JIDE Developer Guide for Web Start Application or Applet

Purpose of This Document

Generally speaking, developing an application for deployment with Java Web Start is the same as developing a stand-alone application. Developing an applet is also almost the same as developing an application except *main()* method is replaced by init() as well as other small differences. Those general topics are not the focus of this developer guide.

A special consideration for running applications over the Internet is security. Users are cautious about downloading and running programs on their computers without a guarantee of security to prevent programs deleting files or uploading personal information. Java Web Start addresses this concern by running all untrusted code in a restricted environment called the sandbox. While the application is in the sandbox, Java Web Start can promise that the application cannot compromise the security of local files or files on the network. How to write your application so that it can run as Web Start or Applet with the least required permission, this is exactly what we need to address in this developer guide.

Font

Font is considered as a restricted resource when running as Web Start or Applet. Any call to create a Font will throw AccessControlException. In JIDE products, we have to create Font. For example, under WindowsLookAndFeel, we use Tahoma font instead of using the default system font that Swing uses[1].

Basically you can't call

```
Font font = new Font(…);
```

You have to bundle the font file inside the jar and use ClassLoader to load the font file.

```
ClassLoader cl = this.getClass().getClassLoader();
Font font = Font.createFont(Font.TRUETYPE_FONT,
cl.getResourceAsStream("font_file"));
```

[1] If you still want to use the default font setting, you can run your application with "swing.useSystemFontSettings" set to "true". This probably doesn't make sense if you in English locale as the default font setting looks really bad. However if you are on other locale especially Chinese, Japanese and Korean, you will have to run with the setting to "true" as Tahoma doesn't work with those character sets.

To make it simply for us and for end users, we made two methods, createFont(..) and createFontUIResource(…), in a class called SecurityUtils.java. Instead of writing new Font(…) in your code, call SecurityUtils.createFont(…) instead. In createFont(…) or createFontUIResource(…), we will try to do create font using using new Font. If failed, we will automatically use ClassLoader to load font file and create the font.

Obviously this requires you to bundle the font files in your jar. We provided an easier way to do this too.

Step 1: You need to know exactly what font your application is using. Find those font files first. In you are on Windows, you can go to C:\Windows\Fonts directory to find them.

Step 2: Once you have all the font files, create a “fonts” folder somewhere on your disk and copy all font files under it

Step 3: Create a fontfiles.properties under the “fonts” folder. Below is an example. Basically this is a file that maps from font names to font files. See below.

```
# For loading fonts using ClassLoader
# Key: FontName[_style]
# Style: The style is optional. It could be empty or one of the three values - Bold, Italic, Bold_Italic.
# The key with style has a higher priority than the one without style.
# Value: the path to font file name. The convention is to create a "fonts" package from root and put all font files under it.
# For example:
Tahoma=fonts/tahoma.ttf
Tahoma_Bold=fonts/tahomabd.ttf
# It means use fonts/tahomabd.ttf to create bold tahoma font and fonts/tahoma.ttf to create all other tahoma font.
#
# If the font name has space, use '_' to replace space.
# For example:
Courier_New=fonts/cour.ttf
Courier_New_Bold=fonts/courbd.ttf
Courier_New_Italic=fonts/couri.ttf
Courier_New_Bold_Italic=fonts/courbi.ttf
```

Left part is the font name. It could have a postfix “Bold”, “Italic”, or “Bold_Italic”. If there is no postfix, it means PLAIN font. Please make sure you replace all spaces in font name with ‘_’ as the Courier New example shows.

Right part is the font file.

The order of strings doesn’t matter. The priority is from more specific to more general. So in the Tahoma example above, Tahoma_Bold has a high priority than Tahoma. If a bold Tahoma font is requested, fonts/tahomabd.ttf will be picked. If an italic Tahoma font is requested, our

code will check for key Tahoma_Italic first. Since this key is not there, it will fall back to key Tahoma, so fonts/tahoma.ttf will be picked eventually.

Step 4: If your application supports different locale, you need to localize the fontfiles.properties too. However this is not really a "localize" per se. For example, if you want to use China locale, you just need to copy the fontfiles.properties to fontfiles_zh_CN.properties, include the Chinese font you want to use in "fonts" folder and modify fontfiles_zh_CN.properties to point to those files.

Step 5: Jar the "fonts" folder into one jar and include this jar in your application class path. Or you can simply jar the "fonts" folder into your application jar.

And that's it.

Access System Property

Except the system properties in the table below, System.getProperty(String key) will throw AccessControlException if running as Web Start or Applet without the related permission granted. For the Font, as we just discussed, there is a workaround. Unfortunately there is no workaround to get the property value if it is not allowed. The only solution is to always provide a default value. If exception happens, use the default value.

java.version	os.name	java.specification.version
java.vendor	os.version	java.specification.vendor
java.vendor.url	os.arch	java.specification.name
java.class.version	file.separator	java.vm.specification.vendor
	path.separator	java.vm.specification.name
	line.separator	java.vm.version
		java.vm.vendor
		java.vm.name

So we added a method called getProperty(String key, String defaultValue) into SecurityUtils.java. It's really easy to use. All you need to do is to replacey all System.getProperty(String key) with SecurityUtiles.getProperty(String key, String defaultValue). Remember to always provide a default value that makes sense in your application.

Access Files

You can't access files on the local computer in Web Start or Applet, nor can you access the registry on Windows. It means you can't use the normal way to save the layout used by JIDE Docking Framework, JIDE Action Framework and DocumentPane in JIDE Components.

However, there are still two possible solutions.

Solution one: If your application has a central server, you could persistent each user's layout file on the server. The layout information can be stream in/out using two methods in LayoutPersistence called loadLayoutFrom(InputStream in) and saveLayoutTo(OutputStream

out). JIDE Docking Framework, JIDE Action Framework and DocumentPane all support this way of saving/loading layout.

Solution two: If your application is a standalone Web Start or Applet, the only way to save layout is to grant permission in policy file. There are several ways to provide such a policy file. You can read http://java.sun.com/j2se/1.4.2/docs/guide/security/PolicyFiles.html for more information. No matter how you do it, you will need one of the following two permissions granted.

If you want to use regular file as layout file, you have to grant

```
grant {
      permission java.io.FilePermission "the_layout_file ", "write, read, delete";
};
```

If you want to use javax pref package, you have to grant

```
grant {
      permission java.lang.RuntimePermission "preferences";
};
```

AWTEventListener

If you are not using JIDE Docking Framework, JIDE Action Framework, DocumentPane or JidePopup (including any comboboxes that indirectly using JidePopup), you will be free from all security issues related to JIDE after taking the steps above. If you do use those, there is still a permission you have to set unfortunately. It is an *AWTPermission* called "listenToAllAWTEvents". We use *AWTEvent* listener to do things like drag-n-drop of dockable frame and command bar. That's why we need this permission. So in ".java.policy", you have to have this entry.

```
grant {
      permission java.awt.AWTPermission "listenToAllAWTEvents";
};
```

However we use *AWTEventListener* only for a specific reason, you can use a method called *setAWTEventListenerDisabled* on *SecurityUtils*. It is a global option to disable all *AWTEventListeners* used by JIDE Docking Framework, JIDE Action Framework and DocumentPane. When we don't use *AWTEventListener,* there are some side effects. For example, when you move the mouse away from the *SidePane* button that has active dockable frame, it doesn't auto-hide the active dockable frame.

In fact, if you ever used drag and drop in your application, according to a Drag and Drop Faq at http://www.rockhoppertech.com/java-drag-and-drop-faq.html#appletpolicy, you got to have this permission too. See below for a copy of the link above.

Applets

Can I use DnD with Applets?

Sort of. As of JDK1.2.1 you can drag from an Applet but not drop into one unless you do not create the drop targets in init or start. There is an example in the Rockhopper DnD library.

What sort of Applet security policy is needed?

Here is an example policy:

```
grant {
  permission java.awt.AWTPermission "accessEventQueue";
  permission java.awt.AWTPermission "setDropTarget";
  permission java.awt.AWTPermission "accessClipboard";
  permission java.awt.AWTPermission "acceptDropBetweenAccessControllerContexts";
  permission java.awt.AWTPermission "listenToAllAWTEvents";
};
```

How do I set the applet security policy?

That's really not just a DnD question. This will do it with appletviewer:

```
appletviewer -J-Djava.security.policy=policy index.html
```

Summary

The original goal of this task is to eliminate all the permission requirements that were needed by JIDE. Even though we successfully remove most of them, unfortunately there is one permission requirement we just can't get rid of if you are using Docking Framework or Action Framework or DocumentPane, the listenToAllAWTEvents permission. So if you deploy your application as web start or applet, you need to make sure this permission is granted correctly on your client machine.