How to write a JAS3 Plugin

Overview

JAS3 is based on the Freehep Application Framework, which provides a basic application framework into which extension modules, or "plugins" can be loaded. The application framework provides general purpose functionality, such as top level GUI (windows, menus etc.), help system, storing and retrieving user preferences etc. All of the data analysis specific functionality of JAS3 is provided by a set of extension modules. The functionality of JAS3 can be customized by adding, removing or replacing modules. In order to make this possible, direct interaction between modules is avoided as far as possible, instead modules communicate with each other via service interfaces and notification events. JAS3 comes with a standard set of service interfaces and notification events, but new modules can also define their own interfaces and events.

Writing a JAS3 Plugin

This document shows how to develop an extension (or "Plugin") for JAS3. In the tutorial we develop the extension in several phases, each phases shows a new feature of JAS3. The extension we develop does not do anything useful, it just aims to illustrate how to interface to the various capabilities of the JAS3 framework.

Developing an extension for JAS3 will require an understanding of Java itself and Swing (the Java GUI). You should also be familiar with running the java compiler, and building jar files using the jar tool. You can either do this using the line-mode tools which come with the Java SDK, or by using a suitable IDE. (The JAS team recommends the Netbeans IDE, but there is no requirement that you use this.) If you are not already familiar with these tools we suggest looking at Sun's Java tutorial, and developing at least a simple "Hello World" application in Swing before attempting to develop a JAS3 extension.

References:

- The Java Tutorial
- Lesson: User Interfaces that Swing: A Quick Start Guide

Extensions vs Plugins

The JAS3 application is based on the FreeHEP application framework. If you were to run JAS3 with no extensions installed at all you would see a hollow application shell, looking something like this:

All of the useful functionality of JAS3 is provided by extensions. These are broken into two types:

- Plugins: These are simple extensions that are always included with JAS3.
- Extensions: These are typically more complex extensions, which can be optionally included with the JAS3 software.

Technically Plugins and Extensions are developed in exactly the same way, the only difference is that Plugins are normally packaged into the jas3.jar file, while extensions are distributed in their own separate jar file(s).

Write a simple Plugin

In this initial part of the tutorial we will develop a very simple extension which simply adds a "Hello World" item to the help menu. When this item is activated a dialog will appear. In this tutorial you will learn how to build and package extensions.

The source code for the first version of the HelloWorld extension is shown below.

As you can see this first version is very simple. There is a single class which extends Plugin.
All extensions must extend this class. For our purposes the class has one important method, Plugin.init(), which is overridden to install our menu item. In this method we create the new JMenuItem, and install it into the Help menu using the Plugin.addMenu(javax.swing.JMenuItem, long) method.

HelloWorldPlugin.java

```
package test;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import org.freehep.application.studio.Plugin;

/**
 * An example of how to write a JAS3 extension. Phase 1.
 * @author tonyj
 */

public class HelloWorldPlugin extends Plugin implements ActionListener
{
   protected void init() throws Throwable
   {
      // Create the menu item
      JMenuItem item = new JMenuItem("Hello World");
      item.addActionListener(this);
      // Add an item to the help menu
      addMenu(item,900901);
   }

   public void actionPerformed(ActionEvent e)
   {
      // Popup a message dialog.
      // The dialog is parented to the main application (JAS3).
      JOptionPane.showMessageDialog(getApplication(),"Hello World!");
   }
}
```

To use this class as an extension you will also need to create a plugins.xml file which describes your extension to JAS3. Here is an example plugins.xml file for the HelloWorld extension. This file must be saved in a directory called PLUGIN-inf (please create it).

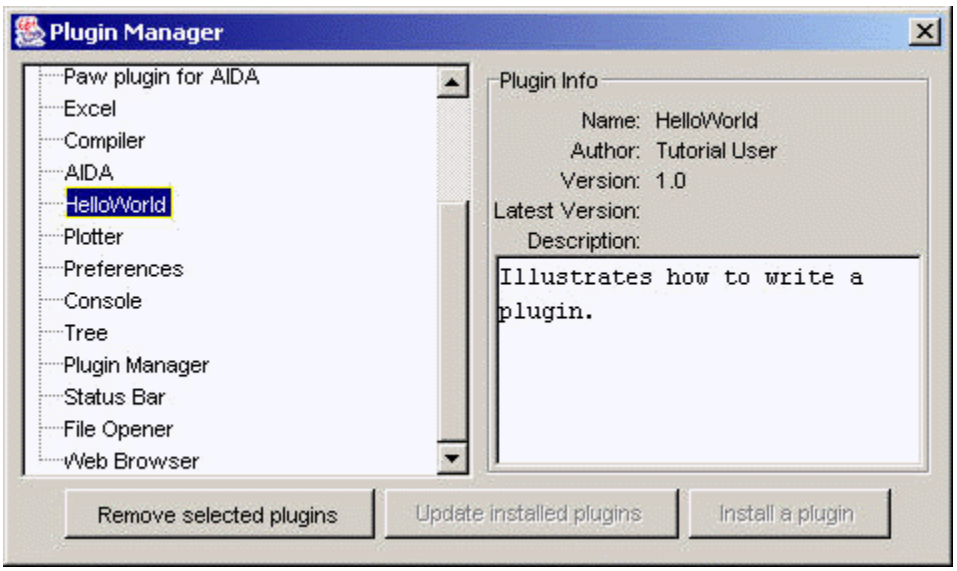
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plugins SYSTEM "http://java.freehep.org/schemas/plugin/1.0/plugin.dtd">

<plugins>
    <plugin>
      <information>
          <name>HelloWorld</name>
          <author>Tutorial User</author>
          <version>1.0</version>
          <description kind="short">HelloWorld extension</description>
          <description>Illustrates how to write a plugin.</description>
          <load-at-start/>
      </information>
      <plugin-desc class="test.HelloWorldPlugin"/>
    </plugin>
</plugins>
```

Finally you need to compile your java file, and package it into a jar file. The easiest way to do this is using ant, a make-like system commonly used by Java projects. Here is an ant build.xml file which will package up your extension into a HelloWorld.jar file, and then install the extension into JAS3.

```
<?xml version="1.0" encoding="UTF-8"?>
<project basedir="." default="all" name="JAS3Tutorial">
    <target depends="init,compile,jar,deploy" name="all"/>
    <target name="init">
       <property name="phase" value="1"/>
       <property name="srcDir" value="phase${phase}"/>
       <property name="tmpDir" value="tmp/${phase}"/>
       <property name="jas3Dir" value="/Program Files/Stanford Linear Accelerator Center/JAS3"/>
    </target>
    <target name="compile" depends="init">
       <mkdir dir="${tmpDir}"/>
       <javac srcdir="${srcDir}" destdir="${tmpDir}">
          <classpath>
            <fileset dir="${jas3Dir}">
              <include name="lib/*.jar"/>
            </fileset>
          </classpath>
       </javac>
    </target>
    <target name="jar" depends="compile">
       <jar jarfile="HelloWorld.jar">
           <fileset dir="${tmpDir}">
                <include name="**/*.class"/>
           </fileset>
           <fileset dir=".">
               <include name="PLUGIN-inf/plugins.xml"/>
           </fileset>
        </jar>
    </target>
    <target name="deploy" depends="jar">
       <copy file="HelloWorld.jar" toDir="${user.home}/.JAS3/extensions"/>
    </target>
    <target name="clean" depends="init">
       <delete dir="tmp"/>
    </target>
</project>
```

Finally you should be able run JAS and see the new item in the Help menu. You should also be able to fire up the Plugin Manager (from the JAS3 View menu) to see that JAS3 has loaded your plugin.



Manipulating Windows

In this phase we will add a second way of displaying Hello World, this time we will open a window in the main JAS GUI, and display "Hello World!" there. Admittedly not a huge leap forward, but you will learn how to create JAS3 windows.

HelloWorldPlugin.java

```
package test;

import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JLabel;
import javax.swing.JMenuItem;
import org.freehep.application.mdi.PageManager;
import org.freehep.application.studio.Plugin;

/**
 * An example of how to write a JAS3 extension. Phase 2.
 * @author tonyj
 */
public class HelloWorldPlugin extends Plugin implements ActionListener
{
   protected void init() throws Throwable
   {
      // Create the menu item
      JMenuItem item = new JMenuItem("Hello World");
      item.addActionListener(this);
      // Add an item to the help menu
      addMenu(item,900901);
   }

   public void actionPerformed(ActionEvent e)
   {
      // Get the applications PageManager
      PageManager manager = getApplication().getPageManager();
      // Ask it to open a new page and display our message
      JLabel label = new JLabel("Hello World!");
      label.setFont(new Font("Serif",Font.BOLD,36));
      manager.openPage(label,"Hello World", null);
   }
}
```

Using the Console Service

In this next phase of the tutorial we will add extra code to log a message to the console each time a window is opened or closed. To do that we will make use of an indirect lookup of the console service, which is implemented by a different extension. The purpose of using the lookup is to isolate our extension from direct dependence on the console service, if the console service is not present our extension will still run, albeit with reduced functionality. In addition if in future an new improved console service is introduced our extension will continue to run correctly.

HelloWorldPlugin.java

```
package test;

import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.PrintWriter;
import javax.swing.JLabel;
import javax.swing.JMenuItem;
import org.freehep.application.mdi.PageContext;
import org.freehep.application.mdi.PageEvent;
import org.freehep.application.mdi.PageListener;
import org.freehep.application.mdi.PageManager;
import org.freehep.application.studio.Plugin;
import org.freehep.jas.plugin.console.ConsoleService;

/**
 * An example of how to write a JAS3 extension. Phase 3.
 * @author tonyj
 */
public class HelloWorldPlugin extends Plugin implements ActionListener, PageListener
{
   private PrintWriter log;
   protected void init() throws Throwable
   {
      // Create the menu item
      JMenuItem item = new JMenuItem("Hello World");
      item.addActionListener(this);
      // Add an item to the help menu
      addMenu(item,900901);

      // Use the console service to open a console
      ConsoleService cs = (ConsoleService) getApplication().getLookup().lookup(ConsoleService.class);
      if (cs != null)
      {
         log = new PrintWriter(cs.getConsoleOutputStream("Log",null),true);
      }
   }

   public void actionPerformed(ActionEvent e)
   {
      // Get the applications PageManager
      PageManager manager = getApplication().getPageManager();
      // Ask it to open a new page and display our message
      JLabel label = new JLabel("Hello World!");
      label.setFont(new Font("Serif",Font.BOLD,36));
      PageContext context = manager.openPage(label,"Hello World", null);
      context.addPageListener(this);
      if (log != null) log.println("Opened Page: "+context.getTitle());
   }

   public void pageChanged(PageEvent e)
   {
      if (log != null) log.println("PageEvent received "+e);
   }
}
```

Registering your own Services

HelloWorldPlugin.java

```
package test;
```

```
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.PrintWriter;
import javax.swing.JLabel;
import javax.swing.JMenuItem;
import org.freehep.application.mdi.PageContext;
import org.freehep.application.mdi.PageEvent;
import org.freehep.application.mdi.PageListener;
import org.freehep.application.mdi.PageManager;
import org.freehep.application.studio.Plugin;
import org.freehep.application.studio.Studio;
import org.freehep.jas.plugin.console.ConsoleService;

/**
 * An example of how to write a JAS3 extension. Phase 4.
 * @author tonyj
 */
public class HelloWorldPlugin extends Plugin implements ActionListener, PageListener, HelloWorldService
{
   private PrintWriter log;
   private Studio app;
   protected void init() throws Throwable
   {
      app = getApplication();
      // Create the menu item
      JMenuItem item = new JMenuItem("Hello World");
      item.addActionListener(this);
      // Add an item to the help menu
      addMenu(item,900901);

      // Register the HelloWorldService
      app.getLookup().add(this);

      // Use the console service to open a console
      ConsoleService cs = (ConsoleService) app.getLookup().lookup(ConsoleService.class);
      if (cs != null)
      {
         log = new PrintWriter(cs.getConsoleOutputStream("Log",null),true);
      }
   }

   public void actionPerformed(ActionEvent e)
   {
      // Look for the HelloWorldService
      HelloWorldService hws = (HelloWorldService) app.getLookup().lookup(HelloWorldService.class);
      if (hws != null) hws.say("Hello World");
      else app.error("HelloWorldService not available");
   }

   public void pageChanged(PageEvent e)
   {
      if (log != null) log.println("PageEvent received "+e);
   }

   public void say(String message)
   {
      // Get the applications PageManager
      PageManager manager = getApplication().getPageManager();
      // Ask it to open a new page and display our message
      JLabel label = new JLabel(message);
      label.setFont(new Font("Serif",Font.BOLD,36));
      PageContext context = manager.openPage(label,"Hello World", null);
      context.addPageListener(this);
      if (log != null) log.println("Opened Page: "+context.getTitle());
   }
}
interface HelloWorldService
{
   void say(String message);
}
```