# Developer Guide

**jWebSocket**

**jWebSocket Watchdog Client  
1.0**

# Control de versiones English please

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| *1/4/2012* | *1.0* | *Creating the Document* | *Lester Alfonso Zaila Viejo.* |

# 1. Introduction.

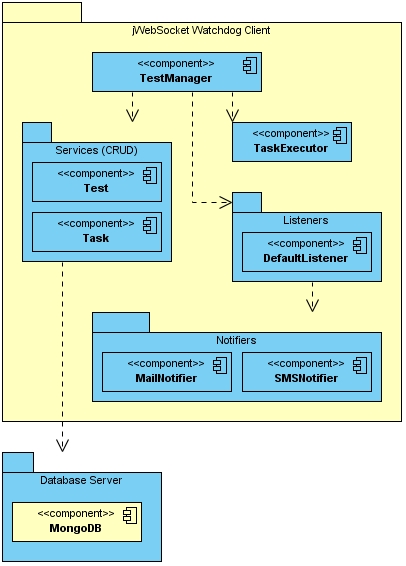
This guide describes how to modify and use jWebSocket Watchdog Client and develop desktop applications.

The jWebSocket Watchdog Client is the guardian of jWebSocket server, allows a WebSocket connection to the server to perform tests both the server and applications running on the same jWebSocket system/machine/instance ?. In return it analyzes the test result and notifies the system administrator in case of failures.

The Application Programming Interface (API) of the jWebSocket Watchdog Client is simple and intuitive, confirming that jWebSocket is a highly configurable and robust technology.

# 2. Infrastructure solution.

# Please extract image and link them as external files to use them also for the web documentation!

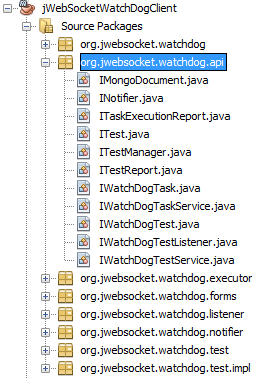


**Imagen1: Diagram describing the structure of the jWebSocket Watchdog Client**

# 3. Modularization of the extension

The jWebSocket Watchdog Client is separated from jWebSocket server, in order to not get affected by a potential server. It is a standalone application that will monitor server status of the jWebSocket server. Can/should it run on a separate machine?

The following diagram can observe the structure of the jWebSocket Watchdog Client.

****

**Imagen 2: Structure of jWebSocket Watchdog java Client.**

# 4. Source code structure

**jWebsocket Watchdog Client:**

# C:\Users\Lester\Desktop\Interfaces\archivos.png

# Description of the folders:

**conf:** This directory contains the configuration file of the application "config.xml", containing all data necessary for the application to start. Don’t miss to explain the content and options of the config.xml in the Administrator manual.

**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 contents of this file may or may not be in version control, to include depends on the decision of the development team.

**pom.xml:** Module configuration file required for the Maven tool. This file defines among other things (a bit more specific please, what things ?), the dependencies.

4.1 Configuring Maven

**jWebSocket Watchdog Client** Module is configured via pom.xml file, an XML representation of the Maven project within the project folder to configure the dependencies and basic properties of the project, then you will see how it is used:

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>${project.groupId}</groupId>

<artifactId>watchdogcommon</artifactId>

<version>1.0</version>

<type>jar</type>

</dependency>

<dependency>

<groupId>${project.groupId}</groupId>

<artifactId>watchdogcommon</artifactId>

<version>${project.version}</version>

</dependency>

<dependency>

<groupId>${project.groupId}</groupId>

<artifactId>watchdogcommon</artifactId>

<version>${project.version}</version>

</dependency>

<dependency>

<groupId>org.jwebsocket</groupId>

<artifactId>jWebSocketJavaSEClient</artifactId>

<version>1.0</version>

</dependency>

<dependency>

<groupId>org.mongodb</groupId>

<artifactId>mongo-java-driver</artifactId>

<version>2.6.3</version>

</dependency>

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-email</artifactId>

<version>1.1</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>3.0.5.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>3.0.5.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-tx</artifactId>

<version>3.0.5.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-core</artifactId>

<version>3.0.5.RELEASE</version>

</dependency>

<dependency>

<groupId>org.mongodb</groupId>

<artifactId>mongo-java-driver</artifactId>

<version>2.6.3</version>

</dependency>

</dependencies>

Mongo Java Driver twice inside? One should be sufficient to convince the system ;-)

4.2 Packet structure

The package **org.jwebsocket.watchdog.api** contains all interfaces, like shown in the below table:

|  |  |
| --- | --- |
| **Class** | **Description** |
| IMongoDocument | Can store data such as documents. |
| INotifier | Allows system administrator to notify the failures of the application. |
| ITaskExecutionReport | Create a task list of reports for analysis by the listeners. |
| ITest | Defines the tests. |
| ITestManager | Manages the tests. |
| ITestReport | Responsible for shaping the outcome and put it in a report. |
| IWatchDogTask | Contains all tasks to be executed at jWebSocket server. |
| IWatchDogTaskService | Takes care to add, delete, update and list tasks. |
| IWatchDogTest | Contains all the tests to be executed at jWebSocket server. |
| IWatchDogTestListener | Responsible for data processing for listeners. |
| IWatchDogTestService | Takes care to add, delete, update and list the test. |

The package **org.jwebsocket.watchdog.executor** contains:

|  |  |
| --- | --- |
| **Class** | **Description** |
| TaskExecutor | Responsible for running all tests to jWebSocket server. |

The package **org.jwebsocket.watchdog.listener** contains:

|  |  |
| --- | --- |
| **Class** | **Description** |
| WatchDogMailListener | Is responsible for hearing the test results by email to notify the system administrator. |
| WatchDogTestListener | This is the default listener, which prints the result of tests on the system console. |

The package **org.jwebsocket.watchdog.test.impl** contains the implementation, a basic test example:

|  |  |
| --- | --- |
| **Class** | **Description** |
| BasicTest | This is a sample implementation of a test for a basic query to the jWebSocket server, based on this prototype developers can build more complex tests. |

**Standard code**

View Standard Template Code: Well, this cannot be accessed from anywhere outside, please provide in the SVN or remove this from here.

<http://repo.hab.uci.cu/svn/tesis/Segundo_Corte_de_Tesis/JWS/Lester%20Zaila/1er%20Corte/10-Plantilla_Estandar_de_codigo.doc>

Detailed description of the application source code.

The most important methods of jWebSocket Watchdog Client are:

In class **TaskExecutor**: the **run** method:

Gets the tasks that are scheduled for execution and executes the appropriate time.

@Override

public void run() {

try {

while (true) {

List<IWatchDogTask> lList = mTaskService.list();

int lComparison;

Date lDate;

for (IWatchDogTask lTask : lList) {

if (lTask.getType().equals("m")) {

//getting the frequency

lDate = addMinutes(lTask.getEveryNMinutes());

} else if (lTask.getType().equals("h")) {

//getting the frequency

lDate = addHour(lTask.getEveryNHours());

} else {

lDate = addDays(lTask.getEveryNDays());

}

lComparison = lDate.compareTo(new Date(lTask.getLastExecution()));

if (0 >= lComparison) {

//Execute the task

mTestManager.execute(lTask);

}

}

Thread.sleep(60000);

}

} catch (InterruptedException ex) {

Logger.getLogger(TaskExecutor.class.getName()).log(Level.SEVERE, null, ex);

}

}

In **TestManager** class: the **execute** method:

After being executed the test, the Test Manager creates the report notifying the listeners implementing the results.

@Override

public TaskExecutionReport execute(IWatchDogTask aTask) {

List<ITestReport> lTestReports = new LinkedList<ITestReport>();

aTask.execute(mClient, lTestReports);

TaskExecutionReport lResult = new TaskExecutionReport(aTask, lTestReports);

for (IWatchDogTestListener lListener : mListeners) {

lListener.process(lResult);

//processing the results

}

return lResult;

}

At the class **WatchDogMailListener**: method **process**: Responsible for e-mail notification to the system administrators, the results obtained from the report failed execution.

@Override

public void process(TaskExecutionReport aReport) {

String lMessage = "";

List<String> lTests = new FastList();

for (ITestReport lTestReport : aReport.getReport()) {

if (lTestReport.getTestResult().equals(Test.NOT\_OK)) {

lTests.add("- Test: '" + lTestReport.getTestId() + "'{"

+ lTestReport.getTestDescription() + "} failed.");

//creating the strucure of the message

}

}

if (!lTests.isEmpty()) {

lMessage = "The task '" + aReport.getTask().getId()

+ "' has failed. Failed execution of the following tests: \n";

for (String lTest : lTests) {

lMessage = lMessage + "\n" + lTest;

}

mNotifier.notify(lMessage + "\n\nPlease check the server status!");

//sending the message!

}

}

# What about the SMS service here?

# 5. Conclusions

After reading this guide, the developer must be familiar with the structure of jWebSocket Watchdog Client for execute Test to the jWebSocket server, knowing the main implementation details will also have an advanced knowledge of the dependencies and files needed to configure the same.

The fact that today holds jWebSocket this new tool ensures greater availability and secure execution of applications without fear of prolonged failure, since it will be instantly notified of any adverse event.