JCOP Tools 3.0 (Eclipse Plugin)

Technical Brief

Revision 1.0

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`.

1 Basic Specifications

IBM JCOP Tools 3.0 ("JCOP Tools") provide a set of development tools for the successful development, testing, and deployment 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 Requirements [3]. Applications for a generic OpenPlatform compliant JavaCard can be fully and conveniently developed using the JCOP Tools. They provide 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 applet 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 JavaCard 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 contact-less 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
- Linux on 32bit x86 (GTK and Motif)
- Mac OS X

2.2 Java Runtime

The JCOP Tools require a Java 2 Standard Edition (J2SE) Runtime Environment (JRE) Version 1.3.x or later, available from `http://java.sun.com/download/`.

2.3 Eclipse

The JCOP Tools are based on the Eclipse Platform [4]. Eclipse SDK 2.1.x is required, available from `http://www.eclipse.org/downloads/`.

2.4 Drivers

The JCOP Tools do not come with a smart card reader, but do support common smart card reader standards such as PC/SC from Windows, Linux and Mac OS X. PC/SC is included with Windows 2000/XP and Mac OS X; on those systems the JCOP Tools can interact with a PC/SC compliant reader without any additional middle-ware. For Linux, the optional M.U.S.C.L.E. PC/SC driver system [5] must be installed to enable 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 JavaCard 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 are on one hand fully integrated into Eclipse, a platform for highly integrated tools, which allows all development steps to be managed from within a single application. On the other hand, individual tools – such as the converter or the shell – can also be executed from the command line. This allows for a better integration in other existing development environments and more flexibility in advanced projects.

Full JCOP Support: The JCOP Tools fully support all members of the JCOP family. The different JCOP cards can either be simulated in detail, or directly used (e.g. over the PC/SC interface).

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

Fully integrated into Eclipse, the JCOP Tools offer all necessary tools with modern graphical user interfaces in a state-of-the-art integrated development environment (IDE).

3.2.2 Project-Oriented Development

Eclipse enforces project-oriented development. A JCOP Project within Eclipse encapsulates all application source files, test scripts, build options, and target settings such as AIDs, version information, etc. All settings can be easily and conveniently modified in typical graphical dialogs. Users of other IDEs will quickly feel familiar with JCOP Tools.

3.2.3 Source Code Editor

The JCOP Tools use the Eclipse JDT Editor for editing JavaCard source files. This advanced and customizable editor features code completion, syntax highlighting, quick assist, find/replace, auto format, refactoring, and many more.

3.2.4 Build Tools

The JCOP Tools use the incremental Java compiler shipped with Eclipse JDT to create class file and an integrated converter for the generation of applet CAP files and export files, as defined in [1]. Compilation and converter errors are displayed in the Eclipse Tasks view, and the offending places in the source files are marked. 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, JCOP11sim, JCOP20, JCOP21id, JCOP21sim, JCOP30, and JCOP31bio. 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 JCOP cards. 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 offer many features typically expected from a source lever debugger: Breakpoints, line stepping, local variable watch, object inspection, etc. Additionally, information about memory and transaction buffer usage are displayed, giving the developer important details concerning performance and possible optimizations. The debugger can also gather profiling and code coverage data, and display these statistics within Eclipse.

3.2.7 Shell environment

Smart cards react to requests sent by a terminal and/or a card reader. Thus, the flexible definition and handling of APDU traffic is an essential part of a smart card 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 permits running complex shell scripts – for instance to test your JavaCard application against complex test suites, or to 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 which simplify handling an OpenPlatform card extremely. Tasks such as authentication, package download, applet instantiation and deletion are all a matter of simple, easy-to-learn commands. Additionally, the shell can be extended by plugins which can implement any kind of complex off-card command for a specific applet. The JCOP Tools are shipped with powerful plugins for OpenPlatform applets, security domains, etc. Other plugins, for instance for PKCS#15, can be made available on demand.

3.2.8 Off-card, Terminal APIs

The shell and shell plugins 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. A terminal application to download, install and communicate with a JavaCard applet is extremely simple to implement. Advanced terminal applications can make use of the plugin mechanism to either extend the functionality offered in the base API or make use of advanced plugins, such as 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 PC/SC, Reader, and Hardware Support

The JCOP Tools API supports all readers/terminals that comply with the PC/SC standard on the three supported platforms: Linux, Mac OS X and Windows. Additionally, the JCOP Tools API 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 JCOP Tools do not always have to be operated from within Eclipse. Shell, converter and off-card APIs can be used totally independent from it. This allows the tools to be used 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

Projects: Projects encapsulate application and script sources, target settings, applet and package properties and preferences. All settings can easily be set and modified in graphical dialogs.

Source File Editor: Supports code completion, syntax highlighting, quick assist, find/replace, auto format, refactoring, undo/redo, and many more.

Preferences: Eclipse is fully configurable regarding fonts, colors, and keyboard settings.

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

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

4.2 Simulation Environment

The JCOP Tools ship with different simulations for the different JCOP families.

Strong Simulation of JCOP Cards: Extremely similar behavior to the physical JCOP cards, i.e. amount of available RAM, EEPROM and ROM is exactly reproduced. Optional speed-emulation allows for simulating the real execution speed of JCOP cards; the default (faster) execution mode speeds up development and testing on the host PC.

Transparent Communication: All JCOP Tools components that talk to a real card can also communicate with a simulation. Furthermore, all applications based on the JCOP Tools API can transparently communicate with a simulation process as well.

4.3 Debugger

The JCOP Tools debugger allows to debug JavaCard applications at source level.

Standard Debugging Features: Breakpoints, single-step execution, stack trace view, local variables view, object inspector, memory statistics.

Profiling: Code coverage analysis, execution frequency by line or block.

4.4 Shell

The JCShell presents the environment to send/receive APDUs from cards/simulations; the target can be any physical JCOP/smart card 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 plugins 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 Open-Platform 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.

4.5 Off-Card APIs

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

- `com.ibm.jc` offers basic functionality like opening a card connection and sending an APDU.
- `com.ibm.jc.tools` contains the various, powerful plugins for advanced functions like OpenPlatform management.
- `com.ibm.jc.terminal` provides access to 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 readers/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. JCOP cards as well as the JCOP Tools are able to communicate over T=0, T=1 and/or T=CL.

For readers that support this feature, the JCOP Shell will show the exact APDU execution time at microsecond level (μsec).

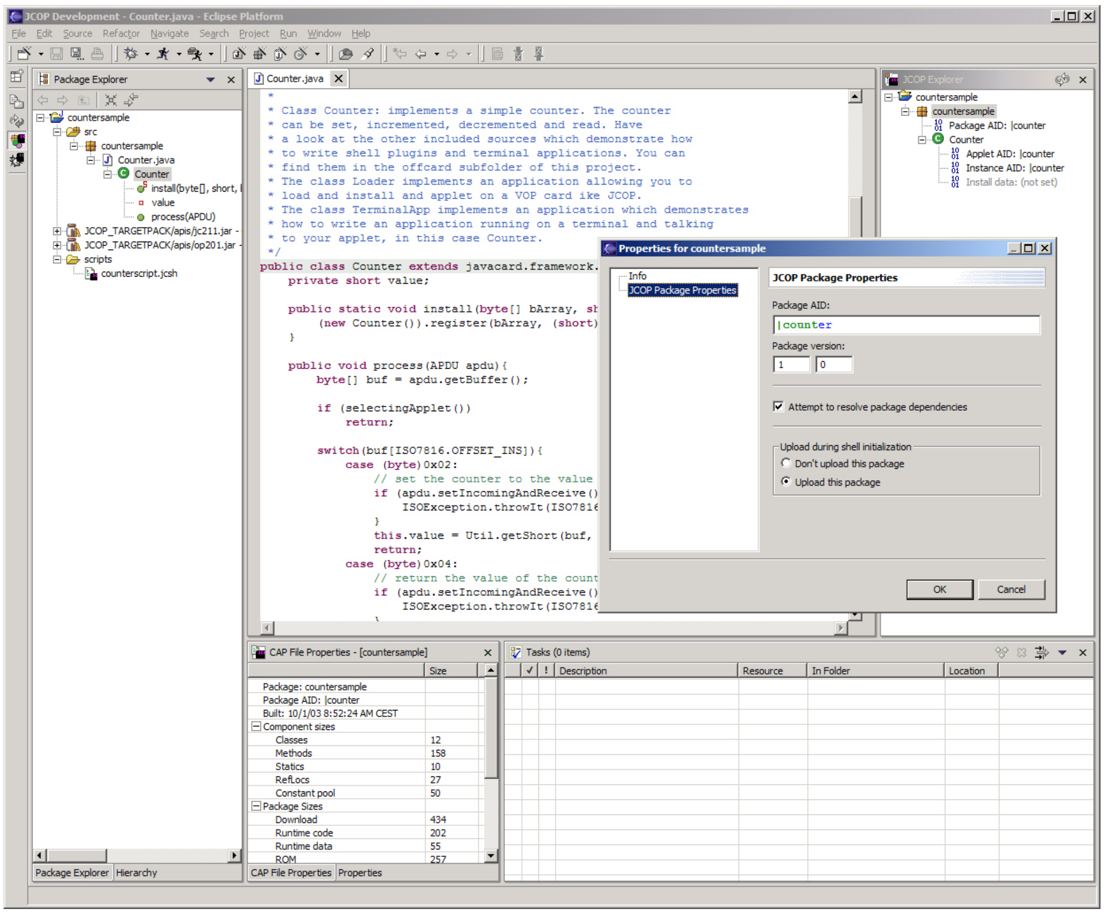
4.7 Command Line Tools

Converter and shell can be integrated into and used in command line environments. Also included in the distribution is CardMan, a simple, standalone command line application allowing for the hassle-free management of any OpenPlatform/JavaCard in a simple manner.

5 Screenshots

5.1 JCOP Development Perspective

A typical view of the Integrated Development Environment is shown below. Active is one of the sample project available for the JCOP Tools.



```
 *
 * Class Counter: implements a simple counter. The counter
 * can be set, incremented, decremented and read. Have
 * a look at the other included sources which demonstrate how
 * to write shell plugins and terminal applications. You can
 * find them in the offcard subfolder of this project.
 * The class Loader implements an application allowing you to
 * load and install and applet on a VOP card ike JCOP.
 * The class TerminalApp implements an application which demonstrates
 * how to write an application running on a terminal and talking
 * to your applet, in this case Counter.
 */
public class Counter extends javacard.framework.
    private short value;

    public static void install(byte[] bArray, sh
        (new Counter()).register(bArray, (short)
    }

    public void process(APDU apdu){
        byte[] buf = apdu.getBuffer();

        if (selectingApplet())
            return;

        switch(buf[ISO7816.OFFSET_INS]){
            case (byte)0x02:
                // set the counter to the value
                if (apdu.setIncomingAndReceive()
                    ISOException.throwIt(ISO7816
                }
                this.value = Util.getShort(buf,
                return;
            case (byte)0x04:
                // return the value of the count
                if (apdu.setIncomingAndReceive()
                    ISOException.throwIt(ISO7816
```

Properties for countersample

Info
JCOP Package Properties

JCOP Package Properties

Package AID: |counter

Package version: 1 0

[✓] Attempt to resolve package dependencies

Upload during shell initialization
Don't upload this package [✓ Upload this package]

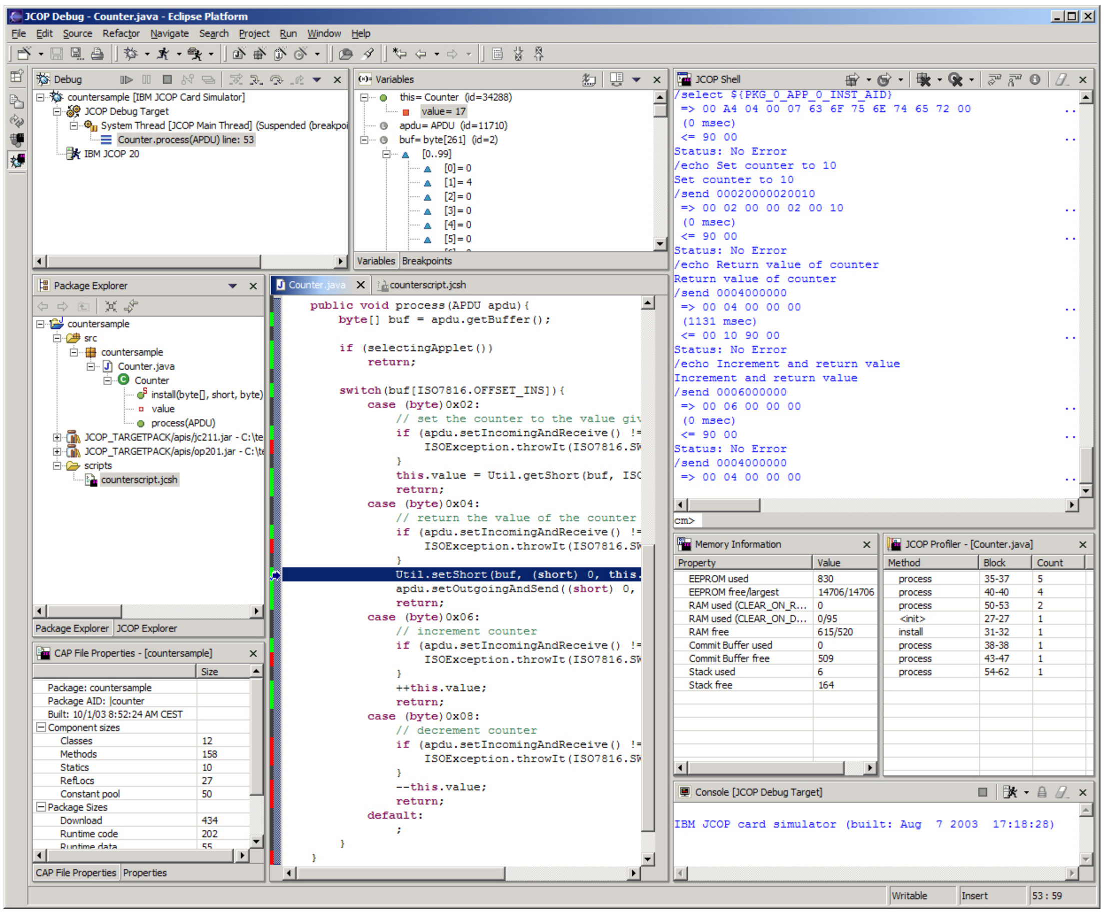
OK Cancel

CAP File Properties - [countersample]

	Size
Package: countersample	
Package AID: |counter	
Built: 10/1/03 8:52:24 AM CEST	
Component sizes	
Classes	12
Methods	158
Statics	10
RefLocs	27
Constant pool	50
Package Sizes	
Download	434
Runtime code	202
Runtime data	55
ROM	257

5.2 JCOP Debug Perspective

For the same sample project, the screenshot below shows the source level debugger for the active applet, which is running a test driven by the shell. Also note the views displaying the resource usage and profiling information.



Debug:

```
countersample [IBM JCOP Card Simulator]
  JCOP Debug Target
    System Thread [JCOP Main Thread] (Suspended (breakpoi
      Counter.process(APDU) line: 53
  IBM JCOP 20
```

Variables:

```
this= Counter  (id=34288)
  value= 17
apdu= APDU  (id=11710)
buf= byte[261]  (id=2)
  [0..99]
    [0]= 0
    [1]= 4
    [2]= 0
    [3]= 0
    [4]= 0
    [5]= 0
```

Package Explorer:

```
countersample
  src
    countersample
      Counter.java
        Counter
          install(byte[], short, byte)
          value
          process(APDU)
  JCOP_TARGETPACK/apis/jc211.jar - C:\te
  JCOP_TARGETPACK/apis/op201.jar - C:\te
  scripts
    counterscript.jcsh
```

CAP File Properties - [countersample]

	Size
Package: countersample	
Package AID: |counter	
Built: 10/1/03 8:52:24 AM CEST	
Component sizes	
Classes	12
Methods	158
Statics	10
RefLocs	27
Constant pool	50
Package Sizes	
Download	434
Runtime code	202
Runtime data	55

Counter.java:

```
    public void process(APDU apdu){
        byte[] buf = apdu.getBuffer();

        if (selectingApplet())
            return;

        switch(buf[ISO7816.OFFSET_INS]){
            case (byte)0x02:
                // set the counter to the value giv
                if (apdu.setIncomingAndReceive() !=
                    ISOException.throwIt(ISO7816.SW
                }
                this.value = Util.getShort(buf, ISC
                return;
            case (byte)0x04:
                // return the value of the counter
                if (apdu.setIncomingAndReceive() !=
                    ISOException.throwIt(ISO7816.SW
                }
                Util.setShort(buf, (short) 0, this.
                apdu.setOutgoingAndSend((short) 0,
                return;
            case (byte)0x06:
                // increment counter
                if (apdu.setIncomingAndReceive() !=
                    ISOException.throwIt(ISO7816.SW
                }
                ++this.value;
                return;
            case (byte)0x08:
                // decrement counter
                if (apdu.setIncomingAndReceive() !=
                    ISOException.throwIt(ISO7816.SW
                }
                --this.value;
                return;
            default:
                ;
        }
    }
```

JCOP Shell:

```
/select ${PKG_0_APP_0_INST_AID}
 => 00 A4 04 00 07 63 6F 75 6E 74 65 72 00          ..
 (0 msec)
 <= 90 00                                           ..
Status: No Error
/echo Set counter to 10
Set counter to 10
/send 00020000020010
 => 00 02 00 00 02 00 10                            ..
 (0 msec)
 <= 90 00                                           ..
Status: No Error
/echo Return value of counter
Return value of counter
/send 0004000000
 => 00 04 00 00 00                                  ..
 (1131 msec)
 <= 00 10 90 00                                     ..
Status: No Error
/echo Increment and return value
Increment and return value
/send 0006000000
 => 00 06 00 00 00                                  ..
 (0 msec)
 <= 90 00                                           ..
Status: No Error
/send 0004000000
 => 00 04 00 00 00                                  ..
cm>
```

Memory Information

Property	Value
EEPROM used	830
EEPROM free/largest	14706/14706
RAM used (CLEAR_ON_R...	0
RAM used (CLEAR_ON_D...	0/95
RAM free	615/520
Commit Buffer used	0
Commit Buffer free	509
Stack used	6
Stack free	164

JCOP Profiler - [Counter.java]

Method	Block	Count
process	35-37	5
process	40-40	4
process	50-53	2
<init>	27-27	1
install	31-32	1
process	38-38	1
process	43-47	1
process	54-62	1

Console [JCOP Debug Target]

```
IBM JCOP card simulator (built: Aug  7 2003  17:18:28)
```

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] `http://www.visa.com/nt/suppliers/vendor/`

[4] Eclipse Platform
`http://www.eclipse.org/`

[5] M.U.S.C.L.E. PC/SC for Linux
`http://www.linuxnet.com/`