JCOP Tools Technical Brief

***Overview**: This document contains a simple overview about the functionality and benefits of the JCOP Tools development environment. Requests for further information may be directed at javacard@zurich.ibm.com .*

Basic specifications

The JCOP Tools provide a set of development tools for the successful development, deployment and testing of applications for any generic OpenPlatform JavaCard, with specific support for the IBM JCOP platform. JCOP itself is the IBM BlueZ implementation of the basic specifications [1] and [2] including refinements from Visa International set in the Visa OpenPlatform Card Implementation Guides (http://www.visa.com/nt/suppliers/vendor). Applications for a generic OpenPlatform compliant JavaCard can be fully and conveniently developed using the JCOP Tools development kit. It provides both a set of command line tools as well as a fully integrated, graphical development environment, the IDE, allowing for all the individual development steps to be performed in a single application. The IDE allows for the creation, modification and management of project data and application source code, simplifies error detection and trouble shooting during the compilation and conversion process, and offers a powerful testing and debugging environment for JCOP applications. The JCOP tools feature

(1) JCOP simulation programs which are executed on the development host, but behave very similar to physical JCOPs and cards

(2) Debugger, allowing for the debugging of JCOP applications at the source code level,

(3) Powerful shell for either issuing interactive commands or executing long-running batch scripts to auto-test the application.

Thus, the JCOP tools enable a flexible and time-efficient development process on the fast and convenient development host, while still allowing for the final deployment and testing of JCOP applications on real smart cards compliant with [1] and [2]. With the support of PC/SC and contactless readers, the JCOP Tools can communicate with a wide range of available card readers and terminals.

Additionally, JCOP Tools do not only support the Microsoft Windows operating system family, but can also be deployed on Linux systems or even Mac OS X computers.

2 Requirements

2.1 Supported host operating systems

Microsoft Windows

RedHat Linux 7.x or similar systems

Mac OS X

2.2 Java Runtime

The JCOP Tools are based on the Java Development Kit 1.3.x or similar. The IBM Java Development Kit 1.3 is included in the JCOP Tools shipped on CD.

2.3 Drivers

The JCOP Tools do not come with a smart card reader, but support the usual smart card reader standards, such as PC/SC for Windows or Linux. PC/SC is fully installed for Windows 2000 and XP, hence adding a PC/SC compliant reader to the PC will activate the physical reader interface of the JCOP Tools for interaction with real cards. For Linux, the optional Muscle PC/SC driver system [3] must be installed prior to interaction with real cards.

3 Component Overview

3.1 Highlights

Portability: The JCOP Tools are available for Windows, Linux and Mac OS X and thus provide a portable development platform for JCOP applications.

Completeness: The JCOP Tools assist in all necessary development steps of a JavaCard/OpenPlatform application (applet). Project management, source code editor, compilation and build tools, source-level debugger, simulation environment, APDU shell and scripting environment are all integral parts of the JCOP Tools.

Development Views: The JCOP Tools on the one hand offer an Integrated Development Environment (IDE) managing all development steps in a single application. On the other hand, they also allow for the execution of the individual tools from the command line. This allows for a better integration into other development environments and for more flexibility in the case of advanced projects.

Full JCOP support: The JCOP Tools fully support all members of the JCOP family. The different JCOPs can either be simulated in detail, or can be integrated via the PC/SC interface in case of contact-based JCOPs or via the Philips Mifare interfaces in case of contact-less JCOPs.

Exported APIs: The JCOP Tools are shipped with an off-card API allowing for the development of off-card, i.e., terminal applications. The API which is used by the JCOP Tools themselves allows for both extending the JCOP Tools as well as the development of stand-alone applications.

3.2 Components

3.2.1 Overview

The IDE integrates most of the individual JCOP Tools in a single application with a modern graphical user interface, and thus is operated like state-of-the-art integrated development environments.

3.2.2 Project-Oriented Development

The IDE enforces project-oriented development. An IDE project encapsulates all application source files, test scripts, compilation options and target settings like AIDs, version information, etc. Both project- and IDE-related settings can be easily and conveniently modified in typical graphical dialogs. Users of other IDEs feel quickly familiar with the JCOP Tools.

3.2.3 Source Code Editor

The IDE provides a state-of-the-art editor component which supports both the editing of JCOP applet sources as well as shell scripts. Find, Find/Replace, syntax-highlighting, source-code formatter, and keyboard shortcuts are standard features.

3.2.4 Build Tools

The IDE uses the Java compiler shipped with all modern Java development kits for the class file generation and exploits an integrated converter for the generation of applet cap files and export files. Within the IDE, a single click on an error message after a project build sets the cursor at the offending line. The converter can also be driven from the command line allowing for flexible deployment in different development environments.

3.2.5 Simulation Environment

The JCOP Tools are shipped with simulations for all important members of the JCOP family, currently JCOP10, JCOP20 and JCOP30. These simulations are executed on the development host, but behave almost exactly the same as real, physical JCOP cards. JCOP applet development can then fully proceed in the simulation environment. Especially, the simulations can be driven at much higher execution rates allowing for faster development processes. The simulations still offers the same limited RAM, EEPROM and ROM sizes and features as the physical JCOPs. Additionally, the simulations can be executed in a speed-emulation mode where the limited execution speeds of the real chips are simulated.

3.2.6 Source level debugging

Applets can be downloaded, installed and executed on the JCOP simulation with the ability to watch and trace the progress of an application in detail. The JCOP Tools therefore offer BugZ, the JCOP application debugger, whose execution is either triggered by the IDE, but can also be executed stand-alone from the command line. BugZ offers the many features typically expected from a modern source-level debugger, for instance class browser, breakpoints, line stepping, local variable watch, object inspection etc. Additionally, BugZ reports about the memory and/or transaction buffer usage giving the developer important details about the application and how to optimize it.

3.2.7 Shell environment

Smartcards react on the requests sent by a terminal and/or a card reader. Thus, the flexible definition and handling of APDU traffic is central for a smartcard application development environment. The JCOP Tools offer a programmable and extensible shell which can be used both interactively as well as in batch mode. The latter allows to run complex shell scripts for instance to test your JCOP application against complex test suites or execute complex JCOP personalization schemes. The interactive mode is typically used during application development or card management. The Shell does not only allow sending and receiving APDUs, but offers a large set of commands extremely simplifying the handling of an OpenPlatform card. For instance, authentication to the card, package download, applet instantiation and deletion are all a matter of simple, easy-to-learn commands. Additionally, the shell can be extended by plug-ins which can implement any kind of complex off-card command for a specific applet. The JCOP Tools are shipped with powerful plug-ins for OpenPlatform applets, security domains etc. Further plug-ins, for instance for PKCS#15, can be made available on demand.

3.2.8 Off-card, Terminal APIs

The shell and shell plug-ins all make use or are even part of the base APIs which are shipped with the JCOP Tools and which allow for the convenient development of powerful off-card and terminal applications. Especially, all OpenPlatform-related services, including

the necessary cryptographic computations, are encapsulated in simple-to-use, but powerful APIs.

For instance, a terminal application having to download, install and communicate with a JCOP applet is simple to implement. Advanced terminal applications can make use of the plug-in mechanism to either extend the functionality offered in the base API or make use of advanced plug-ins like PKCS#15 to handle complex JCOP applications.

The JCOP Tools APIs are fully supported on all host platforms, including Windows, Linux and Mac OS X.

3.2.9 PCSC/Reader/Hardware Support

The JCOP Tools API supports all readers/terminals obeying the PC/SC standard on the three supported platforms, Linux, Mac OS X and Windows. Additionally, the JCOP Tools APII seamlessly supports the contact-less Mifare readers manufactured by Philips Semiconductors. On demand, the JCOP Tools API can also drive a number of readers/terminals adhering to the CT-API standard. The JCOP Tools thus support a large variety of readers/terminals and their protocols (ISO 7816 T=0, T=1; ISO14443 T=CL).

3.2.10 Command Line Tools

The different JCOP Tools do not always have to be operated from within the IDE. The shell, debugger BugZ, converter and off-card APIs can be used totally independent from it. This allows for using the tools in other development environments, for instance in UNIX-like environments. Additionally, the JCOP Tools are shipped with a number of tools which are expected to be used from the command-line. For instance, the CardMan application represents a powerful, easy-to-use command line application for executing all simple, and typical JCOP management tasks in a quick manner.

4 Feature List

4.1 IDE

Project-View: projects encapsulate application and script sources, build settings, applet and package properties and preferences. All settings and preferences can be easily set and modified intuitively in graphical dialogs.

Built-in-Editor: supports find, find/replace, syntax-highlighting, cut/copy/paste, pretty-print (source code formatter), undo/redo.

IDE-Preferences: the IDE is configurable regarding fonts, colors and keyboard-settings.

Build-Process: dependency check, source compilation and class conversion in a single step, error/message window, jump-to-error on single click, information dialog about code size/code dependencies/component sizes.

Package-/Key-Management: JCOP Tools offer graphical dialogs to define OpenPlatform key-sets and the AIDs to use for packages, applets and instance at download and installation time. This package and key information can then be used transparently from the Shell.

Test-Modes: the IDE can be configured to restart or reuse a running simulation, debugger and shell automatically after every applet build. This allows for flexible and fast edit/build/debug-cycles.

4.2 Simulation Environment

The JCOP Tools ship with different simulations for the different JCOP families, JCOP10, JCOP20 and JCOP30.

Strong simulation of JCOP versions: extremely similar behavior to the physical JCOPs, for instance the amount of available RAM, EEPROM and ROM is exactly reproduced. Optional speed-emulation allows for simulating the real execution speed of JCOPs, the default, faster execution mode speeds up development and testing on the host PC.

Transparent communication: all JCOP Tools being able to talk to a real card, can also communicate with a simulation. Also, all applications being based on the JCOP Tools API can transparently communicate with a simulation process.

4.3 Debugger BugZ

- Allows for the debugging of JCOP applications at the source level.
- Integrated in IDE and/or executable from the command line.
- Features class browser and application source browser.
- Breakpoints, single-step-execution, stack-trace window, local variables window, object inspector, memory statistics.

4.4 Shell

The JCShell presents the environment to send/receive APDUs from cards/simulations; the target can be any physical JCOP/smartcard and/or simulation.

4.4.1 Operation Modes

Interactive: commands can be sent interactively to cards/simulations during the development or management of a card.

Batch: the shell allows the execution complex scripts, for instance test suites or personalization/initialization scripts.

Script language: the script language supports variables, variable substitution, command help, echo, APDU tracing.

4.4.2 Plugins

The shell offers the possibility to register plug-ins for different applets which may provide advanced, specialized commands for this applet.

Built-in plugins: the shell is shipped with a number of powerful plug-ins, especially for OpenPlatform applets, security domain and/or card manager. Key-management, package download, applet deletion, authentication, secure messaging, pin handling is simple to achieve by few script lines.

4.4.3 Graphical user-interface:

The basic development-related shell functions can be executed by single mouse clicks, for instance specification of AIDs, download of packages, applet instantiation, key-set configuration etc.

4.5 Off-card APIs

These APIs are an integral part of the JCOP Tools, but accessible and open to any kind of terminal application.

These offer all basic and many advanced functions offered by the JCOP Tools, for instance OpenPlatform management functions, secure messaging etc. Documentation and Samples for their use is included in the distribution.

Three main packages:

- com.ibm.jc offering basic functionality like opening a card connection and sending an APDU
- com.ibm.jc.tools contains the various, powerful plug-ins for advanced functions like OpenPlatform management
- com.ibm.jc.terminal to access all the different physical terminals supported and/or the many virtual terminals (for debugging, connections via the Internet).

The APIs are portable across Windows, Linux and Mac OS X.

4.6 Communication

The JCOP Tools support PC/SC reader/terminals on Windows, Mac OS X and Linux. Various readers adhering to the CT-API standard can be integrated on demand as well.

Seamless support for the contact-less Mifare readers by Philips is given. Both IDE and Shell detect the presence of the Philips tools and then automatically offer the access to contact-less interfaces.

JCOP as well as JCOP Tools are able to communicate over T=0, T=1 and/or T=CL.

4.7 Command Line Tools

Converter, Shell, and BugZ can be integrated into and used in command line environments.

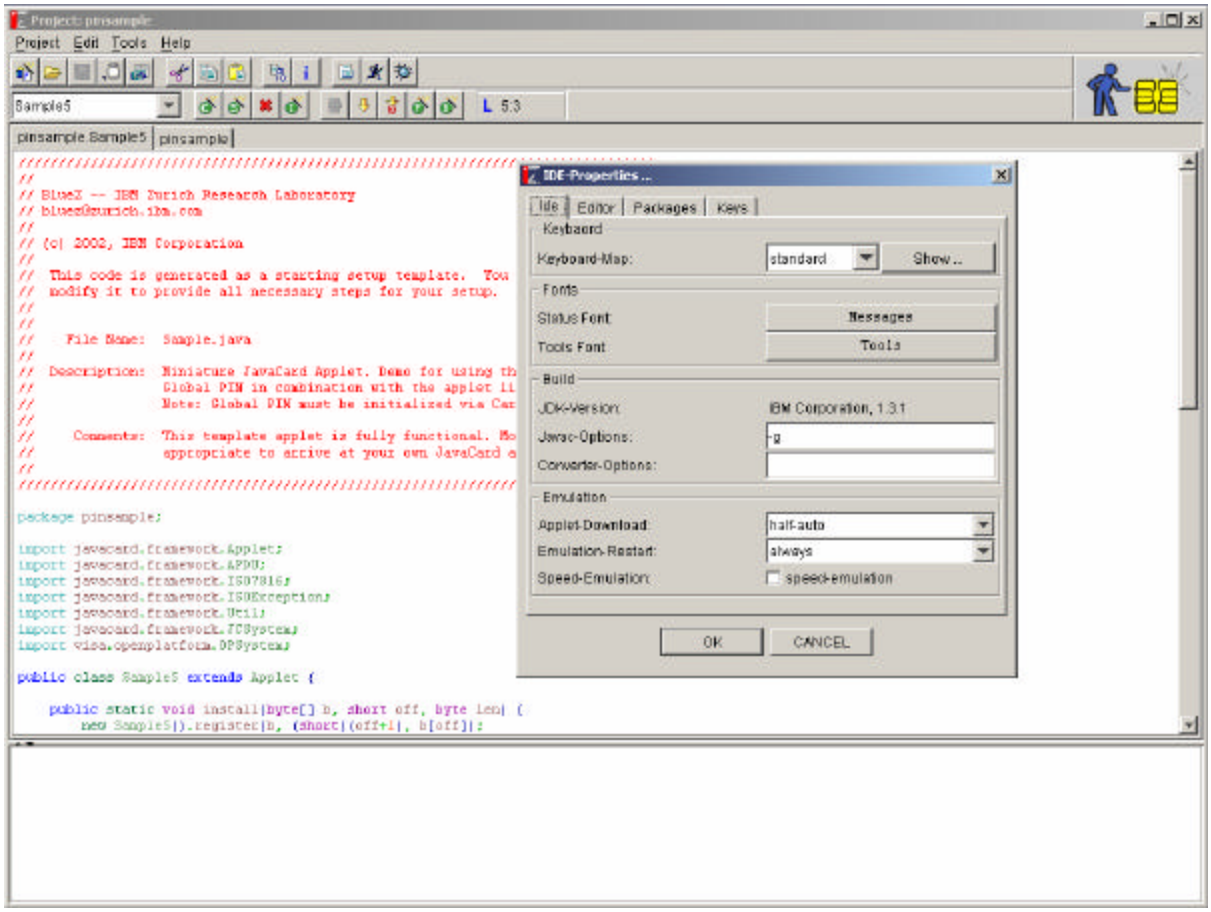
The JCOP Tools also ship with tools to access certain functionality on Mifare chips. This software is provided by Philips Semiconductors.

Also included in the distribution is CardMan, a simple, standalone command line application allowing for the hassle-free management of a OpenPlatform/JavaCard in a simple manner.

5 Screen Shots

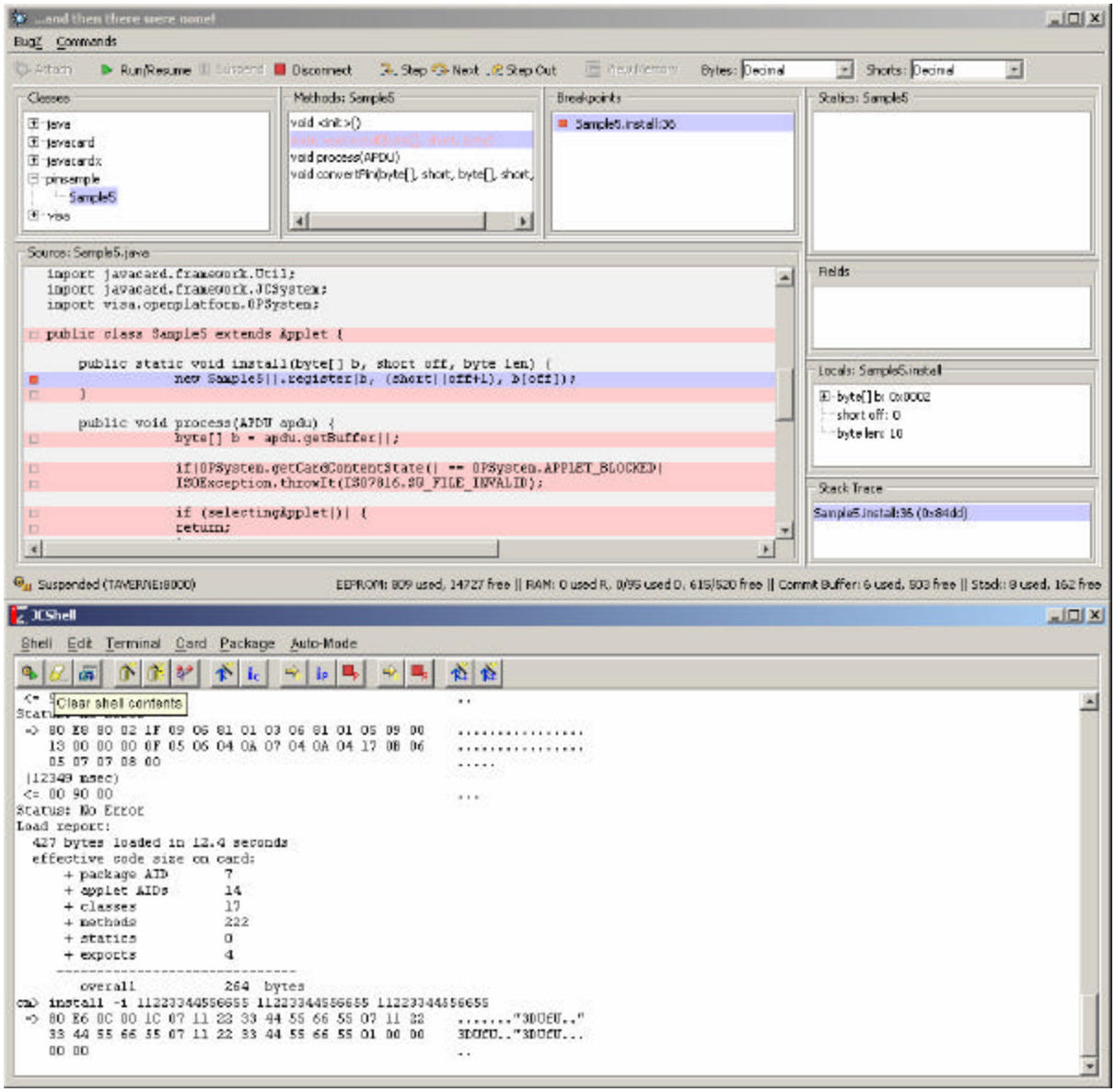
5.1 IDE

A typical view of the Integrated Development environment is shown below. Active is one of the sample projects shipped with the JCOP Tools.



5.2 BugZ

For the same project as above, the screen shot below shows the graphical source level debugger for applets active. Also shown is the JCShell driving a test run


...and then there were none!
BugZ Commands
Run/Resume
Disconnect
Step
Next
Step Out
Bytes: Decimal
Shorts: Decimal
Classes
java
javacard
javacardx
pinsample
Sample5
visa
Methods: Sample5
void <init>()
void process(APDU)
void convertPin(byte[], short, byte[], short,
Breakpoints
Sample5.install:36
Statics: Sample5
Source: Sample5.java
import javacard.framework.Util;
import javacard.framework.JCSystem;
import visa.openplatform.OPSystem;
public class Sample5 extends Applet {
public static void install(byte[] b, short off, byte len) {
new Sample5().register(b, (short)(off+1), b[off]);
}
public void process(APDU apdu) {
byte[] b = apdu.getBuffer();
if(OPSystem.getCardContentState() == OPSystem.APPLET_BLOCKED)
ISOException.throwIt(ISO7816.SW_FILE_INVALID);
if (selectingApplet()) {
return;
Fields
Locals: Sample5.install
byte[] b: 0x0002
short off: 0
byte len: 10
Stack Trace
Sample5.install:36 (0x84dd)
Suspended (TAVERNE:8000)
EEPROM: 809 used, 14727 free || RAM: 0 used R, 0/95 used O, 615/520 free || Commit Buffer: 6 used, 503 free || Stack: 8 used, 162 free
JCShell
Shell Edit Terminal Card Package Auto-Mode
Clear shell contents
=> 80 E6 80 02 1F 09 06 61 01 03 06 61 01 05 09 00 .....a.....a....
13 00 00 00 0F 05 06 04 0A 07 04 0A 04 17 0B 06 ....
05 07 07 08 00 .....
(12349 msec)
<= 00 90 00 ...
Status: No Error
Load report:
427 bytes loaded in 12.4 seconds
effective code size on card:
+ package AID 7
+ applet AIDs 14
+ classes 17
+ methods 222
+ statics 0
+ exports 4
overall 264 bytes
=> 80 E6 0C 00 1C 07 11 22 33 44 55 66 55 07 11 22 ...."3DUfU.."
33 44 55 66 55 07 11 22 33 44 55 66 55 01 00 00 3DUfU.."3DUfU...
00 00 ..

A Revision History

1.0 Initial Version

B References

[1] Sun Microsystems: JavaCard 2.1.1 http://java.sun.com/products/javacard

[2] Global Platform Consortium: OpenPlatform 2.0.1' http://www.globalplatform.org/

[3] Muscle PC/SC for Linux http://www.linuxnet.com