrl+N	Creates a new command file.
	Open	Ctrl+O	Opens an existing command file.
-	Close		Closes the active command file.
	Save	Ctrl+S	Saves the active command file.
-	Save As		Saves the active command file with selected name.
-	Recent File		Opens one of the four last opened command files.
-	Exit	.	Exits the application.






Edit

Icon	Menu Item	Control Key	Description
-	Undo	Ctrl+Z	Undoes the last edit action.
	Cut	Ctrl+X	Cuts the selection and moves it to the clipboard.
	Copy	Ctrl+C	Copies the selection to the clipboard.
	Paste	Ctrl+V	Inserts the clipboard contents at the insertion point.
-	Find	Ctrl+F	Finds the specified text.
-	Replace		Replaces the specified text.
-	Go To	Ctrl+G	Moves to a specified line.

View

Menu Item	Description
Toolbar	Toggles toolbar view.
Status Bar	Toggles status bar view.

Process

Icon	Menu Item	Control Key	Description
	Go	F5	Starts or continues the processing of the command file.
	Restart	Alt+F5	Stops and restarts processing from the beginning of the file.
	End	Escape	Aborts processing of the command file.
	Stop	Space	Stops processing of the command file.
	Next Step	Enter	Processes the next directive or command of the command file.

Options

Menu Item	Description
Settings	Allows the configuration of the execution mode, command file logging and error or end of file beeps.
Readers	Allows logical reader configuration.

Window

Menu Item	Description
Cascade	Arranges windows so they overlap.
Tile	Arranges windows as non – overlapping tiles.
Arrange Icons	Arranges icons at the bottom of the window.
Opened File	Activates the opened file.

About

Menu Item	Description
About PCOM32	Displays PCOM32 information, version number and copyright.

Example - Command File

The following command file `chkloy.cmd`, which contains a list of arbitrary commands, shows how commands can be sent one at a time under manual control. The toolbar and control keys allow different options when processing the command file, such as stop, end or restart. Use the Window menu to switch between the log file and command file windows.

When opened in the command file window the following code appears.

```
.INSERT
.POWER_ON

00 A4 04 00 07 A0000000039010 (9000)
```

1. Press F5 to process command file.

The log file display window opens showing the following code.

```
Command File : C:\javacard\Course\GalactIC\PCOM32-
RI\chkloy.cmd
Logging File : C:\javacard\Course\GalactIC\PCOM32-
RI\chkloy.L97
Date          : 25 August 1998 at 11h28 18s
Version       : PCOM32 Version 3.24
```

```
0001 :
0002 : .INSERT
```

2. Press Enter.

Window asks you to insert card in reader.

3. Insert card.

PCOM32 continues processing.

```
0001 :
0002 : .INSERT
0003 : .POWER_ON
```

4. Press Enter to step through commands.

```

0003 : .POWER_ON

          Command      : POWER_ON
          Output Data   : 00 31 80 71 96 64 32 CE 01 00 82
          Status        : 90 00

0004 :
0005 : 00 A4 04 00 07 A0000000039010 (9000)

          Command      : 00 A4 04 00 07
          Input Data    : A0 00 00 00 03 90 10
          Status        : 90 00

0006 :
0007 :
0008 :

```

```

*****
* FILE PROCESSING RESULT : *
*                               *
*      NORMAL EXECUTION      *
*                               *
*****

```

PCOM32 processes the command file and gives the processing result.

Erasing card data

Reuse.cmd

To add a new set of applications onto the card and discard all the currently loaded applications, you must erase the applications on the card. To do so, launch the Debug Tool in the Tools folder.

1. Select *Open* under the *File* menu. The browser appears. The *Reuse.cmd* file is in the same directory as Debug Tool. The Reuse commands lets you erase data from the card.
2. Open the *Reuse.cmd* file. The following screen appears:

```

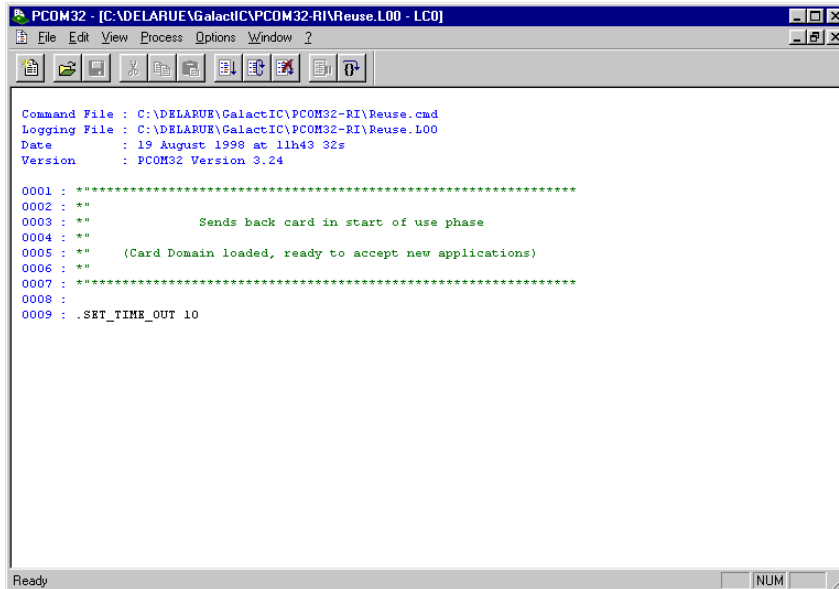
PCOM32 - [Reuse.cmd]
File Edit View Process Options Window ?
*****
**
**      Sends back card in start of use phase
**
**      (Card Domain loaded, ready to accept new applications)
**
*****
.SET_TIME_OUT 10
.INSERT
.POWER_ON
32 3B 73 96 00
Ready NUM

```

3. To launch the Reuse command, click on the Go icon twice.



Clicking on this icon runs the program and brings up this screen:

A screenshot of a Windows command window titled "PCOM32 - [C:\DELLARUE\GalactIC\PCOM32-RI\Reuse.L00 - LC0]". The window shows the following text:

```
Command File : C:\DELLARUE\GalactIC\PCOM32-RI\Reuse.cmd
Logging File : C:\DELLARUE\GalactIC\PCOM32-RI\Reuse.L00
Date       : 19 August 1998 at 11h43 32s
Version    : PCOM32 Version 3.24

0001 : *****
0002 : **
0003 : **           Sends back card in start of use phase
0004 : **
0005 : **   (Card Domain loaded, ready to accept new applications)
0006 : **
0007 : *****
0008 :
0009 : .SET_TIME_OUT 10
```

The window has a standard Windows menu bar (File, Edit, View, Process, Options, Window) and a toolbar with various icons. The status bar at the bottom shows "Ready" and "NUM".

The card is now ready to accept new applications.

Command Files

Introduction

A command file is made up from following elements:

- Card commands
- Card reader directives
- PCOM32 directives
- PCOM32 external directives
- Comments

The format of the card commands is standard, as defined by ISO 7816-4. The card reader directives control the reader: turning it on or off, or prompting the user to insert or remove the card. The PCOM32 directives include:

- flow control constructs for looping and stepping through command sequences and
- directives to assign and access variables and constants.

The external directives provide a mechanism for passing data to external functions (.DLLs).

Typographic conventions

The following conventions are used in the command syntax descriptions. Reserved words and symbols are marked in bold. Parameter names, which stand for values that the user has to supply, are marked in *italic*. Optional input is delimited with square brackets. Note that square brackets are also used to delimit expected return values, in this context the square brackets are part of the command syntax and are marked accordingly in bold.

Card commands

ISO 7816-4 specifies four command formats:

CLA **INS** **P1** **P2**

CLA **INS** **P1** **P2** **Le**

CLA **INS** **P1** **P2** **Lc** **Input Data**

CLA **INS** **P1** **P2** **Lc** **Input Data** **Le**

CLA	Command class (1 byte)
INS	Command instruction code (1 byte)
P1	First parameter of the command (1 byte)
P2	Second parameter of the command (1 byte)
Lc	Length of the input data (1byte)
Le	Length expected, of data returned by the card (1 byte)
Input Data	Data sent to the smart card (Lc bytes)

The ISO commands appear in the command file unmodified, as strings of hexadecimal digits.

Example

A0 A4 00 00 02 3F 00

Expected return status

The expected return status for any command, or for the .POWER_ON directive, can be appended to the command. If the status returned by the card does not match the expected return status, a STATUS ERROR message is displayed in the logging file. An X anywhere in the expected return status matches any hexadecimal digit in the corresponding position in the actual return status. The expected return status must be on the same line as the command to which it applies.

Syntax

Command (*ExpectedReturnedStatus*)
or
 .POWER_ON (*ExpectedReturnedStatus*)

Example

```
A0 A4 00 00 02 3F 00 (XF XX)
; expected return status:
; StatusWord1: any hex digit followed by F
; StatusWord2: any two hex digits
```

Expected return data

The expected return data for any command, or for the .POWER_ON directive, can be appended to the command. If the data returned by the card does not match the expected return data, a DATA ERROR message is displayed in the logging file, and the offending bytes are flagged with a "<" character. An X anywhere in the expected return data matches any hexadecimal digit in the corresponding position in the actual return data. The expected return data must be on the same line as the command to which it applies.

Syntax

Command [*ExpectedReturnData*]
or
 .POWER_ON [*ExpectedReturnData*]

Example

```
A0 C0 00 00 08 [XX XX XX XX 3F 00 XX XX]
A0 C0 00 00 08 [XX XX XX XX 3X X0 XX XX]
```

Special characters

Comment character

The ";" and "*" characters are defined to start a comment in a line of the command file. A comment may be placed anywhere in a command file, all the characters following the ";" or the "*" up to the end of the line are then considered to be part of the comment.

Example

```
; this is a comment
* this is a comment
this is not a comment
A0 A4 00 00 02 3F 00 ; this is a comment
```

Line continuation character

A line continuation character "\" can be added after the last significant character of a line to indicate that the card command continues on the next line. Any character on the same line as the line continuation character and following it is ignored.

Example

```
A0 DC 01 04 10 FF FF FF FF FF FF FF FF \
                FF FF FF FF FF FF FF FF
```

Indentation

The space and tabulation characters can be used for indentation.

Card reader directives

There are four card reader directives:

- .INSERT
- .EJECT
- .POWER_ON
- .POWER_OFF

.INSERT

If there is no card present in the reader, PCOM32 prompts the reader to insert one.

Syntax

.INSERT

.EJECT

Prompts the user to remove the card from the reader.

Syntax

.EJECT

.POWER_ON

Switches on the card reader.

Syntax

.POWER_ON [/PROTOCOL_ON] [/FClockFrequency]

Parameters and switches

- *ClockFrequency*
Frequency (in MHz) of the card reader clock. The value must be between 4 and 8, the default value is 4.
- PROTOCOL_ON
Enables the return of the protocol bytes by the card at power on.

Example

.POWER_ON /PROTOCOL_ON /F8

.POWER_ON /F4

.POWER_OFF

Switches off the card reader.

Syntax

.POWER_OFF

Setup directives

The following directives are used to initialise PCOM32:

- `.LIST_ON`
- `.LIST_OFF`
- `.STEP_ON`
- `.STEP_OFF`
- `.ERROR_BEEP_ON`
- `.ERROR_BEEP_OFF`
- `.SET_TIME_OUT`
- `.READER`

.LIST_ON

Enables logging. The output from the logging process appears in a separate file.

.LIST_OFF

Disables logging. However, logging is automatically re-enabled when an error is detected.

.STEP_ON

Enables step by step execution. This is the default setting.

.STEP_OFF

Disables step by step execution.

.ERROR_BEEP_ON

Enables beep on error.

.ERROR_BEEP_OFF

Disables beep on error.

.END_BEEP_ON

Enables beep on end of file.

.END_BEEP_OFF

Disables beep on end of file.

.SET_TIME_OUT

Sets the time-out for the transmit command. The directive can be used to prevent premature time-outs.

Syntax

```
.SET_TIME_OUT TimeOut
```

Parameters

- *TimeOut*
Value (in seconds) of the time-out delay. The default value is 1 second, the maximum value is 250 seconds.

Example

```
.SET_TIME_OUT 10
```

.READER

Changes the default reader assignment. The name and characteristics of the card reader can be configured with the windows Control Panel.

Syntax

```
.READER Name
```

Parameters

- *Name*
Name of the selected reader. The length of the name is limited to 3 characters.

Example

```
.READER TE1
```

File management directives

Two directives are used to manage the command files:

- .CALL
- .EXECUTE

.CALL

Calls and executes a secondary command file.

Syntax

```
.CALL FileName [/LIST_ON] [/LIST_OFF] [/STEP_ON] [/STEP_OFF]
```

Parameters and switches

- *FileName*
Name of the called command file, the name may include the full path to reach the file.
- LIST_ON
Enables logging for the called command file.
- LIST_OFF
Disables logging for the called command file.
- STEP_ON
Enables the step by step execution of the called command file.
- STEP_OFF
Disables the step by step execution of the called command file.

The default execution mode is the execution mode of the calling command file. When the last instruction of the called file has been executed, control returns to the command immediately following the CALL directive. CALL directives can be nested.

Example

```
; Responses to commands in INI_LOCK.CMD not logged
.CALL C:\A_TEST\C\COMMUNS\INI_LOCK.CMD /LIST_OFF
```

.EXECUTE

Calls and executes a secondary command file and logs the responses in a text file.

Syntax

.EXECUTE *FileName*

Parameters

FileName

Name of command file to execute, the name may include the full path to reach the file. The responses to the called file are always logged separately: in the first instance, in *FileName.L00*, and if the same command file is executed a second time, in *FileName.L01*.

Note: all the buffers are reinitialised by the .EXECUTE directive. The DLLs loaded before the .EXECUTE directive are not considered loaded by the executed command file, and the DLLs loaded by the executed command file are unloaded after the executed command file processing.

Example

```
; Execute scenario.cmd and generate SCENARIO.L00  
.EXECUTE SCENARIO.CMD
```

Loop management directives

.BEGIN_LOOP and .LOOP

A basic loop instruction is available in PCOM32 through the `.BEGIN_LOOP` and `.LOOP` directives. A legal loop instruction must begin with a `.BEGIN_LOOP` and end with a `.LOOP`. It is possible to nest up to 16 loops.

Syntax

```
.LOOP LoopCount [/HEX]
```

Parameters and switches

- *LoopCount*
Total number of iterations: must be greater than 1 and less than 2^{32} .
- `/HEX`
Interpret the `LoopCount` parameter as a hexadecimal number.

Example

```
.BEGIN_LOOP
  .BEGIN_LOOP
    A0 A4 00 00 02 3F 00
  .LOOP 10 /HEX
.LOOP 2

; the inner loop is executed 32 times
```

Buffer management directives

The PCOM32 buffers

PCOM32 provides five general purpose buffers and two buffers for storing card responses. The general purpose buffers are accessible to directives for both reading and writing. The two buffers that store card responses are read only. The table below summarises the buffer properties.

PCOM32 buffers

Name	Function	Size (bytes)	Accessibility
G	general purpose	256	read/write
I	general purpose	256	read/write
J	general purpose	256	read/write
K	general purpose	256	read/write
M	general purpose	256	read/write
R	stores data returned by the card	256	read only
W	stores the status returned by the card	2	read only

Accessing data from buffers

Data from any of the buffers can be inserted after either a command or a directive. If the buffer name is used without a range specification, the entire contents of the buffer are inserted. Using range specifications, it is possible to insert substrings from a buffer.

Syntax

Command *BufferName*[(*begin:end*)]

or

Command *BufferName*[(*begin;length*)]

or

Directive *BufferName*[(*begin:end*)]

or

Directive *BufferName*[(*begin;length*)]

Parameters

- *BufferName*
One of the buffers: G, I, J, K, L, M, R or W. If the *BufferId* is not followed by a range, the entire content of the buffer is inserted. If the *BufferId* is followed by a range, then the specified range is inserted.
- *begin:end*
The range to be inserted: 2:6 means bytes 2 to 6 inclusive, and 6:6 means just the sixth byte. Note that the first element in the buffer is element 1.
- *begin;end*
The range to be inserted: 2;6 means insert 6 bytes starting at byte 2, and 6;2 means insert 2 bytes starting at byte 6. Note that the first element in the buffer is element 1.

Example

A0 A4 00 00 02 R(2:2) R(1:1)

A0 A4 00 00 02 R(2;1) R(1;1)

The directives

The following directives are used to initialise or modify the buffers:

- .SET_BUFFER
- .INCREASE_BUFFER
- .DECREASE_BUFFER
- .APPEND_BUFFER

.SET_BUFFER

- Sets or modifies the contents of any of the read/write buffers.

Syntax

```
.SET_BUFFER BufferName[ (begin:end) ] HexData
```

```
.SET_BUFFER BufferName[ (begin;end) ] HexData
```

Parameters

- *BufferName*
If no range is specified, the first data byte is assigned to the first element of the buffer, and subsequent bytes to subsequent elements, in left to right order. This is the only way to initialise a buffer.
- *begin:end*
The range of buffer elements to be assigned. G(3;7), for example, means elements: 3 to 7 inclusive. The range specified must already contain data.
- *begin;end*
The range of buffer elements to be assigned. G(3;7), for example, means 7 elements: starting at element 3 and ending at element 9. The range specified must already contain data.
- *HexData*
If you are assigning to a range of elements, the number of elements must be the same on both sides of the assignment.

Example

```
.SET_BUFFER I 11 22 33 44 55 66 77 88
; I = 11 22 33 44 55 66 77 88
```

```
.SET_BUFFER I(5;4) I(1;4)
; I = 11 22 33 44 11 22 33 44
```

```
.SET_BUFFER I(1;4) I(8;1) I(7;1) I(6;1) I(5;1)
; I = 44 33 22 11 11 22 33 44
```

.INCREASE_BUFFER

Increments one or more buffer elements. If a range is not specified the increment is added to the last element of the buffer, and any overflow is carried over to the left. So a 256-byte buffer used in this way provides a counter with a maximum value of $2^{2048}-1$, which should be enough for most purposes! Of more practical value, is the possibility of specifying a number of bytes for a counter somewhere within the buffer. For example:

```
.INCREASE_BUFFER G(10:11) 01
```

could be used to provide a modulo 2^{16} counter, using bytes 10 and 11.

Syntax

```
.INCREASE_BUFFER BufferName[(begin:end)] Increment
```

```
.INCREASE_BUFFER BufferName[(begin;end)] Increment
```

Parameters

- *BufferName*
Name of buffer to which the increment is to be applied.
- *begin:end begin;end*
The range of the buffer to which the increment is to be applied.
- *Increment*
The value of the increment. The value can be provided from the same or another buffer. The number of bytes in the value must be less than or equal to the number of bytes specified in *BufferName*[(*begin:end*)]. Note that two characters are required to express the value of a byte, so an increment of one is written 01 not 1.

Examples

```
; I = 00 00 00
```

```
.INCREASE_BUFFER I 01
```

```
; I = 00 00 01
```

```
.INCREASE_BUFFER I FF
```

```
; I = 00 01 00
```

```
.INCREASE_BUFFER I(2:2) FF
```

```
; I = 00 00 00
```

```
; note, that the overflow is not carried to the left
```


.DECREASE_BUFFER

Decrements one or more buffer elements. If a range is not specified the decrement is subtracted from the last element of the buffer, and any borrow that may be required is taken from the left.

Syntax

```
.DECREASE_BUFFER BufferName[(begin:end)] Increment
```

```
.DECREASE_BUFFER BufferName[(begin;end)] Increment
```

Parameters

- *BufferName*
Name of buffer to which the decrement is to be applied.
- *begin:end begin;end*
The range of the buffer to which the decrement is to be applied.
- *Increment*
The value of the decrement. The value can be provided from the same or another buffer. The number of bytes in the decrement must be less than or equal to the number specified in *BufferName*[(*begin:end*)]. Note that two characters are required to express the value of a byte, so a decrement of one is written 01 not 1.

Example

```
i I = FF FF FF

.DECREASE_BUFFER I 01
i I = FF FF FE

.DECREASE_BUFFER I FF
i I = FF FE FF

.DECREASE_BUFFER I(2;1) FF
i I = FF FF FF

.SET_BUFFER I = 00 00 00

.DECREASE_BUFFER I 01
i I = FF FF FF
```

.APPEND_BUFFER

Concatenates two or more byte sequences and assigns the result to the first named buffer. The byte sequences can be provided directly as data, or from one or more buffers. Note that the total length of the string assigned is the total length of the strings specified, so that a buffer can be truncated by an `.APPEND_BUFFER` directive. For example:

```
.SET_BUFFER G 00 00 00 00 00 00 00 00  
.APPEND_BUFFER G(3:3) 11
```

G(3:3) is one byte: 00, it is concatenated with the 11, and the result is assigned to the buffer G, so the buffer now contains just two bytes: 00 11.

Syntax

```
.APPEND_BUFFER BufferName [(begin:end)] HexData
```

```
.APPEND_BUFFER BufferName [(begin;end)] HexData
```

Parameters

- *BufferName*
The name of the buffer to which the concatenated string is assigned. If the name is not qualified with a range, the whole buffer is used as the first part of the string to be concatenated. Otherwise just the characters specified are used.
- *begin:end* or *begin:length*
The range from the named buffer.
- *HexData*
Any sequence of hexadecimal data. The data may derive from one or more buffers, including the buffer being assigned to, or it may be supplied directly.

Example

```
; I = FF FF FF  
; J = 22 33  
  
.APPEND_BUFFER I 01  
; I = FF FF FF 01  
  
.APPEND_BUFFER I 00 J  
; I = FF FF FF 01 00 22 33  
  
.APPEND_BUFFER I(4;1) 44  
; I = 01 44
```

Constant management directives

PCOM32 provides a constant definition mechanism which works along the same lines as the `#define` in C.

.DEFINE

Defines a constant value.

Syntax

```
.DEFINE %ConstantName ConstantValue
```

Parameters

- *ConstantName*
The constant name. It must be preceded by a % character, and must be less than 32 characters.
- *ConstantValue*
The replacement string. Everything following the *ConstantName* up to the end of the line is part of the replacement string.

Example

```
.DEFINE %SELECT A0 A4 00 00 02
```

.UNDEFINE

Undefineds a constant name. The name can then be reused in a further `.DEFINE` directive.

Syntax

```
.UNDEFINE %ConstantName
```

Parameters

- *ConstantName*
Name of the constant to be undefined.

Examples

```
.DEFINE %SELECT A0 A4 00 00 02
.DEFINE %MF 3F 00

%SELECT %MF

.DEFINE %BUFFER_INIT 00 00 00 00

.SET_BUFFER I %BUFFER_INIT
```


APPENDIX A – ERROR STATUS

Reader error status

value (hex)	meaning
60 C0	Function unknown
60 C1	Illegal function parameters
60 C2	Illegal format of the function (bytes)
60 C3	The length byte and the number of data bytes following do not match, or a read function consists of more than five bytes
60 FE	Data I/O line held at 0 volt or POWER_CARD not executed
60 80	Operation correctly terminated
60 81	No card
60 82	Card in slot
60 83	Card in position, but too short
60 84	Card in position
60 85	Card pulled out and then replaced in correct position
60 86	Motor failure in CAD
60 87	Card still in correct position
60 F0	Card answer error (INS not correct) or erroneous TS byte during POWER_CARD
60 F1	3 parity errors in TS byte reception
60 F2	Card cannot be processed
60 F3	Card protocol not supported (T0 <> 0 in TD1 byte during POWER_CARD)
60 F4	Framing error in reception mode
60 F8	3 parity errors in reception mode
60 FC	3 parity errors in transmission mode
60 FD	Failure in programming voltage generator or possible short circuit (Vpp <4 volts)
90 FF	Card not in place, or does not respond (mute)

Card error status

Value (Hex)	Meaning
61LL	Correct execution, response is LL bytes
6300	EXTERNAL AUTHENTICATE failed
6490	Data not found. AID not written during personalisation
6491	EEPROM integrity error Incorrect optional code checksum BAD DES key Data checksum error
6581	EEPROM write error
6600	Java unhandled Exception
6601	Java unhandled Throwable
6602	Java unhandled ArithmeticException
6603	Java unhandled ArrayIndexOutOfBoundsException
6604	Java unhandled ArrayStoreException
6605	Java unhandled ClassCastException
6606	Java unhandled IndexOutOfBoundsException
6607	Java unhandled NegativeArraySizeException
6608	Java unhandled NullPointerException
6609	Java unhandled RuntimeException
660A	Java unhandled SecurityException
660B	Java unhandled UserException
660C	Java unhandled APDU Exception
660D	Java unhandled PINException
660E	Java unhandled SystemException
660F	Java unhandled TransactionException
6610	Java unhandled CryptoException
6640	JavaStackOverflowError
6641	JavaOutOfMemoryError
6680	DLRJC VM wrong package file version
6681	DLRJC VM unsupported bytecode
6682	DLRJC VM feature not supported yet
6683	DLRJC VM wrong number of parameters
6684	DLRJC VM wrong data type
6685	DLRJC VM class definition not found
6686	DLRJC VM method not found
6687	DLRJC VM native API entry not found
6688	DLRJC VM field not found
6689	DLRJC VM no access to field
668A	DLRJC VM no access to method
6690	DLRJC VM stack underflow
6691	DLRJC VM stack not empty on method return
6692	DLRJC VM invalid local variable access

Value (Hex)	Meaning
6700	Length error
6982	ACs not satisfied
6985	Sequence error
6990	Command forbidden in current life phase
6991	No random generated before
6A80	Incorrect parameter in data field
6A81	Command not supported
6A84	Not enough space
6A85	Lc inconsistent with length recorded in TLV object
6A86	Incorrect P1, P2
6A87	P1, P2 inconsistent with Lc