

PC/SC DEVELOPMENT TECHNIQUES

A Java applet for smartcard-aware web pages

Headquarters, Europe

SpringCard
13 voie la Cardon
Parc Gutenberg
91120 Palaiseau
FRANCE

Phone : +33 (0) 164 53 20 10
Fax : +33 (0) 164 53 20 18

Americas

SpringCard
6161 El Cajon blvd
Suite B, PMB 437
San Diego, CA 92115
USA

Phone : +1 (713) 261 6746

www.springcard.com

DOCUMENT INFORMATION

Category : Developer's manual
Group : CSB6 Family
Reference : PMD0160
Version : AA
Status : DRAFT

Keywords :
java applet pc/sc scriptable signed javascript

Abstract :
This document aims to help you in programming inside a web page a program to connect to your PC/SC reader and communicate with a card; the graphics part of this program being defined in the HTML page and the communication functions being managed in a Java

[pmd0160-aa] pcsc web page with a java
applet.doc
saved 05/07/10 - printed 05/07/10

REVISION HISTORY

Ver.	Date	Author	Valid. by Tech.	Qual.	Approv. by	Remarks :
AA	24/06/10	ECL		JDA	JDA	Initial release

TABLE OF CONTENT

1.	INTRODUCTION.....	5
1.1.	ABSTRACT.....	5
1.2.	AUDIENCE.....	5
1.3.	PREREQUISITES	5
1.4.	SUPPORT AND UPDATES	5
2.	JAVA SMARTCARD I/O API AT A GLANCE .	6
2.1.	IDENTIFICATION AND DOCUMENTATION	6
2.2.	OVERVIEW OF THE PRINCIPLE CLASSES AND METHODS.....	6
2.3.	A TRIVIAL EXAMPLE.....	7
3.	JAVA SMARTCARD I/O IN AN APPLET	8
3.1.	BACKGROUND	8
3.2.	SIGNING THE APPLET	8
3.3.	WRITING A SCRIPTABLE APPLET	9
3.4.	USING THE APPLET IN A WEB PAGE	13
4.	A WORKED DEMO WITH CODE	14
4.1.	LIVE DEMO	15

1. INTRODUCTION

1.1. ABSTRACT

The **Java Smartcard I/O API** has been defined by JSR 268 and introduced in version 6 of the platform. It implements communication with smartcards using ISO 7816-4 APDUs, to allow Java applications to interact with the smartcard's application.

The **Java Smartcard I/O API** runs on top of a PC/SC subsystem (Microsoft' PC/SC middleware on Windows, open source PCSC-Lite middleware on Linux and MacOS X). Therefore, this API ensures access to **SpringCard PC/SC readers** (Prox'N'Roll PC/SC, CSB6, CrazyWriter, EasyFinger, etc) from Java applications and applets.

This document provides an introduction to the **Java Smartcard I/O API** and explains how to build a smartcard-aware web page thanks to a Java applet.

1.2. AUDIENCE

This manual is designed for use by application developers. It assumes that the reader has expert knowledge of the Java language and platform.

1.3. PREREQUISITES

First step is to have a working Java SDK (JDK) and a working runtime (JRE) on your computer, supporting the version 6 of the platform.

Go to <http://java.sun.com/javase/downloads> to download and install the latest Java SE environment.

1.4. SUPPORT AND UPDATES

Interesting related materials (product datasheets, application notes, sample software, HOWTOs and FAQs...) are available at SpringCard's web site:

www.springcard.com

Updated versions of this document and others will be posted on this web site as soon as they are made available.

For technical support enquiries, please refer to SpringCard support page, on the web at address www.springcard.com/support .

2. JAVA SMARTCARD I/O API AT A GLANCE

2.1. IDENTIFICATION AND DOCUMENTATION

The **Java Smartcard I/O API** is implemented in `javax.smartcardio`.

The documentation is available online at

<http://java.sun.com/javase/6/docs/jre/api/security/smartcardio/spec/javax/smartcardio/package-summary.html>

2.2. OVERVIEW OF THE PRINCIPLE CLASSES AND METHODS

2.2.1. CardTerminals

The `javax.smartcardio.CardTerminals` class provides access to the list of available PC/SC readers.

2.2.2. CardTerminal

The `javax.smartcardio.CardTerminal` class is the representation of a PC/SC smartcard reader¹.

The method `isCardPresent` tells whether or not there's a card in the reader.

The method `waitForCardPresent` blocks until a card is inserted.

2.2.3. Card

The `javax.smartcardio.Card` class is the representation of a smartcard, inserted in a PC/SC reader.

The `getATR` method returns the ATR of the card.

Once the card is connected, communication is done through a `CardChannel` object.

2.2.4. CardChannel, CommandAPDU, ResponseAPDU.

Exchanges with the smartcard at APDU level are performed through the `javax.smartcardio.CardChannel` class.

The `transmit` method implements the exchange. A `CommandAPDU` object is passed to the reader down to the card, and the response of the card is returned by the reader in a `ResponseADPU` object.

¹ Some SpringCard readers provide more than one physical smartcard slot (CrazyWriter with its contactless slot plus SAM slots, CSB6 with its contactless slot, a ISO 7816 contact slot plus SAM slots, etc). In this case, they are seen as multiple PC/SC smartcard readers, and therefore as multiple `CardTerminal` instances under Java.

2.3. A TRIVIAL EXAMPLE

```
/* Get the list of readers */
CardTerminals terminalList;

TerminalFactory factory = TerminalFactory.getDefault();
terminalList = factory.terminals();

/* Choose a reader knowing its name */
CardTerminal MyReader = terminalList.getTerminal(ReaderName);

/* Connect to the card currently in the reader */
Card card = MyReader.connect("*");

/* Exchange APDUS with the card */
CardChannel channel = card.getBasicChannel();

byte[] ApduArray = {
    (byte) 0xFF,
    (byte) 0xCA,
    (byte) 0x00,
    (byte) 0x00,
    (byte) 0x00
};

CommandAPDU GetData = new CommandAPDU(ApduArray);
ResponseAPDU CardApduResponse = channel.transmit(GetData);

/* Disconnect */
card.disconnect(true);
```

3. JAVA SMARTCARD I/O IN AN APPLET

3.1. BACKGROUND

A **Java applet** is a small application, designed to be distributed through Internet and to run in a web browser.

An applet's main class inherits from `JApplet` :

```
import javax.swing.JApplet;
public class my_applet extends JApplet
{
    // ...
}
```

3.2. SIGNING THE APPLET

3.2.1. Why the PC/SC applet has to be signed?

To protect both computer's security and user's privacy, Java applets running in a web browser only have a very limited access to the computer's resources (*sandbox* principle). For instance, they are allowed to list the readers connected to the system, but are **not allowed** to connect to the smartcards themselves.

Therefore, only a signed applet could be given sufficient privileges to communicate with the smartcards.

3.2.2. Applet signing HOWTO

Before signing an applet, we need a signing key, to be created using `keytool`. Then we'll have to invoke `jarsigner` to sign the applet with our key, after every build.

3.2.3. Creation of the signing key

Use **Keytool** to generate a public-private key pair.

```
keytool -genkey -alias KeyName
```

`-genkey` flag means you are about to generate a key,

`-alias` flag allows you to name your key; provide a friendly name to identify your key.

Keytool asks to provide information during the process:

- *A Password*, to protect the private key,
- *First Name* and your *Last Name*,
- *Organisation Name*,
- *City*,

- *Country* and your *Country Code*.

Try to be as accurate as possible when entering these settings as they will be displayed every time a user will download your applet for the first time.

Once the key is generated, remember its *KeyName* and *Password* to be able to sign your projects.

3.2.4. Creation of the .jar file

After every build, the applet has to be packaged in a .jar file. It is the .jar file (and not the applet itself) that will be signed.

3.2.5. Signature

Once your applet has been built and your .jar file created, use **JarSigner** to sign it:

```
jarsigner -verbose my_applet.jar KeyName
```

Use the `-verbose` flag to have the different steps of the operation displayed.

`my_applet.jar` is the name of your JAR file.

KeyName is the name of the key created in 3.2.3. You will be prompted for the *Password*.

You may also provide the flag `-storepass Password` to specify this password on the command line.

3.3. WRITING A SCRIPTABLE APPLET

3.3.1. Why the applet shall be scriptable?

A **scriptable applet** is an applet whose methods may be accessed by the parent web page (from JavaScript typically).

We aim to develop a smartcard-aware web page, so the Java applet has to be scriptable. The GUI will be implemented in HTML, and the 'logic' in JavaScript. The applet is only an entry-point to the PC/SC subsystem.

3.3.2. Constraints on the JavaScript

When working with a signed applet, the JavaScript file should be signed too, to be able to call restricted functions (functions not allowed in the sandbox).

We want to avoid that, as signed JavaScript files are not easy to work with when developing and prototyping a web application.

The solution is to add code into the applet itself, to specify that the extended security privileges (gained by the applet's signature) shall be transmitted to the JavaScript code, too. This is done using an `AccessController.doPrivileged` block:

```
try
{
    AccessController.doPrivileged(
        new PrivilegedExceptionAction<Integer>()
        {
            public Integer run()
            {
                // Your code here
                return null;
            }
        });
} catch (PrivilegedActionException e)
{
    // catch block
    e.printStackTrace();
}
```

3.3.3. Passing parameters from the web page to the applet

An applet's method may receive input parameters from the JavaScript. The parameters could not be used 'as is' inside an `AccessController.doPrivileged` block. They must be copied into local `final` variables as follow:

```
public String TransmitArray(byte[] ApduIn)
{
    final byte[] ApduCmd = ApduIn;
    try
    {
        AccessController.doPrivileged(
            new PrivilegedExceptionAction<Boolean>()
            {
                public Boolean run()
                {
                    CommandAPDU getData = new
                        CommandAPDU(ApduCmd);

                    // ... transmit the C-APDU

                }
            });
    }
    // ... return the R-APDU
}
```

3.3.4. Returning a value from the applet to the web page

The return variable has to be a 'global' variable of the applet. It will be modified in the privileged area of the code, and returned outside the privileged block.

```
boolean CardIsPresent; // class-level variable

public boolean IsCardPresent()
{
    try
    {
        AccessController.doPrivileged(
            new PrivilegedExceptionAction<Boolean>()
            {
                public Boolean run()
                {
                    try
                    {
                        CardIsPresent =
                            MyReader.isCardPresent();
                    } catch (CardException e)
                    {
                        e.printStackTrace();
                    }
                    return true;
                }
            });
    } catch (PrivilegedActionException e)
    {
        e.printStackTrace();
    }

    return CardIsPresent; // this is the value returned
                        // to the web page
}
```

3.3.5. Invoking JavaScript functions from the applet

The applet may call JavaScript functions, for instance to update the GUI when an external event occurs (card inserted, card removed, ...).

This is done through the JSobject package:

Import the following packages:

```
import netscape.javascript.JSException;  
import netscape.javascript.JSObject;
```

Declare and initialize a JSObject:

```
private JSObject jso;  
try  
{  
    jso = JSObject.getWindow(this);  
} catch (JSException e)  
{  
    e.printStackTrace();  
}
```

Invoke the JavaScript function as follow:

```
jso.call("AJSFunction");
```

Parameters could be transmitted to the JavaScript function:

```
jso.call("AnotherJSFunction",  
        new String[] {"a string parameter"});
```

3.4. USING THE APPLLET IN A WEB PAGE

3.4.1. Inserting the applet in the HTML code

Use the following tag to insert your applet in your web page:

```
<object
  classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
  codebase="http://java.sun.com/products/plugin/1.3/jinstall-13-
  win32.cab#Version=1,3,0,0">

  <param name="CODE" value="my_applet_main_class.class">
  <param name="ARCHIVE" value="my_applet.jar">
  <param name="type" value="application/x-java-applet;version=1.3">
  <param name="scriptable" value="true">

  <comment></comment>

  <embed type="application/x-java-applet;version=1.3" hidden="true"
  code="my_applet_main_class.class" archive="my_applet.jar"
  scriptable="true"
  pluginspage="http://java.sun.com/products/plugin/1.3/plugin-
  install.html" MAYSCRIPT>
</object>
```

Of course change the bold items to reflect the name of your applet and the name of its .jar file.

The *MAYSCRIPT* attribute makes it possible to use JavaScript within the applet,

The *scriptable* attribute specifies that the applet is scriptable.

3.4.2. Invoking the applet's methods from JavaScript

You can call your applet function in your Javascript code like this:

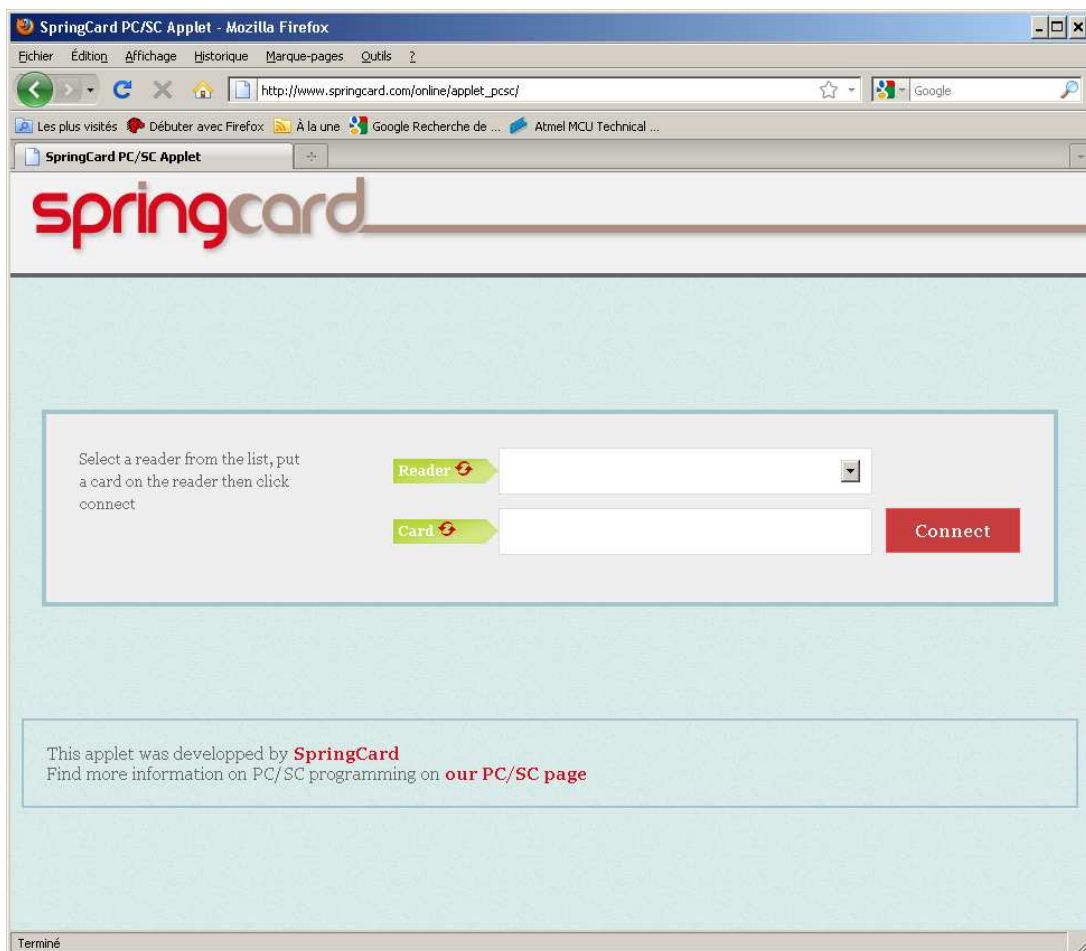
```
var retVal = document.embeds[0].AnAppletFunction(AParameter);
```

4. A WORKED DEMO WITH CODE

Open your web browser and navigate to URL

www.springcard.com/online/applet_pcsc/ .

Your browser will warn you about the applet's signature. Our applet has been signed with a self-signed certificate named 'www.springcard.com'; confirm that you accept it.



4.1. LIVE DEMO

The page lists the PC/SC reader(s) that are connected to your computer (click the 'Refresh' icon in the 'Reader' button to update the list).

Select a reader from the list, put a card on the reader then click connect

Reader SpringCard Prox'N'Roll 11 0

Card No Card on the reader

Connect

Once a card is present in² the selected reader, the 'Card' box says 'Card present'. Click 'Connect' to open a communication channel with it.

Card present on the reader, click Connect

Reader SpringCard Prox'N'Roll 11 0

Card Card Present

Connect

Once connected, the page displays 2 new text boxes.

Enter an APDU and click Transmit

Reader SpringCard Prox'N'Roll 11 0

Card Card Present

ADPU

Response

Disconnect

Transmit

² Or on, in case of a contactless reader.

Enter a valid Command APDU, and click 'Transmit'. Card's Response is displayed in the box at the bottom.

In this first example, we send 'Select Master File' according to ISO 7816-4. The card answers with the FCI (File Control Information) and 'OK' as Status Word (90 00).

Enter an APDU and click Transmit

Reader SpringCard Prox'N'Roll 11 0

Card Card Present Disconnect

ADPU 00 a4 00 00 02 3f00 Transmit

Response 85 17 00 01 00 00 00 02 02 00 00 01 04 00 00 00 05 05 05 00 00 00 00 00 90 00

In this second example, we send 'Get Card UID' according to PC/SC v2, chapter 3. The reader (not the card) returns the card's serial number and 'OK' as Status Word (90 00).

Enter an APDU and click Transmit

Reader SpringCard Prox'N'Roll 11 0

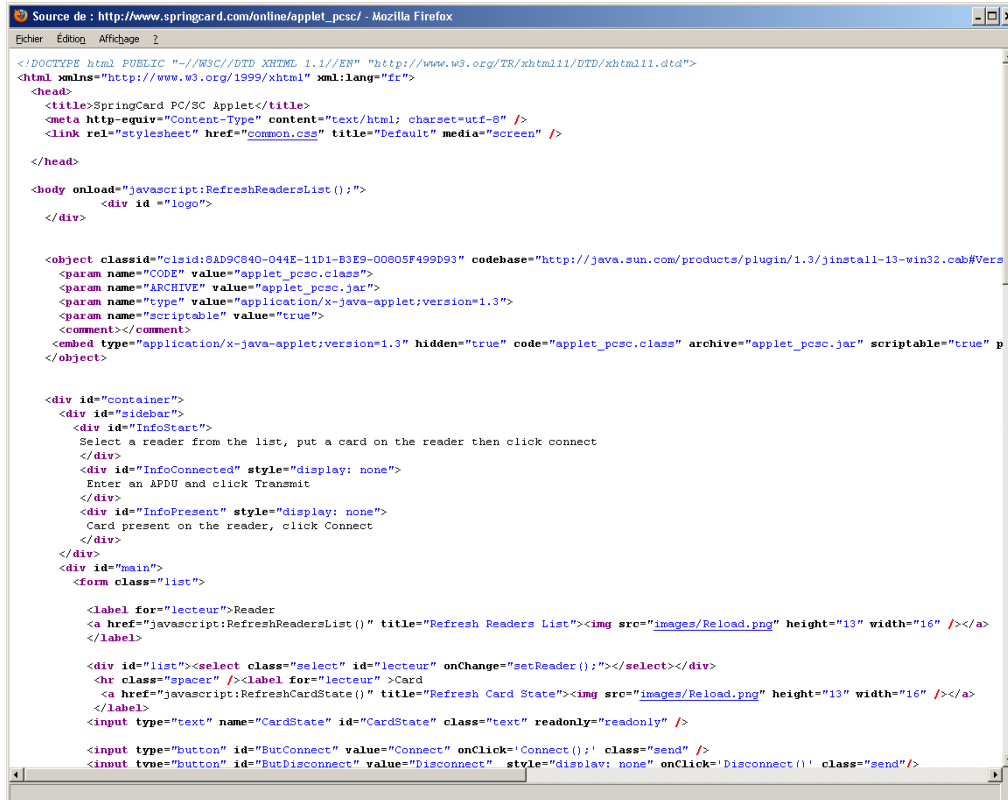
Card Card Present Disconnect

ADPU ff ca 00 00 00 Transmit

Response 25 3F 4E 1B 90 00

4.2. SOURCE CODE

Use your web browser to display the source of the page.



```
Source de : http://www.springcard.com/online/applet_pcsc/ - Mozilla Firefox
Fichier  Edition  Affichage  2
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr">
<head>
<title>SpringCard PC/SC Applet</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<link rel="stylesheet" href="common.css" title="Default" media="screen" />
</head>
<body onload="javascript:RefreshReadersList();">
<div id="logo">
</div>
<object classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93" codebase="http://java.sun.com/products/plugin/1.3/jinstall-13-win32.cab#Vers
<param name="CODE" value="applet_pcsc.class">
<param name="ARCHIVE" value="applet_pcsc.jar">
<param name="type" value="application/x-java-applet;version=1.3">
<param name="scriptable" value="true">
<comment></comment>
<embed type="application/x-java-applet;version=1.3" hidden="true" code="applet_pcsc.class" archive="applet_pcsc.jar" scriptable="true" p
</object>
<div id="container">
<div id="sidebar">
<div id="InfoStart">
Select a reader from the list, put a card on the reader then click connect
</div>
<div id="InfoConnected" style="display: none">
Enter an APDU and click Transmit
</div>
<div id="InfoPresent" style="display: none">
Card present on the reader, click Connect
</div>
</div>
<div id="main">
<form class="list">
<label for="lecteur">Reader
<a href="javascript:RefreshReadersList()" title="Refresh Readers List"></a>
</label>
<div id="list"><select class="select" id="lecteur" onChange="setReader();"></select></div>
<hr class="spacer" /><label for="lecteur" >Card
<a href="javascript:RefreshCardState()" title="Refresh Card State"></a>
</label>
<input type="text" name="CardState" id="CardState" class="text" readonly="readonly" />
<input type="button" id="ButConnect" value="Connect" onClick='Connect();' class="send" />
<input type="button" id="ButDisconnect" value="Disconnect" style="display: none" onClick='Disconnect();' class="send"/>
</form>
</div>
</div>
```

There are only 6 very short JavaScript functions to drive the applet:

- RefreshReadersList
- setReader
- RefreshCardState
- Connect
- SendApdu
- Disconnect

DISCLAIMER

This document is provided for informational purposes only and shall not be construed as a commercial offer, a license, an advisory, fiduciary or professional relationship between PRO ACTIVE and you. No information provided in this document shall be considered a substitute for your independent investigation.

The information provided in this document may be related to products or services that are not available in your country.

This document is provided "as is" and without warranty of any kind to the extent allowed by the applicable law. While PRO ACTIVE will use reasonable efforts to provide reliable information, we don't warrant that this document is free of inaccuracies, errors and/or omissions, or that its content is appropriate for your particular use or up to date. PRO ACTIVE reserves the right to change the information at any time without notice.

PRO ACTIVE does not warrant any results derived from the use of the products described in this document. PRO ACTIVE will not be liable for any indirect, consequential or incidental damages, including but not limited to lost profits or revenues, business interruption, loss of data arising out of or in connection with the use, inability to use or reliance on any product (either hardware or software) described in this document.

These products are not designed for use in life support appliances, devices, or systems where malfunction of these products may result in personal injury. PRO ACTIVE customers using or selling these products for use in such applications do so on their own risk and agree to fully indemnify PRO ACTIVE for any damages resulting from such improper use or sale.

COPYRIGHT NOTICE

All information in this document is either public information or is the intellectual property of PRO ACTIVE and/or its suppliers or partners.

You are free to view and print this document for your own use only. Those rights granted to you constitute a license and not a transfer of title: you may not remove this copyright notice nor the proprietary notices contained in this document, and you are not allowed to publish or reproduce this document, either on the web or by any mean, without written permission of PRO ACTIVE.

Copyright © PRO ACTIVE SAS 2010, all rights reserved.

EDITOR'S INFORMATION

PRO ACTIVE SAS company with a capital of 227 000 €
RCS EVRY B 429 665 482
Parc Gutenberg, 13 voie La Cardon
91120 Palaiseau – France