**Developer Guide**

**jWebSocket**

**API for smart card management**

**v1.0**

**Version Control**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 28/3/2012 | 1.0 | Document Creation | Marta Rodríguez Freire |

# Overview

The main objective of the project is to increase the potential of the framework jWebSocket, providing it with new features that allow you to develop web applications in real time with the benefits of smart cards. To achieve this was developed an API for smart card management, that allows develop web applications using the smart card, giving this applications a better security and usability.

# Infrastructure, Model, Approach

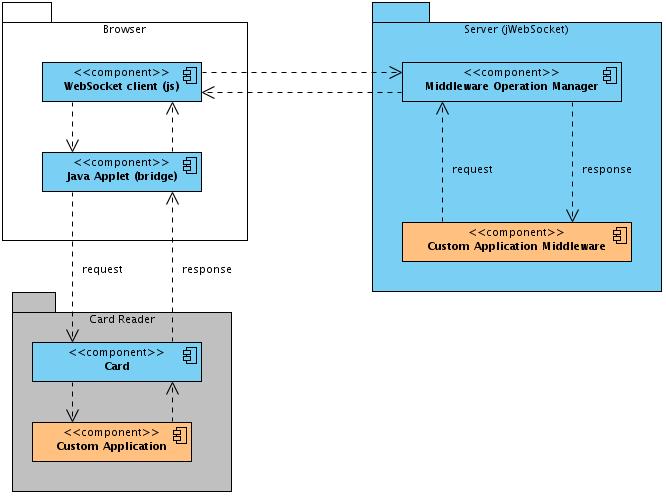


Image 1: This diagram describes the API infrastructure.

The API allows the smart card management in web applications developed with the jWebSocket framework. On the client side must have a browser that supports the WebSocket protocol. The jWebSocket client communicates with the jWebSocket server, sending a request to the Middleware Operation Manager component that handles the request and send a response to the smart card. This response communicates with the jWebSocket client and Java applet, who works as a bridge between the client and the card reader, the smart card process the response and send another response to the middleware on the server. In the card reader is inserted the smart card, it contains all the applications.

**Use design patterns**

The design patterns used in the development of the solution was observer, this pattern was used to detect and handle the client and server events.

**EventPlugIn or TokenPlugIn**

The solution was developed using EventPlugin, this is aimed at developing professional applications with a set of requirements as could be security, validation, cache and its more focus to object oriented programming.

# Requirements and prerequisites

The requirements needed for the continued development of the solution are:

* Network connection
* Card reader to accomplish with the standard PC/SC.
* The framework used for the smart cards is JavaCard.

# Modules, Structure

The main aspects that describe the organizational structure of the API for smart card management is reflected below:

**In the client side**

|  |  |
| --- | --- |
| Project Name: | SmartCardManager |
| Location of the sources in the SVN server: | http://repo.hab.uci.cu/svn/tesis/Tercer\_Corte\_de\_Tesis/JWS/Marta\_Rodriguez\_Freire/Codigo/ |
| SVN branch: | jWebSocket-1.0 |
| JAR Module: | SmartCard.jar |
| Structure Package | paquetes |
| org.jwebsocket.cardplataform:  It contains the definition and implementation of the components. | |
| org.jwebsocket.cardplataform.api:  It contains all the interfaces that are implemented by the components. | |
| org.jwebsocket.cardplataform.listener:  Contains the definition and implementation of the notifications for events in the smart card reader. | |
| org.jwebsocket.cardplataform.util:  It contains the generic libraries with utility functions used in the extension. | |

**In the server side**

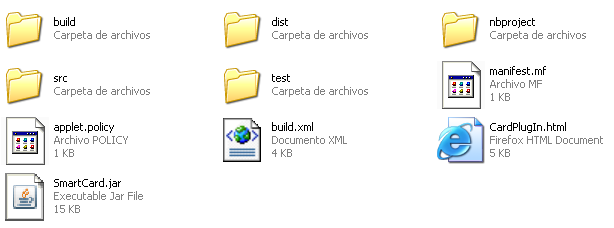
The application for the server side it’s contain in EventsPlugin (Extension for the jWebSocket framework for enterprise web application development).

In EventPlugin found the **org.jwebsocket.eventmodel.plugin.jc** package, which contains all definition and implementation to manage sending APDU commands and middleware

* 1. **Code Structure**

The structure of the source code from the developed library shown below:

**In the client side**

*****Image 2: Structure from the directory that contains the solution folders.*

**Description of the elements:**

src: This directory stores files with the source code (.java) that are part of the solution.

SmartCard.jar: File that lets you collect in a single file several different files, storing them in a compressed format.

build, dist, nbproject, test: Folders are automatically created when you build a NetBeans project.

* 1. **Description of the API packages**

**In the client side**

The package **org.jwebsocket.cardplataform** consists of the following classes, specifying the most relevant methods

|  |  |  |
| --- | --- | --- |
| **CardPlugIn Class: This is the Java applet that acts as a bridge for communication between the card reader and JavaScript.** | | |
| **Type** | **Method** | **Description** |
| **String** | **getActiveTerminalNames()** | Return a comma separated string with the names list of the active terminal |
| **String** | **transmit(String aTerminalName, String aCommandAPDU)** | Transmit and APDU command to the card |

The package **org.jwebsocket.cardplataform.listener** consists of the following classes, specifying the most relevant methods

|  |  |  |
| --- | --- | --- |
| **CardTerminalListener Class: Notify the JavaScript side about the smart card reader events** | | |
| **Type** | **Method** | **Description** |
| **void** | **onTerminalReady(CardTerminal aTerminal)** | Notify the JavaScript side that a smart card reader is ready |
| **void** | **OnTerminalNotReady(CardTerminal aTerminal)** | Notify the JavaScript side that a smart card reader is not ready |

**In the server side**

The package **org.jwebsocket.eventmodel.plugin.jc** consists of the following classes, specifying the most relevant methods

|  |  |  |
| --- | --- | --- |
| **JCPlugin Class: Extension that allows you to send APDU commands from server to client.** | | |
| **Type** | **Method** | **Description** |
| **void** | **registerTerminal(String aConnectorId, String aTerminal)** | Register a terminal on a connector as ready. |
| **void** | **unregisterTerminal(String aConnectorId, String aTerminal)** | Unregister a terminal on a connector. |
| **void** | **transmit(String aConnectorId, String aTerminalId, CommandAPDU aCommand, JcResponseCallback aCallback)** | Transmit a Command APDU to the client smart card terminals. |

|  |  |  |
| --- | --- | --- |
| **JcResponseCallback Class: Allows handle successfully the responses coming from the client.** | | |
| **Type** | **Method** | **Description** |
| **boolean** | **isValid(ResponseAPDU aResponse, String aFrom)** | Execute custom validations in client responses. |

# Source Code

* 1. **Code Standard**

The common code standards used to develop the solution are the same defined to jWebSocket framework.

* 1. **Reuse of Components**

For future developments can reuse libraries containing the API, that allows the development of web applications using smart cards.

1. **Interfaces**

The solutions don’t require an interface for right operation.

1. **Frameworks, Libraries and Tools**

For the solution developed was used for the client side JavaCard framework, this framework has a GNU licensed (General Public License). This platform provides the base for achieving interoperability between manufacturers and security in smart card environments or devices with resource constraints. This technology aims to bring the benefits of Java software development in the world of smart cards. For more information about the download and update process can visit the following link:

<http://www.oracle.com/technetwork/java/javacard/overview/index.html>

1. **Database and Persistence**

To use the API to manage smart cards in web applications developed with the jWebSocket framework not require a database manager.

1. **Hardware**

To use the API to manage smart cards in web applications developed with the jWebSocket framework must have the following characteristics in the work environment:

* OpenJDK 1.7 or higher installed to run the applet in the client browser.
* Card reader to accomplish with the standard PC/SC

1. **Security**

*This section describes all security aspects of the solution. Here it is more about the implementation rather than the configuration, which is part of the Administrator Guide.*

* *How are credentials maintained (e.g. to databases or to other servers like Mail-Server or Twitter).*
* *What about encyrption/SSL options, how to ensure security for the users.*
* *How is the user data privacy ensured?*
* *How is this service protected against spam or other attacks?*
* *What validation criteria is applied and how?*
* *Filters, black- and whitelists, firewall considerations (open ports and protocols)*
* *What other mechanism have been applied in security aspects (or are supposed to be applied in the future)?*
* *What known security issues do currently exist and are these acceptable for a production release? What protection is possible? Are workarounds available?*

1. **Tests, Quality Assurance, Continuos Integration**

*This section describes the execution of automated and manual tests for quality ensurance. Right now we do not yet have continuos integration, so this is still pending.*

* *Test cases, functional (JUnit, Jasmine) and UI tests (iMacros, pending).  
  Which automated tests are available, how are they invoked, how are the test cases integrated into our deployment test suite? Where are the reports?  
  Are there known issues (traps, also refer to bug-list), what are the reasons, is this approved and what can be done in case of these issues (work-arounds)?*
* *What about the error and potential issue handling (e.g. database not available, network connection broken other resources do not exist).*
* *What about the general error and exception handling?*
* *If required (because not (yet) covered or not coverable), which manual tests can/have to be run before deployment?*

1. **Profiling**

*This section describes the methods and strategies for continous improvements.*

* *"Improvements" means provisions for higher speed, less memory consumption, simpler maintainability or similar, which do not affect the functionality in general (e.g. "the solutions uses a quicksort instead of a bubblesort", or "in future the solution should use MongoDB instead of MySQL because..."). It does not mean bug-fixing.*

1. **Reference**

*This is a structured lookup document, here it is about a complete reference of details, just with keywords or short sentences rather than much text,*

* *Complete JavaDocs or JavaScriptDocs (inline documentation) are mandatory for publishing (classes, constants, variables, methods, arguments, result, examples, properties, flags), all classes and methods in all languages should be fully documented for efficient knowledge management and maintainability.  
  The JavaDocs and JavaScript Docs as well as other automatically generated documention which is targetted for online-distribution (on our website), does not need to be printed in the Developer Guide, here a reference to the online area is sufficient.*
* *A "Token Reference" is mandatory for the Developer Guide (see e.g. the Channel Documentation on our WebSite)*
* *List of files, as far as not automatically generated by any inline documentation tool, like JavaDocs. (e.g. a certain set of configuration files and their purpose), for configuration files, list of settings and possible options. This is also important for the packaging and deployment of the solution.*