**Administrator Guide**

**jWebSocket Management Desk 1.0**

Marcos Antonio González Huerta

1. **Overview**

jWebSocket Management Desk is an application to provide maintenance for Java applications developed with jWebSocket framework that are running on the server. This application is integrated into this framework without the need for any additional requirements other than those that are required so you can run the jWebSocket server. This tool uses custom class loaders that are available in the JCL library v2.2.

This tool allows developers and companies that use it disinterested in the design and implementation of a method that ensures the maintenance process. jWebSocket Management Desk also enables a significant reduction in time and the complexity of developing applications using jWebSocket framework.

1. **Infrastructure, Model, Approach**

JWebSocket Desk Management application is developed in Java on the server side and JavaScript on the client side. ExtJS technology was used to develop the client to achieve greater simplicity and comfort to the sights, while for the server side we used jWebSocket framework thus achieving greater compatibility and integration. The architecture for the development of the application was client-server architecture or 2 layers as it is also known.

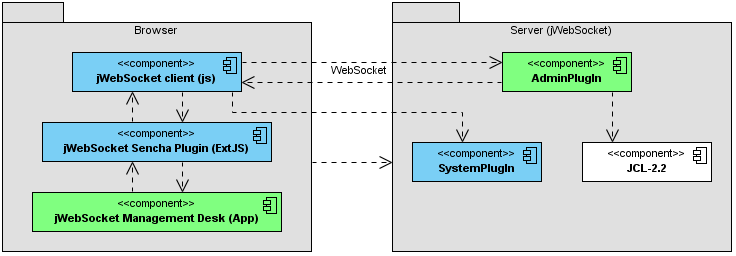


Fig. 1 Components diagram of the application.

**Description:**

* Communication between the client and jWebSocket server is through WebSocket protocol.
* All components of the client, ie belonging to the Browser package are closely related. JWebSocket client is used by Sencha Plugin to establish the connection with the server, in the same way jWebSocket Management Desk uses Sencha Plugin.
* SystemPlugIn component belonging to jWebSocket server is responsible for providing the safety of the processes and actions that are made in the application, controlling access to them as authorized.
* The JCL-2.2 (Jar Class Loader) component is a representation of the JCL library for reloading classes in the Java language. Therefore it is used by the management component to load or reload the plug-ins or filters of the applications running on the server.

1. **Requirement and Prerequisites**

The requirements for the continued development of jWebSocket Management Desk module are:

* Network connection
* Have installed an IDE to work with Java on the server side and Javascript, HTML and CSS on the client side.
* To have Maven installed properly.

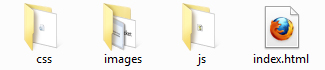
1. **Modules, Structure**

**On the server side:**

|  |  |
| --- | --- |
| Project Name: | jWebSocket Management Desk |
| Location of the sources in the SVN server: | <https://jwsdev.org:9443/svn/jWebSocket/branches/jWebSocket-1.0/jWebSocketPlugIns/jWebSocketAdminPlugIn> |
| SVN branch: | jWebSocket-1.0 |
| Maven dependencies: | <dependency>  <groupId>org.jwebsocket</groupId>  <artifactId>jWebSocketCommon</artifactId>  <version>1.0</version>  </dependency>  <dependency>  <groupId>org.jwebsocket</groupId>  <artifactId>jWebSocketServerAPI</artifactId>  <version>1.0</version>  </dependency>  <dependency>  <groupId>org.jwebsocket</groupId>  <artifactId>jWebSocketServer</artifactId>  <version>1.0</version>  </dependency> |
| JAR module: | jWebSocketAdminPlugIn-1.0.jar |
| Package structure: | NameSpace: org.jwebsocket.plugins.admin  Pantallazo |
| org.jwebsocket.plugins.admin:  Contains the management plugin of the framework called "AdminPlugIn", which uses the services implemented at the "AdminPlugInService" class which are located in the same package. | |

**On the client side:**

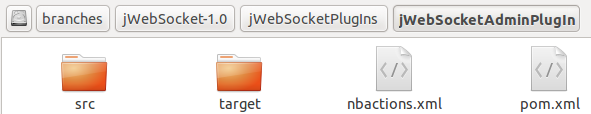
jWebSocket Management Desk client application located in *“branches\jWebSocket-1.0\jWebSocketClient\web\apps\jwsManager*” contains the following folder structure:



In the *"js"* folder is the *"jwsManager.js"* JavaScript file which contains all the logic of the client application. This client application depends on two JavaScript files:

* [jWebSocket.js](https://jwsdev.org:9443/svn/jWebSocket/branches/jWebSocket-1.0/jWebSocketClient/web/res/js/jWebSocket.js): JWebSocket client library for JavaScript.
* [jwsExtJSPlugin.js](https://jwsdev.org:9443/svn/jWebSocket/branches/jWebSocket-1.0/jWebSocketClient/web/res/js/jwsCache.js): Library containing the ExtJS Plugin for jWebSocket.

**4.1 Code Structure**



**Elements Description:**

**src:** This directory contains all the source code of classes and libraries of the solution.

**target:** This directory temporarily stores the compiled source code, its content is not included in the version control.

**nbactions.xml:** Project configuration file created by the NetBeans IDE. The content of this file is not included in the version control.

**pom.xml:** Configuration file that contains information about the project and details of the settings used by Maven to build the project.

**4.2 Package Description**

jWebSocket Management Desk application is structured in one packege org.jwebsokcket.plugins.admin which groups the following classes:

|  |  |
| --- | --- |
| **Class** | **Description** |
| AdminPlugIn | Class that contains all the administrative functions of jWebSocket server. Among these are the features introduced with the implementation of the jWebSocket Desk Management for the maintenance of applications at runtime. This class executes the functionalities found in the management service in the *AdminPlugInService* class. |
| AdminPlugInService | Class that implements each one of the features that are shown in jWebSocket Management Desk. This class separates the business logic of this tool from the others management features that already exist. |

1. **Source Code**

**5.1 Common Code Standards**

To develop the application used the standard of code of the jWebSocket framework.

**5.2 Reusable Components**

For future developments can reuse this application, using each one of the components implemented in either the client or the server only making minor changes in most of these.

**AdminPlugIn (server)**: This component is performing the administrative tasks of jWebSocket framework, so its logic is identified with a large percentage with the architecture of this framework, making difficult to integrate the entire solution in another framework other than jWebSocket. However, the use of design patterns and some algorithms that appears in this plugin can be reused without any consequences.

**JCL (server)**: Jar Class Loader, library integrated to jWebSocket framework to achieve the dynamic class loading. Fully flexible, it allows the integration with Spring IoC and other frameworks.

**Web Client (ExtJS)**: This client is implemented multiple views and components. Since ExtJS features they all can be reused without major changes. You can reuse the authentication component and view, the component that makes up a table for recording traces or other information, allowing you to search through all the information displayed. ExtJS largely guarantees the decoupling of components and its reuse.

1. **Interfaces (if applicable)**

**Network protocols used**

The application Shared Canvas Demo was developed with the jWebSocket framework, which used to establish communication, the WebSocket protocol (ws). WebSocket is a technology that provides a bidirectional communication channel and full-duplex over a single TCP socket. However it can be accessed through JMX technology, using the RMI / HTTP protocols.

1. **Frameworks, Libraries and Tools**

ExtJS 4 is a framework for building Rich Internet Applications (RIA for its acronym in English). It is based on open source JavaScript libraries, lightweight, high performance, compatible with most browsers that allow you to create dynamic web pages and interfaces. It includes technologies such as Ajax, DHTML, XML, XSLT, JSON, CSS and DOM. It included most of the Web forms controls including tables to display data and similar elements to desktop programming such as forms, panels, toolbars, menus and more.

JCL v2.2 (<https://github.com/kamranzafar/JCL/>) is a Java library specializes in custom class loaders, configurable and dynamic. JCL is designed to create, manage and manipulate Java class loaders in isolation in different frameworks and web applications.

1. **Database and Persistence (if applicable)**

jWebSocket Management Desk does not require the use of a database for its operation.

1. **Hardware (if applicable)**

jWebSocket Management Desk does not require the use of additional hardware for its operation.

1. **Security**

In order to ensure the security of jWebSocket Desk Management was used the authentication system that has the jWebSocket framework, which in turn implements one of the authentication methods of Spring framework. The users, its passwords encrypted in MD5 and the application roles are stored in a configuration file of the jWebSocket server.

1. **Tests, Quality Assurance and Continuous Integration**

*This section describes the execution of automated and manual tests for quality insurance. Right now we do not yet have continuous 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.*