JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2006**

Vol. 5. No. 8, November-December 2006

The Initium RJS Screensaver: Part 4, Automatic Deployment

By **Douglas A. Lyon** and **Francisco Castellanos**

Abstract

The Intium RJS System makes use of screensavers to perform CPU scavenging for grid computing. This article shows how to automate the installation of the IRJS screensaver. The manual installation of the IRJS screensaver, presented in parts 1 and 2 of this series, has been automated to ease installation. A Java Web Start application has been created to complete the installation of the screensaver for Windows and Unix/Linux platform. This application performs an operating system identification and proceeds to download, install, and configure the screensaver files. The installation of the screensaver is a function of the OS, due to differences in screensavers support. Our three previous papers showed how to create screensavers in Java for Windows, Linux and Mac operating systems.

1 INTRODUCTION

This article presents a deployment method for the Initium RJS screensaver. The screensaver is based on the *SaverBeans* [Saverbeans] framework, which has its roots in the JDIC project (**JD**esktop **I**ntegration **C**omponents). The JDIC mission is to enable seamless desktop/Java integration [JDIC1]. The kit is available from [JDIC2] as an open-source distribution.

The deployment process is a part of the overall project that aims to utilize a cross-platform screensaver as a launching facility for a Compute Server (CS), improving grid integration. In parts 1, 2, and 3 of this series, we described the process of building, compiling, and deploying a Saverbeans screensaver in Unix, Windows and the Macintosh. This paper describes the automation of deployment for the IRJS screensaver in the Windows and Unix platforms. The automation of deployment for the Macintosh platform is described in part 3 of this series.

Cite this column as follows: Douglas A. Lyon and Francisco Castellanos “The Initium RJS Screensaver: Part 4, Automatic Deployment”, in *Journal of Object Technology*, vol. 5. no. 8, November - December 2006, pp. 31-40 http://www.jot.fm/issues/issue_2006_11/column4

2 ARCHITECTURE

We use Java Web Start technology to deploy the screensaver. The application is downloaded from a web server. A user volunteers a computer into the grid by running a web start application. The application checks the platform type and sets the location where the resources reside. These include the location of screensaver support files. The web start application beams the resource files into the computer from a web server, updating them, automatically, when needed. The files are decompressed and verified before they are installed. Lastly, the web start application configures the screensaver according to OS requirements.

Figure 2-1 shows an overview of the events, and the parties involved, in first deploying and installing the screensaver and second in volunteering the user's computer to a grid. This article addresses events 1 and 2 in Figure 2-1.

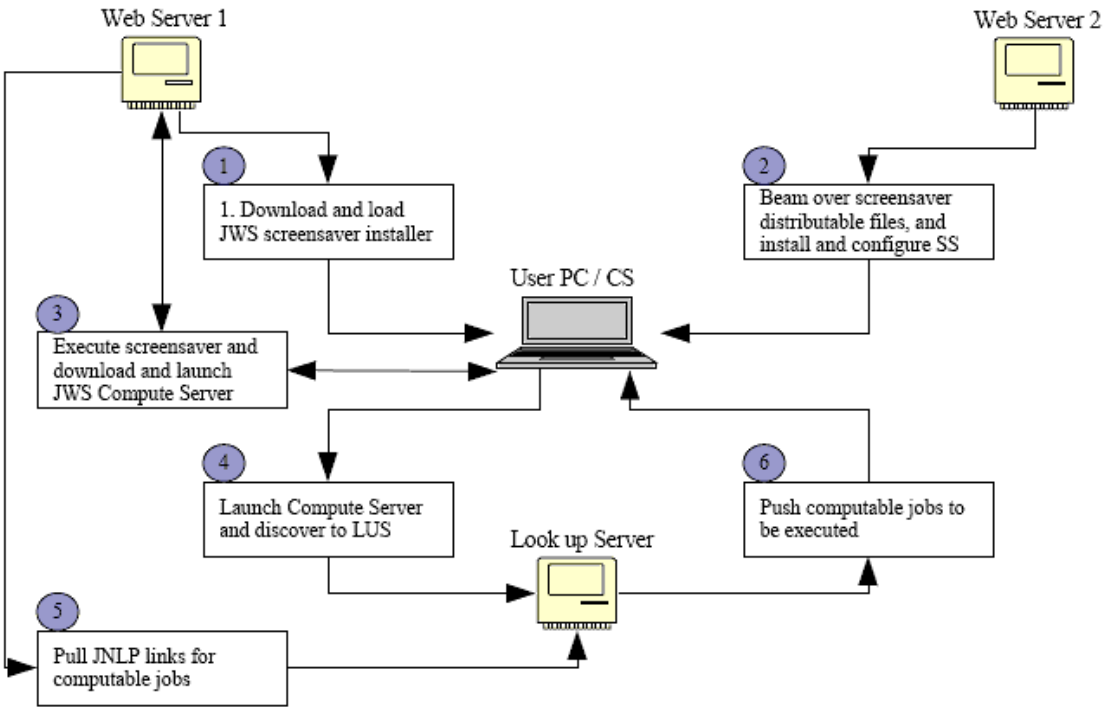
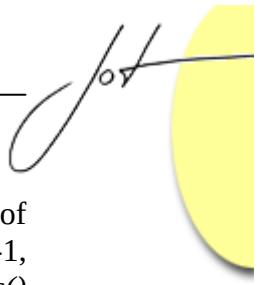


Figure 2-1. A diagram of part of the IRJS System Architecture

3 OPERATING SYSTEM IDENTIFICATION

Identification of the user's operating system is an important step in the installation process, due to the differences between the operating systems in supporting screensavers. Different flavors of the Windows OS have different ways of supporting screensavers. In



particular, the screensaver files belong to different directories depending on the version of Windows. The web start application uses some utility methods, as shown in Example 3-1, to help in the task of identifying the OS. These methods utilize the *System.getProperties()* method from the Java API which determines the system properties and returns a *Properties* object.

Example 3-1

```
public static String getOsName() {
        Properties prop = System.getProperties();
        return prop.getProperty("os.name");
}
public static boolean isWindows(String str) {
        if (isWindows()) {
            String os = getOsName().toLowerCase();
            if (os.indexOf(str) > -1) return true;
        }
        return false;
}
public static boolean isOsPrefix(final String prefix) {
        String os = getOsName();
        return os != null &&
os.toLowerCase().startsWith(prefix) ?
                true :
                false;
 }
public static boolean isWindowsXp() {
        return isWindows("xp");
 }
…
public static boolean isLinux() {
        return isOsPrefix("linux");
 }
```

4 BEAM OVER AND DECOMPRESSION

A file-transfer process, called “beam over” enables the transfer of native resources into critical file areas. The web start application beams over the IRJS screensaver files, based on the OS identified. Beam over and decompression takes care of the job of downloading needed resources, and placing them where appropriate. Resources needed, and their location, are a function of the OS, as shown in Example 4-1.

Example 4-1

```
properties.put(SS_WIN_URL,
"http://www.myjavaserver.com/~fsophisco/thesis/libs/screensave
r/win/rjssaverjar.jar");

properties.put(SS_UNIX_URL,
"http://www.myjavaserver.com/~fsophisco/thesis/libs/screensave
r/unix/rjssaverjar.jar");
```

Example 4-2 shows the location where files will be installed.

Example 4-2

```
windowsSystemDir = "";
if (OsUtils.isWindows98()){
   windowsSystemDir = "C:"      + File.separator +
                "windows" + File.separator +
                "system"  +  File.separator;
   }
else if (OsUtils.isWindowsNt()){
   windowsSystemDir = "C:"      + File.separator +
              "winnt"         + File.separator +
             "system"       + File.separator; }
```

File beam over enables downloads on demand. This simplifies resource bundling and lowers average download time (since only the resources that are needed are downloaded). This is shown in Example 4-3.

Example 4-3

```
public static void downloadScreenSaverJar(File outputJarFile,
String urlStr) throws IOException {
   URL screenSaverUrl = getResourceUrl(urlStr);
   UrlUtils.getUrl(screenSaverUrl, outputJarFile);
}
…
//Beam over jar file for Windows
outputJarFile = new File (windowsSystemDir + File.separator +
SSInstallerUtil.getJarName() + ".jar");
String urlString = SSInstallerUtil.getSSWinUrl();
try{
   SSInstallerUtil.downloadScreenSaverJar(outputJarFile,
urlString);
}catch(Exception e){
   return "Error Downloading Jar File: " + e.toString();
}
return "";
…
```

```
//Beam over jar file for Unix
outputJarFile = new File (ssHome + File.separator +
SSInstallerUtil.getJarName() + ".jar");
String urlString = SSInstallerUtil.getSSUnixUrl();
try{
   SSInstallerUtil.downloadScreenSaverJar(outputJarFile,
urlString);
}catch(Exception e){
   return "Error Downloading Jar File: " + e.toString();
}
return "";
```

Once the jar files are downloaded, the web start application uncompresses them as shown in example 4-4.

Example 4-4

```
public static void uncompressScreenSaverJar(File jarFile){
    Unzipper.uncompressJarFile(jarFile);
    jarFile.deleteOnExit();
}
…
public String uncompressFiles(){
try{
   SSInstallerUtil.uncompressScreenSaverJar(outputJarFile);
}catch(Exception e){
        return "Error uncompressing Jar File: " +
e.toString();
} return "";
}
```

5 CONFIGURATION

Up until this point, deployment has been platform independent. However, differences in how screensavers are supported make the configuration task non portable. For example, the screensaver files for the Windows OS must be placed in the directory *system* or *system32* under the *Windows* directory. These directories are part of the OS itself; therefore administrators may choose to protect them from being modified. Due to this type of constraint, we require that the installer have privileges to write to these directories.

For Windows screensaver configuration, the web start application places the screensaver files in version sensitive directories. For example:

For Windows 98: C:/windows/system/.

For Windows NT: C:/winnt/system/.

For other version: C:/windows/sytem32/.

The screensaver file-package contains a file with extension .scr E.g.<scrensaverName>.SCR. This file allows the screensaver to be visible on the Display Properties screen next time it is opened, as shown in Figure 5-1. After the screensaver installation is complete there is a manual step that the user must accomplish. He/She must select the *rjsssaver* from the screensaver list, as shown in Figure 5-1, and set the time to "wait before launching". This step completes the IRJS screensaver configuration for the Windows platform.

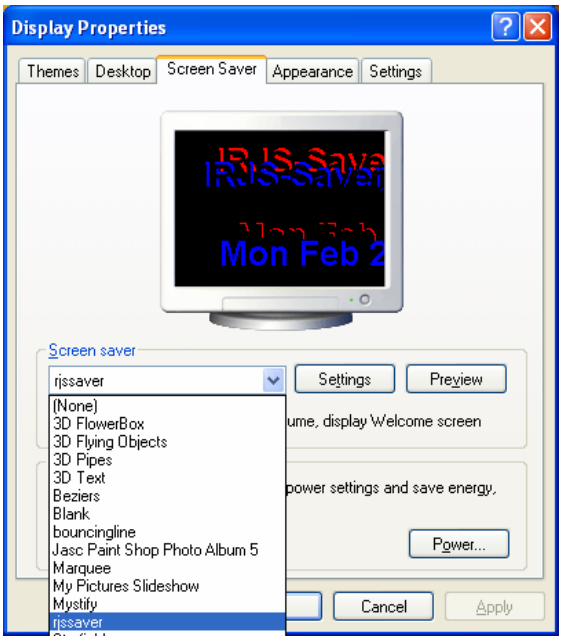


Figure 5-1 Display properties-user interface for the Windows platform.

In Linux/Unix OS, the IRJS screensaver configuration is a little more challenging. The [Saverbeans] SDK relies on a software called Xscreensaver; which is often part of the OS installation package. The **Xscreensaver** program waits until the keyboard and mouse have been idle, and then runs a graphics demo. It turns off as soon as there is any mouse or keyboard activity [Zawinski]. **Xscreensaver** consists of two parts: a driver or daemon that detects idleness and does locking, and the many graphics demos that are launched by Xscreensaver [Zawinski]. To learn more about this package visit http://www.jwz.org/xscreensaver/.

The first part of the configuration happens in the beam over and decompression task. There are no strict requirements that dictate the location of screensavers in the file system. Therefore, for convenience and to prevent any access issues the web start application installs the screensaver files in the user home in the directory <userHome>/ss/<screensaverName> as shown in Example 5-1.

Example 5-1

```
public String createSSHome(){
   String ssloc =  SSInstallerUtil.getUserHome() +
File.separator
                          + SSInstallerUtil.getSSHome();
   ssHome = SSInstallerUtil.getUserHome() + File.separator
             + SSInstallerUtil.getSSHome() + File.separator
             + SSInstallerUtil.getSSName();
   try{
        SSInstallerUtil.createDir(ssloc);
        SSInstallerUtil.createDir(ssHome);
   }catch (Exception e){
        return "Error creating ss home: " + e.toString();
   }
   return "";
}
```

In Unix/Linux the Xscreensaver application uses a file named *.xscreensaver*, located at the user home. This file lists user properties, active screensavers and their location. The *.xscreensaver* file is created automatically the first time that the client application (xscreensaver-demo) of the Xscreensaver is executed.

The web start application modifies the *.xscreensaver* file to make the IRJS screensaver available. In particular the IRJS screensaver must be included and must be selected as the only active screensaver. The web start application reads the contents of the *.xscreensaver* file, makes modifications where necessary, and rewrites the file as shown in Example 5-2.

Example 5-2

```
//mode: one -> one screensaver working
sloc = sline.indexOf("mode:");
if (sloc > -1){
   lines.add("mode:\tone");
   sline = raf.readLine();
   continue;
}

//Once it finds the string "programs:" insert the line with
the SS info into the List.
if (sloc > -1){
lines.add("\t\" " + SSInstallerUtil.getSSName() + " (java)\""
+ ssHome
    + File.separator + SSInstallerUtil.getSSName() + " -root -
jdkhome " + jhome +" \\n\\");
}
```

To complete the screensaver configuration in Unix/Linux platform, the web start application grants executable privileges to the screensaver executables. This task is necessary because when files are archived into a jar file and at later time extracted, they do not retain file permissions. To overcome this issue, the web start application executes a small shell script using the Runtime Java API as shown in Example 5-3.

Example 5-3

```
Script:
LOC=$HOME/ss/rjssaver
chmod +x $LOC/rjssaver $LOC/rjssaver-bin

public String changeFilePrivs(){
   String[] pc = new String[2];
   int exitValue = 0;
   //Grant executable permissions
   try{  pc[0]= "sh";
         pc[1]= ssHome + File.separator + scriptName;
         p = Runtime.getRuntime().exec(pc);
         exitValue = p.waitFor();
   }catch(Exception e){
         return "Error granting privs Exit Value: " + exitValue
+ e.toString();
   }
   if (exitValue != 0)
         return "Error granting privs Exit Value: " +
exitValue;
   return "";
}
```

Lastly, after the screensaver installation is completed there is one manual step remaining. The user must execute the program xscreensaver-demo, so the *.xscreensaver* file is read and the IRJS screensaver is recognized as one of the screensavers. This step completes the IRJS screensaver configuration for the Unix/Linux platform.

6 INTERFACE

The web start application uses a small interface to indicate the steps of the installation. Each step will be displayed in the interface at completion, as shown Figure 6-1 and Figure 6-2. In case of a failure the interface describes the failed step and the exception caught.

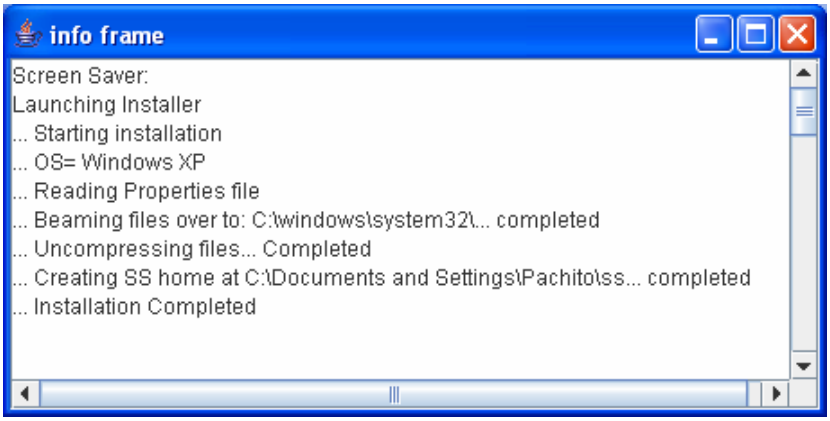


Figure 6-1 Installer interface indicating the installation steps for the Windows platform.

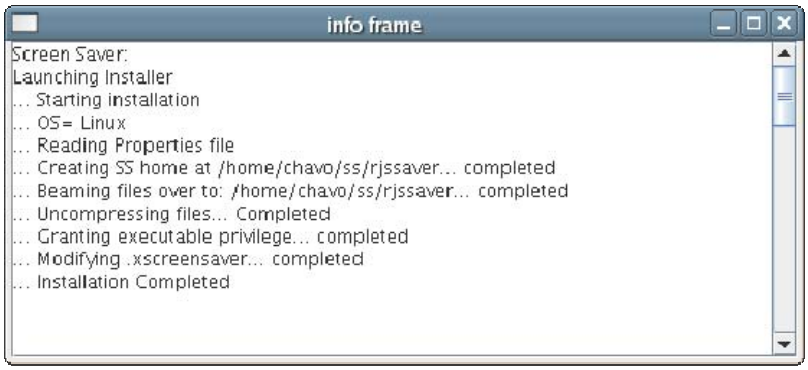


Figure 6-2 Installer interface indicating the installation steps for the Linux platform.

7 SUMMARY

This paper described the process followed to automate the deployment of the IRJS screensaver for the Unix/Linux and Windows platforms. The automation of the screensaver deployment is accomplished in a few steps. In the first step, the system identifies the operating system of the user. In the second step, a file-transfer process called beam over is used to download the appropriate IRJS screensaver files, according to the OS identified. Lastly, the files are installed and configured based on the requirements of each OS. We use Java Web Start technology to execute the installation and deploy the IRJS screensaver to the user's computer.

REFERENCE

[JDIC1] Java.net : "JDIC project home", https://jdic.dev.java.net/ Last accessed March 14, 2005.

[JDIC2] https://jdic.dev.java.net/documentation/incubator/screensaver/index.html Last accessed March 14, 2005.

[SaverBeans] https://jdic.dev.java.net/documentation/incubator/screensaver/index.html Last accessed March 14, 2005.

[Zawinski] Jamie Zawinski: "A screen saver and locker for the X Window System" http://www.jwz.org/xscreensaver/

About the authors

After receiving his Ph.D. from Rensselaer Polytechnic Institute, **Dr. Lyon** worked at AT&T Bell Laboratories. He has also worked for the Jet Propulsion Laboratory at the California Institute of Technology. He is currently the Chairman of the Computer Engineering Department at Fairfield University, a senior member of the IEEE and President of DocJava, Inc., a consulting firm in Connecticut. E-mail Dr. Lyon at Lyon@DocJava.com. His website is http://www.DocJava.com.

Francisco Castellanos earned his bachelors degree with honors in Computer Science at Western Connecticut State University. Francisco Castellanos worked at Pepsi Bottling Group in Somers, NY as a software developer. Currently he is working on a thesis to complete his Master's Degree in Computer Engineering from Fairfield University. His research interests include grid computing. Francisco Castellanos is also employed by Access Worldwide in Boca Raton, FL as a software developer. He can be contacted at fsophisco@yahoo.com.