Project Report

Java Web Start for SIP Communicator

by Gabor JAGER
gabor.jager@ulp.u-strasbg.fr

LOUIS PASTEUR UNIVERSITY – STRASBOURG
May 2008

1. Introduction

I am a student of Louis Pasteur University (Strasbourg). I worked on Java Web Start for SIP Communicator. It was a project work for my studies. Actually, I am doing my MSc in informatics.

My job was to find a solution how we can launch SIP Communicator from Java Web Start (directly from a website). As SIP Communicator is based on Felix, the question was rather how we can launch a Felix based project which has huge sized bundles from Java Web Start. (Actually, SIP Communicator has about 53 bundles whose size is more than 30 Mbyte.)

I have to write a full progress report, but at the end you will find a tutorial about the final solution which can be very useful if somebody wants to use Felix for other projects.

2. Launching Felix by Java Web Start

Let's start by understanding a simple Felix launch command what we will transform to a Java Web Start version. We will use only a few bundles.

For starting a Felix based application we need to configure Felix in the `felix.properties` file, and then we can launch our application. We only have to specify the `felix.properties` path and the classpath to Felix's main:

```
> javaw -classpath "lib/felix.jar" -Dfelix.config.properties=file:./lib/felix.properties
org.apache.felix.main.Main
```

That is what we have to transform to Java Web Start version. For Java Web Start we have to modify a little bit the `felix.properties` file. Originally, (for a local version) to start automatically some bundles we have something like this in the `felix.properties` file:

```
...
felix.auto.start.1=file:bundle/org.apache.felix.bundlerepository-1.0.0.jar
felix.auto.start.2=file:bundle/mybundle.jar
...
```

For every bundle we have to define a full HTTP path. Something like this:

```
...
felix.auto.start.1=http://www.mysite.com/bundle/org.apache.felix.bundlerepository-1.0.0.jar
felix.auto.start.2=http://www.mysite.com/bundle/mybundle.jar
...
```

Java Web Start needs a JNLP file, so we have to create `felix_launch.jnlp`:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- JNLP File -->

<jnlp spec="1.0+" codebase="http://www.mysite.com/" href="felix_launch.jnlp">
   <information>
      <title>Felix Launcher</title>
      <vendor>Gabor JAGER (gabor.jager@ulp.u-strasbg.fr)</vendor>
      <description>This is a basic Felix launcher for my progress report</description>
      <homepage href="http://felix.apache.org"/>
      <description kind="short">Basic Felix launcher</description>
   </information>
   <security>
      <all-permissions/>
   </security>
   <resources>
      <j2se version="1.6+" href="http://java.sun.com/products/autodl/j2se"/>
      <jar href="felix.jar"/>
      <property name="felix.config.properties" value="http://www.mysite.com/felix.properties"
/>
   </resources>
   <application-desc main-class="org.apache.felix.main.Main"/>
</jnlp>
```

And we need to add a link to the JNLP file to our `index.html` file:

```
<a href=”felix_launch.jnlp”>Launch Felix</a>
```

Before uploading the files we have to digitally sign every JAR file which is in the JNLP file (only `felix.jar` in our example). If there are more than one JAR file in the JNLP's resources section, we have to sign all of them with the same signature. There is a good tutorial on Sun's website: http://java.sun.com/developer/technicalArticles/WebServices/distSJSIS/

Finally, we can upload our files to the server and we can test our example. In this example you have to follow this directory structure:

```
-> www.mysite.com/
       -> bundle/
              org.apache.felix.bundlerepository-1.0.0.jar
              mybundle.jar
              ...
       felix.properties
       felix.jar
       felix_launch.jnlp
       index.html
```

Now in a browser on the http://www.mysite.com/index.html page we can launch our Felix based application directly from web.

3. Launching SIP Communicator by Java Web Start

To use our example with SIP Communicator we have to modify `felix.client.run.properties` file (this is SIP Communicator's `felix.properties` file) as written above, sign SIP Communicator's `felix.jar` (and the other JAR files) before upload to server and modify our JNLP file:

```
...
   <resources>
      <j2se version="1.6+" href="http://java.sun.com/products/autodl/j2se"/>

      <jar href="sip/lib/jdic-all.jar" />
      <jar href="sip/lib/jdic_stub.jar" />
      <jar href="sip/lib/felix.jar" main="true" />
      <jar href="sip/lib/sc-launcher.jar" main="true" />

      <property name="felix.config.properties"
value="http://www.mysite.com/sip/lib/felix.client.run.properties" />
      <property name="java.util.logging.config.file"
value="http://www.mysite.com/sip/lib/logging.properties" />

   </resources>
   <application-desc main-class="net.java.sip.communicator.launcher.SIPCommunicator"/>
...
```

If we launch SIP Communicator with this solution it works, but…

When we click on the JNLP link Java Web Start launches. We have a download progress bar for JAR files included to the resource part of JNLP file. In our case Java Web Start download only `felix.jar` (360 Kbyte):

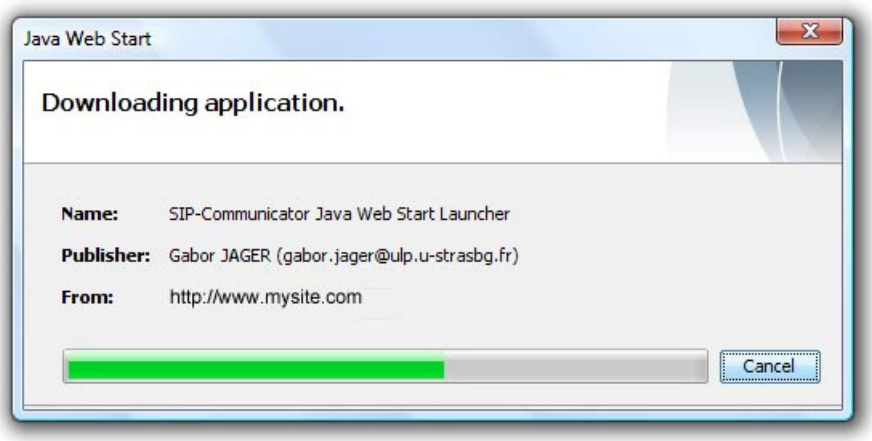

Figure 1 *- Java Web Start progress bar*

When `felix.jar` is downloaded, Java Web Start asks about signature if we used a self signature:

Figure 2 *- Trust signature window*

We have to accept this signature by clicking on "Run". At this point Felix is activated and starting to download all bundles. In the case of SIP Communicator it is more than 30 Mbyte.

During the bundle download users couldn't see anything on the screen: every window is closed. It seems to nothing happened. They will continue their work and in 3-5 minutes when Felix has downloaded all bundles, SIP Communicator starts. So it is NOT a user friendly solution. We can't use this.

As developer we can follow the download process if we check the directory defined as `felix.cache.profiledir` in `felix.client.run.properties`. Generally, we can find this in the user's directory:

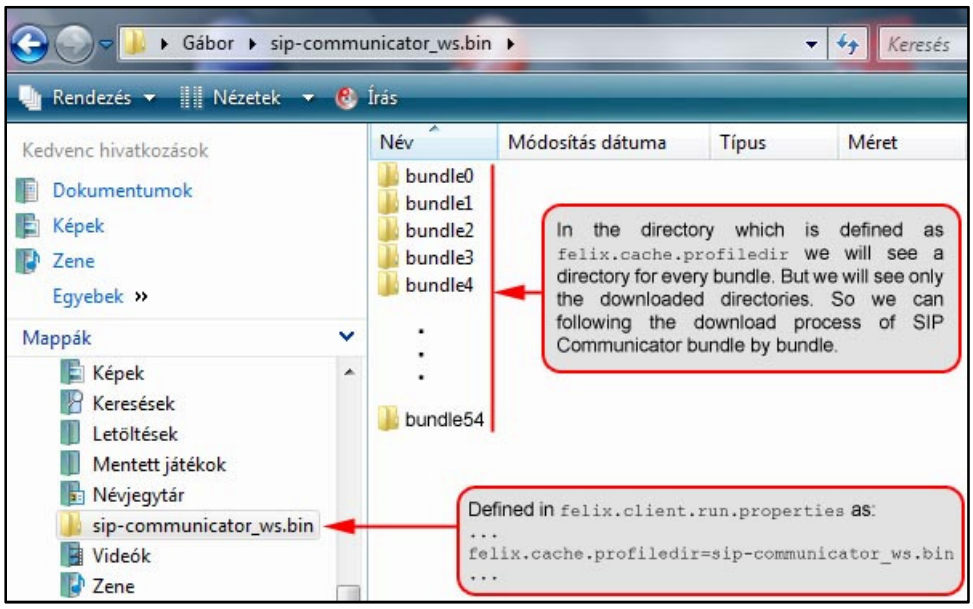

Figure 3 *- felix.cache.profiledir during the download*

We can follow the download process only for the first time, because if we launch SIP Communicator for a second time, directories won't be deleted. Felix updates the downloaded

bundles, then SIP Communicator launches. So for the second time (if we didn't turn off the computer and Felix had downloaded all bundles) SIP Communicator launches almost immediately.

The problems with this solution are:

- We have no control over Felix during bundle download (no stop, no pause, no restart, we can't follow the download process).
- Users have no information about what is happening.

4. Making progress bar for Felix (SIP Communicator)

In this final section, let's see how we can write a progress bar for bundle downloading. We will create a progress window and embed Felix to the `net.java.sip.communicator.launcher` package which is actually used to launch SIP Communicator.

We have to modify a little bit the last JNLP file for using our SIP launcher:

```
...
      <jar href="sip/lib/felix.jar" />
...
   <application-desc main-class="net.java.sip.communicator.launcher.SIPCommunicatorWS"/>
...
```

Now, we will create

- a server side script (PHP) for counting bundles,
- and four Java classes:
  - `WebStartLoader` – for the progress bar
  - `WebStartThread` – for following the download process
  - `WebStartCleaner` – for deleting old bundles
  - `SIPCommunicatorWS` – for embedding and launching Felix

4.1 Server side script

We need a server side script which count files in the bundle directory on the server (for SIP Communicator this is `sip/sc-bundles` directory). We have to upload our script to `sip/lib` directory. For this script we can use PHP for example:

```
<?php

//define the path as relative
$path = "../sc-bundles";

//using the opendir function
$dir_handle = @opendir($path) or die("Unable to open $path");

$num = 0;

// loop
while ($file = readdir($dir_handle)) {
   if ($file != "." && $file != "..") $num++;
}

echo "sip.bundles=$num";

//closing the directory
closedir($dir_handle);

?>
```

4.2 WebStartLoader class

Now we have to create `WebStartLoader` class which is a `JFrame` with a `JLabel`, a `JButton` and a `JProgressBar`. Something like this:

Figure 4 *- SIP Communicator bundle download progress bar*

In this class we need to add some functions:

```
package net.java.sip.communicator.launcher;

import java.io.File;
import org.apache.felix.framework.*;

public class WebStartLoader extends javax.swing.JFrame {

       // window components
       private javax.swing.JButton       btn_cancel;
       private javax.swing.JLabel        lbl_text;
       private javax.swing.JProgressBar  prg_download;

       // local variables
       private Felix           m_felix;            // Felix
       private WebStartThread  th;                 // Thread for following download process

       public WebStartLoader() { ... }             // Constructor
                                                   //   - deleting old bundles
                                                   //   - creating and running WebStartThread
                                                   //   - initialize JFrame

       public void setFelix(Felix f) { ... }       // Set Felix
                                                   //   - saving Felix to m_felix variable

       public void writeLabel(String text) { ... } // Writing label (lbl_text) on the JFrame

       public void setMax(int max) { ... }         // Set maximum value of progress bar

       public void setValue(int val) { ... }       // Set actual value of progress bar

       private void closing() { ... }              // Before closing SIP web starter
                                                   // called by Cancel or window close button
       ...
}
```

As we have no control over Felix's bundle download, we have to kill Felix when we click on cancel button. That's why we have to delete old bundles before we start to download. If we wouldn't delete all old bundles, Felix can't continue downloading the broken bundle.

```
...
private void closing() {

       try {
              m_felix.stop();       // Stop felix immediately
       } catch (Exception e) {
              e.printStackTrace();
       }

       this.dispose();              // Close window
       System.exit(0);              // Exit
}
...
```

4.3 WebStartThread class

This class will follow the download progress.

```
package net.java.sip.communicator.launcher;

import java.io.*;
import java.net.URL;
import java.util.Properties;

public class WebStartThread extends Thread {

        private String dir;                                     // Full path to Felix's profile
                                                                // directory locally

        private WebStartLoader loader;                          // WebStartLoader window
        private int totalBundles;                               // Total bundle count

        public WebStartThread(WebStartLoader loader) { ... }// Constructor
                                                                //   - getting profiledir from
                                                                //     properties file
                                                                //   - retreive bundle count from
                                                                //     online script
                                                                //   - set max bundle to progress
                                                                //     bar's maximum value

        private String getBundleFilename(int num) { ... }   // Get bundle name from
                                                                // bundle.location file (on local
                                                                // directory)

        private long getBundleFileSize(int num) { ... }     // Get local downloaded bundle
                                                                // part file size (in KByte)

        public String getDirectory() { ... }                // profildir of Felix
                                                                // return the saved value

        public String getLocalDirectory() { ... }           // Get felix.cache.profiledir value
                                                                // from properties file

        public void run() { ... }                           // Loop for following download
}
```

In the constructor we have to get the full path to Felix's profile directory. To do this we use `getLocalDirectory()` function:

```
public String getLocalDirectory() {
        ...
        Properties felixProps = new Properties();

        URL url = new URL(System.getProperty("felix.config.properties"));

        InputStream is = url.openStream();
        felixProps.load(is);
        is.close();

        return new File(".").getCanonicalPath() + File.separatorChar
                              + felixProps.getProperty("felix.cache.profiledir");
        ...
}
```

Also in the constructor we need the total bundle count. For getting this value we have to use our online script:

```
...
// retreive bundle count from online script
Properties bundleProp = new Properties();

// as the online script (PHP) must be in the same directory as
// the felix.client.run.properties file, we can make our script URL
// by a simple substring and a concatenation
//    step 1 - http://www.mysite.com/sip/lib/felix.client.run.properties
//    step 2 - http://www.mysite.com/sip/lib/
//    step 3 - http://www.mysite.com/sip/lib/bundleNum.php
String urlString = System.getProperty("felix.config.properties")
                     .substring(0, System.getProperty("felix.config.properties")
                     .lastIndexOf('/') + 1) + "bundleNum.php";

url = new URL(urlString);

InputStream is = url.openStream();
bundleProp.load(is);
is.close();

// save total bundle count to variable (and to the progress bar max value)
totalBundles = Integer.parseInt(bundleProp.getProperty("sip.bundles").trim()) + 1;
loader.setMax(totalBundles);
...
```

Before writing our `run()` function which is the most important part of this class, let's see how we can get some more information about the bundle which is downloading at a given moment. For these information we will use only the local file system.

As I wrote before, Felix can't continue the broken bundle. (We can break a bundle when we click on "Cancel" button or we close `JFrame`.) So the simplest way to resolve this problem if we delete all downloaded bundle and we restart the whole downloading process before we launch the download process. If we use this method always the last bundle's directory will be in download process.

Before downloading a JAR bundle file, Felix will create a container directory named to `bundleX`. Where X is the next bundle number from 0 to total bundle count. The `bundle0` directory is Felix's id container. In the following `bundleX` directories we can find the following file system:

```
...\bundle1:
       version0.0            <dir>
       bundle.id
       bundle.lastmodified
       bundle.location
       bundle.startlevel
```

In the `version0.0` directory we will find the downloaded part of JAR file named to `bundle.jar`. The `bundle.location` file contain the URL to the JAR file on the server. Par example:

```
http://www.mysite.com/sip/sc-bundles/browserlauncher.jar
```

We will use `getBundleFilename()` to retrieve JAR filename locally.

```
// Get bundle name from bundle.location file (on local computer)
private String getBundleFilename(int num) {

        URL url;

        try {
                // (local) path to bundle.location file
                File f = new File(dir + File.separatorChar + "bundle" + num +
                                                File.separatorChar + "bundle.location");

                if (f.exists()) {
                        // get the JAR file's URL
                        BufferedReader input =  new BufferedReader(new FileReader(f));
                        url = new URL(input.readLine().trim());
                        input.close();
                        String ret = url.getFile();

                        // return only the file name
                        return ret.substring(ret.lastIndexOf('/') + 1, ret.length());
                } else {
                        return "Error";
                }
        } catch (Exception e) {
                return "Error";
        }
}
```

When a bundle has a big size we may want to give detailed information about download process on our `JFrame`. For this we need a function which is capable to check the `bundle.jar` file's size:

```
// Get local downloaded bundle part file size (in KByte)
private long getBundleFileSize(int num) {
        ...
        // return bundle.jar size in KB
        File f = new File(dir + File.separatorChar + "bundle" + num + File.separatorChar +
                                "version0.0" + File.separatorChar + "bundle.jar");
        return f.length() / 1024;
}
```

Finally, we can write our `run()` function. In this function we have to do a loop until all bundle downloaded. In the loop, periodically we have to check which bundle is in download process and what size of this bundle is downloaded:

```
public void run() {
	try {
		File f = new File(dir);
		int act = 0;

		// wait until directory is created
		while (!f.exists()) {
			sleep(1000);
		}

		// check bundles in directory
		while (act < totalBundles) {

			// wait
			sleep(500);

			// get actual bundle number in the felix.cache.profiledir directory
			act = f.list().length;

			// print to JFrame
			loader.setValue(act);
			loader.writeLabel("Downloading " + getBundleFilename(act-1) + " (" +
							getBundleFileSize(act-1) + " KByte)");
		}
	} catch (Exception e) {
		e.printStackTrace();
	}
}
```

4.4 WebStartCleaner class

This class contains only a static recursive function to delete a directory with all the sub-directory and files. (Actually, we don't need to put it to a different class.) We will use this static function before launching the download process to delete `felix.cache.profiledir` directory from the local file system. We can call it (for example) in the constructor of `WebStartLoader`.

```
package net.java.sip.communicator.launcher;

import java.io.File;

// Class to delete directory with all subdirectories
public class WebStartCleaner {

	// Deletes all files and subdirectories under directory
	public static boolean deleteDir(File dir) {
		if (dir.isDirectory()) {
			String[] children = dir.list();
			for (int i = 0; i < children.length; i++) {
				boolean success = deleteDir(new File(dir, children[i]));
				if (!success) {
					return false;
				}
			}
		}

		// The directory is now empty so delete it
		return dir.delete();
	}
}
```

4.5 SIPCommunicatorWS class

In this class we will launch the whole download process. To do this we have to create our `JFrame` and then we can launch our embedded Felix. (For embedding Felix you can find more information on http://felix.apache.org/site/launching-and-embedding-apache-felix.html.)

```
package net.java.sip.communicator.launcher;

import java.awt.Toolkit;
import java.util.*;

import org.apache.felix.main.*;
import org.apache.felix.framework.*;
import org.apache.felix.framework.util.*;

// Main class to run SIP-Communicator from Java WebStart
public class SIPCommunicatorWS {

	public static void main(String[] argv) throws Exception {

		// create loader frame
		WebStartLoader loader = new WebStartLoader();

		// Load system properties.
		Main.loadSystemProperties();

		// Read configuration properties.
		Properties configProps = Main.loadConfigProperties();

		// Copy framework properties from the system properties.
		Main.copySystemProperties(configProps);

		try
		{
			// Create a list for custom framework activators and
			// add an instance of the auto-activator it for processing
			// auto-install and auto-start properties.
			List list = new ArrayList();
			list.add(new AutoActivator(configProps));

			// Create a case-insensitive property map.
			Map configMap = new StringMap(configProps, false);

			// Create an instance of the framework.
			Felix m_felix = new Felix(configMap, list);

			loader.setFelix(m_felix);
			loader.setVisible(true);
			loader.setLocation(
					Toolkit.getDefaultToolkit().getScreenSize().width/2
					- loader.getWidth()/2,
					Toolkit.getDefaultToolkit().getScreenSize().height/2
					- loader.getHeight()/2);

			m_felix.start();

			// close loader window
			loader.dispose();
		} catch (Exception ex) {
			System.err.println("Could not create framework: " + ex);
			ex.printStackTrace();
			System.exit(-1);
		}
	}
}
```

For us the most important thing is that Felix start to download bundles when we call the `start()` function. This start function block our main until Felix has downloaded all bundles. But this is no more problem for us because we have a Thread and a Window and they follow Felix's work.