Truly Platform Independent Java Applets

How to integrate image resources into Java applications and applets and run them on any operating system or browser

Abstract

One of the most important reasons why developers chose Java to program an application is Java's platform independence. A typical workflow consists of 1) writing your program, 2) packing it into an executable JAR file, and 3) execute it on any operating system as application and on any browser as Java applet. Even though this is not very complicated, there are a few details that must be precisely met and otherwise result in exceptions that are tedious to fix. In particular, if you want to include resources like images within your program. This guide explains how to get from your Eclipse project to the final applet deployment as JAR file with strict XHMTL code.

Table of Contents

1 Basic Terminology

Platform independence: Platform independence is one of the most important aspects why a developer would chose Java over another language. In contrast to languages like C/C++, compiled Java code runs on any system on which the Java Runtime Environment is installed. This includes Windows, Linux, Mac OS X, Solaris and any other crazy platform you might come up with.

Java Applet: Java provides support for web integration in form of applets. An applet can be embedded within a website and runs on the client system. Due to platform independence, any browser that supports Java is able to display Java applet. This includes Internet Explorer, Mozilla Firefox, Google Chrome, Safari, Opera, and so on.

JAR file: A JAR file is typically used to deploy a Java program. If you are a Windows user you can think of it as an .exe file. However, unlike .exe files, it can be executed on every platform that supports Java. There are other ways to ship your Java application like Java Web Start (JNLP)[1] but this is for another paper.

Eclipse: Eclipse[2] is an integrated development environment (IDE) that supports programming Java (and much more). Eclipse highlights keywords in the source code, underlines compiler errors and warnings, manages your project structure and is extensible by numerous plugins. This guide concentrates on the development process in Eclipse. There are other IDEs like NetBeans[3] or BlueJ[4].

2 How you want to develop your Java program

Let's start with what you don't want: Programming your application and compressing it into a JAR file which then does not work on a Linux system or as applet within Internet Explorer and to make matter worse, each time due to another exception that you can only read out from the console. In principle, your program logic may be just fine but the errors come from subtle technicalities such as the usage of a java.io.File for a resource in a jar-container or using the wrong format to describe the path to your image resource.

Instead, you want to program your code, test it, compress it to a JAR file and run it on any Windows, Linux, Apple iOS, Solaris, Internet Explorer, Mozilla Firefox, Google Chrome, Safari, Opera or your refrigerator if it only supports Java. Finally, you want to embed your applet using valid HTML code that even meets the strict XHTML requirements.

3 Making your project truly platform independent

The following five steps give you a guidance to make your Java application truly platform independent. By now, I use this procedure in each of my Java projects and it has always been successful. I am not saying this is a perfectly elegant way but the most elegant one I have found so far.

---

[1] http://docs.oracle.com/javase/6/docs/technotes/guides/javaws/
[2] http://www.eclipse.org/
[3] http://netbeans.org/
[4] http://www.bluej.org/

3.1 Project Structure

A typical Java project is divided into four directories:

- src/ Source code divided into packages
- bin/ Compiled bytecode
- lib/ Integrated libraries
- ress/ The resources of your project, e.g. images, sound, other media

Start with adding two nested folder called “ress” to your project (Figure 1): Right click on your project -> New -> Folder. Include the outer folder to the build path: Right click onto the outer resource folder -> Build path -> Use as source folder. Your resources are now included in the project.

Having two “ress” folder is not great but at least it works!

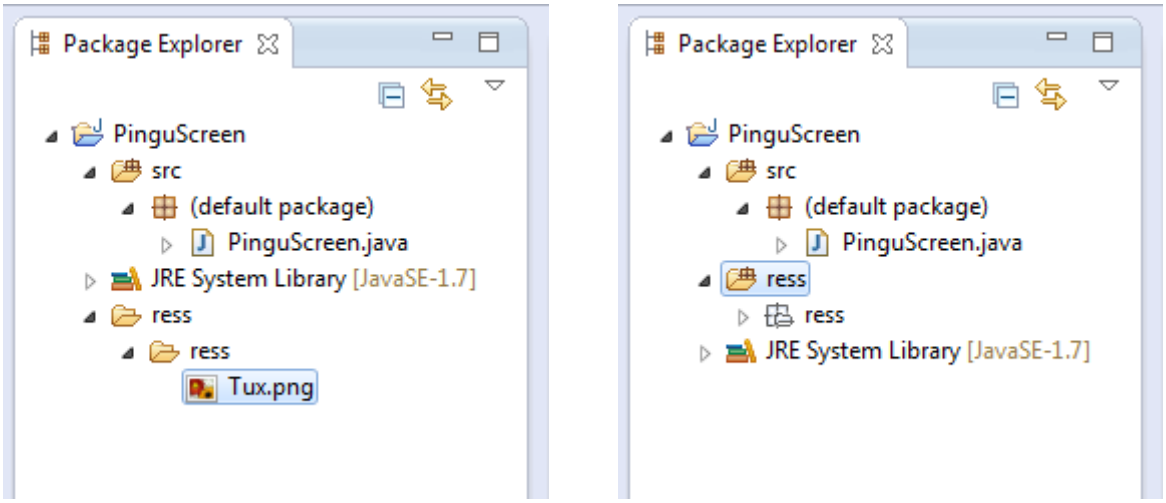

Figure 1: Add two nested „ress“ folders and put your images within (left). Make the outer resource folder a source folder (right).

Later, Eclipse will only export the inner “ress” folder to the jar. With only one “ress” folder, only the content will be exported and the class loader will not be able to find the resources which’s path leads into a “ress” folder.

3.2 Loading Resources, e.g. Images

There are many ways to load images into your program. Usually, you end up storing the image as a variable of type java.awt.image.BufferedImage. The way you load the image is crucial to achieve platform independence. It might be tempting to use a java.io.File and then use ImageIO.read(file). This, however, will be problematic when you have compressed your program to a JAR file that contains the image. Your JAR file already is a file and it is not possible to reference a file within a file. Thus, this approach will be broken for a JAR file, while it was always fine while you were hacking and launching your code in Eclipse.

Instead, you should use a URL. A URL can describe the path to a resource within a JAR file as well as to a location on your computer or the internet. In this guide, I use a class called PinguPanel for demonstration purposes. Use the following code to load your BufferedImage in the fictional class PinguPanel.

```
	private BufferedImage loadImage(String path) {
		try {
			URL url = PinguPanel.class.getResource(path);
			BufferedImage image = ImageIO.read(url);
			return image;
		} catch (IOException e) {
			throw new RuntimeException(e);
		}
	}
```

The method loadImage() first converts the specified path to a URL that references your resource. If this translation is not possible for some reason (for instance if the resource is not there) it returns null (instead of throwing and exception). Second, ImageIO reads the image from the URL and returns the BufferedImage. One might consider using BufferedImage's superclass java.awt.Image instead of BufferedImage since it looks a little cleaner. However, BufferedImage provides many useful methods that Image does not.

The path format is imperative. It must have a leading slash and "ress" should only appear once, e.g. *"/ress/Tux.png"*. In a Linux system, the leading slash describes the root folder but Class.getResource will not interpret it as such but find your image within your JAR file. Without the slash, it will not find it.

Note: If you change the files in your "ress" folder, you have to refresh your Eclipse project. Just right click your project and click "Refresh" or press F5. The changes then will be integrated into the binary folder.

3.3 Design your program as application and applet

You certainly think about how your users are going to access the masterpiece you are working on. You may be happy with only providing an applet so users need to go to your website or your program might only be viable as heavyweight application. In this guide, the class PinguScreen shall serve as entry point of your software whether you start it as application or applet. Applets require a class that inherits from javax.swing.JApplet (Swing) or java.awt.Applet (AWT) and overrides the method init(). An application could use a GUI that is constructed within a javax.swing.JFrame (Swing) or a java.awt.Frame (AWT) and starts in a static main method. SWT[5] is another Java GUI library but not considered here.

The following code snipped shows how to combine the starting points for JFrame applications and JApplets within the same class. If preferred, splitting it up into two separate classes is straight forward. Here, PinguPanel (from above) is the JPanel that shall lie within the JFrame or JApplet respectively.

[5] http://www.eclipse.org/swt/

```
public class PinguScreen extends JApplet {

	// Entry point for Java Application
	public static void main(String[] args) {
		SwingUtilities.invokeLater(new Runnable() {
			public void run() {

				JFrame frame = new JFrame("PinguScreen");
				frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
				frame.add(new PinguPanel());

				frame.pack();
				frame.setLocationRelativeTo(null);
				frame.setVisible(true);
			}
		});
	}

	// Entry point for Java Applet
	@Override
	public void init() {
		JPanel panel = new PinguPanel();
		panel.setBorder(new EtchedBorder());
		add(panel);
		setSize(getPreferredSize());
	}
}
```

Side note: To make this a little more concrete: in a typical project that integrates a graphical user interface (GUI), you probably make use of the Model-View-Controller (MVC) programming pattern[6]. The MVC pattern basically splits up the program into the model component that holds the current state of the program, the view component that graphically represents this state and the controller component that mediates actions from the view, e.g. a mouse click, to the model. A JFrame or JApplet would be part of the view. You probably want to write only one controller to control your program, no matter whether it runs in a JFrame or a JApplet. The greatest common divisor is their common superclass java.awt.Container. A good idea is to give the controller a constructor that expects a container and then inserts its created view. One can then give the JFrame or JApplet as container to the controller. Alternatively, the Controller can expect a JPanel and the entry method of the JFrame or JApplet creates such a JPanel, adds it to the JFrame or JApplet respectively and then calls the constructor of the controller with this JPanel as parameter.

[6] http://www.oracle.com/technetwork/articles/javase/index-142890.html

3.4 Export your Java Program

Now, all paths to your resource files have the correct form and are loaded the correct way. Your program is ready to be deployed. In Eclipse, it is very easy to export your Java program to an executable JAR file.

Click File -> Export. Find Java -> Runnable JAR File and click “Next”. Choose your main class and export destination as shown in figure 2. Choose how to handle libraries as you desire. Click “Finish”.

Your JAR File should be generated. You might be able to double click the JAR file to start it. Otherwise, open a console, move to the directory that contains the JAR file and run: *“java -jar PinguScreen.jar”*. Note: In the JAR file you only find one “ress” folder.

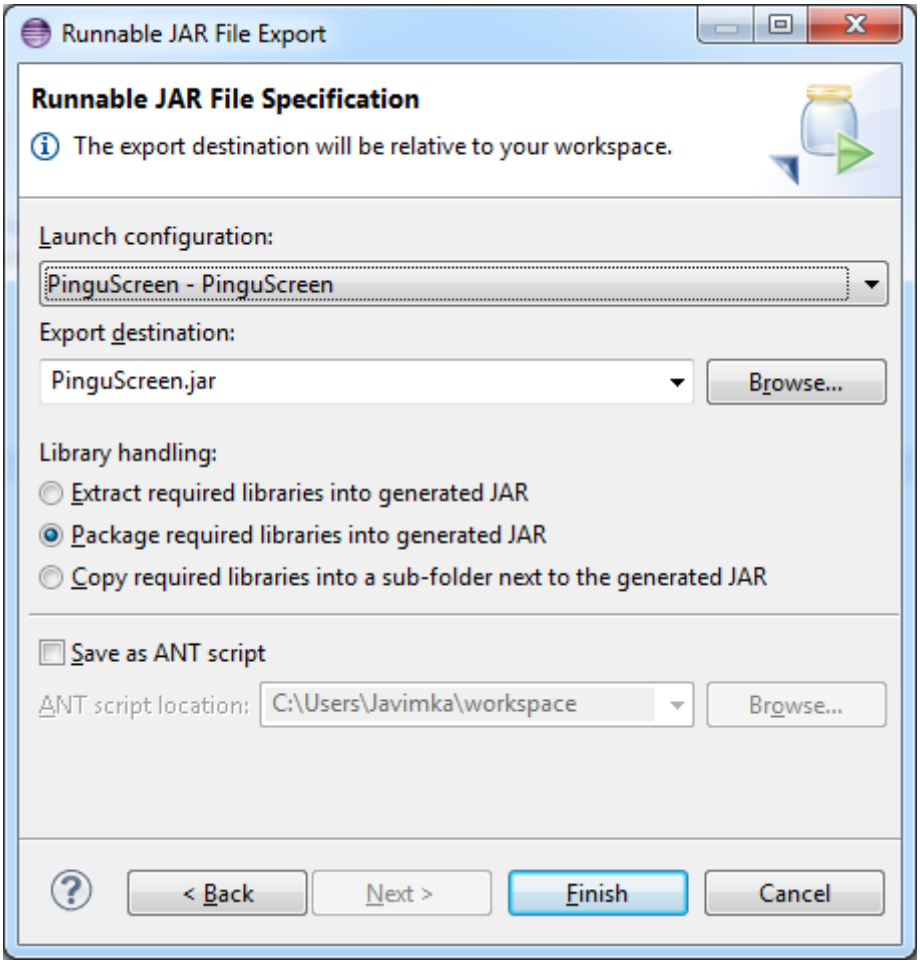

Figure 2: Runnable JAR File Specification

3.5 Embed the applet within a HTML website

In the early days of applets, the HTML tags <applet> </applet> were used to integrate a Java applet into a website. These tags, however, are deprecated by now and replaced by the <object> </object> tags. Due to rigorous backwards compatibility of most browsers, the old method still works even though the HTML code is invalid.

In a perfect world, simple HTML code to embed a Java applet would work for Internet Explorer and Mozilla Firefox. However, the two browsers interpret HTML code slightly different. To make sure it works in both, there is a HTML and a JavaScript solution. This is the HTML version:

```
<!--[if !IE]> Hidden to IE -->
<object classid="java:your.pack.PinguScreen.class"
      type="application/x-java-applet"
      archive="PinguScreen.jar" width="200px" height="235px">
<!--<![endif]-->
      <object classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
            width="200px" height="235px">
            <param name="code" value="your.pack.PinguScreen.class" />
            <param name="archive" value="PinguScreen.jar" />
<p>Here should be a Java Applet. Make sure Java is installed</p>
</object>
<!--[if !IE]> Hidden to IE -->
</object>
```

The comments make sure that the Internet Explorer interprets different HTML code than other browsers. As a result, your applet works in all of them. This code satisfies strict XHTML requirements.

4 Conclusion

Java is platform independent but to truly be able to run a program on any operating system and browser, several details must be precisely met. Use two nested “ress” directories, include them into the build path, prefix paths with a leading slash and only use “ress” once. Translate paths to a URL over the classloader and load the images with ImageIO to store them within a BufferedImage. Define two different entry points for a Java application and a Java applet and give the controller a container for the view. Export your program as JAR file and embed it into your website using modified HTML code for Internet Explorer and other browsers to satisfy strict XHTML.