SWAMP Eclipse Plug-in Maintenance and Documentation

Malcolm Reid Jr.

May 11, 2017

1 Eclipse Architecture

The Eclipse ecosystem is made up of plug-ins and features, where features are entities that package one or more plug-ins and/or features. Eclipse provides the Plugin Development Environment perspective to simplify the creation of a plug-in. You can download the Eclipse Plugin Development Environment either when you install Eclipse or from the Eclipse Marketplace.

2 Plug-in Code Structure

2.1 plugin.xml

Actions, buttons, and menus are all specified in a plugin.xml file. These can be associated with handlers to provide some functionality. This is also where plug-in build, version, and dependencies reside.

In the org.continuousassurance.swamp.eclipse.handlers package, you can find the Handler classes that the plug-in uses. These all subclass the Eclipse-provided AbstractHandler class.

2.2 Packages

The plug-in has the following packages:

- **org.continuousassurance.swamp.eclipse**: The classes in this package provide core functionality for the plug-in including start-up, serialization/deserialization of assessments, build file generation, results retrieval, and results parsing.
- **org.continuousassurance.swamp.eclipse.dialogs**: This package houses the dialog classes. All of the dialogs inherit from Eclipse SWT's TitleAreaDialog.
- **org.continuousassurance.swamp.eclipse.exceptions**: This package contains plugin-specific exceptions.

- **org.continuousassurance.swamp.eclipse.handlers**: This package contains handlers, which are classes that link an action, button, menu, etc. from the plugin.xml to actual code execution.

- **org.continuousassurance.swamp.eclipse.ui**: This package contains classes for results UI/UX. For example, all of the new views in the SWAMP perspective are in this package.

2.3 Architecture

The Activator class is invoked when a user opens Eclipse. In this class, we do a lot of initialization such as trying to log the user again and determining the user's marker preferences. We also start the background job for results checking at this point.

You can think of the plug-in as having two separate but linked parts: submitting assessments and viewing results.

In terms of submitting assessments, the most important components are the SwampSubmitter class and the SubmissionInfo class. The SwampSubmitter class is the workhorse for assessment submission. It is what triggers the submission process. The SubmissionInfo class is the back-end data store for information about a submission (i.e. a set of assessments). It can be populated either by the dialogs, with the user entering the necessary information, or from file, in which the object is given its information based on the contents of a file generated from the previous assessment with this Eclipse project.

When we auto-generate a build file, the ImprovedClasspathHandler class processes the project's classpath. It traverses through every entry in the classpath and handles it accordingly. The BuildfileGenerator class then uses the information that the ImprovedClasspathHandler gleaned to generate an Ant build file.

In terms of viewing results, there are three components: views, the SWAMP perspective, and the Controller. Views display information and/or allow user interaction. They are relatively simple to implement and use the same SWT widgets as dialogs. The SWAMP perspective places all of the views and also initializes a FileChangeListener. This listener just runs when the file is changed while the user is in the SWAMP perspective. The purpose of this is to allow us to show different results if appropriate. Finally, the Controller is a singleton that allows the views to communicate with each other and with the perspective. For example, if the file changes, the FileChangeListener might need to tell the views to update. The Controller also allows us to refresh the workspace when new results come in.

The StatusChecker job runs every 30 seconds to check for pending jobs. It reads assessment identifiers from a file of unfinished assessment information and writes the new status of each formerly unfinished assessment either to the unfinished file or a file of finished assessment information (if the assessment is finished).

2.4 Dialogs

All of the dialogs subclass TitleAreaDialog and use the Eclipse Standard Widget Toolkit (SWT). SWT provides a high-level API for adding UI components. On okPressed(), each dialog does whatever validation checks it needs to do.

3 Dependencies

Here are the dependencies of the plug-in and a brief explanation of what they are used for and where to get new versions of them:

1. **commons-io-2.5.jar**: Provides simple file utility methods. New versions come from Apache Commons website
2. **java-cli-1.0-SNAPSHOT-jar-with-dependencies.jar**: Provides APIs for interacting with the SWAMP. We can get new versions from https://github.com/mirswamp/java-cli.
3. **javaSCARF.jar**: Java parser for SCARF data. We can get new versions from https://github.com/mirswamp/scarf-io.
4. **javax.json-1.0.4.jar**: Provides APIs for dealing with JSON objects using Java. We can get new versions from https://mvnrepository.com.
5. **json-lib-2.4-jdk15.jar**: Provides APIs for dealing with JSON objects using Java. We can get new versions from https://mvnrepository.com.

4 Common Tasks

4.1 Setting up Eclipse for Plug-in Development

To set up Eclipse for plug-in development, either install Eclipse with the Plug-in Development Environment included by following the instructions at https://www.eclipse.org or install Plug-in Development Environment from the Eclipse Marketplace.

4.2 Opening a Project in Eclipse

To open a project in Eclipse, do the following:

1. Click "File" ⟩ "Open Projects from File System."
2. Click "Directory..." and select the directory of the project
3. Click "Finish"

4.3 Opening the Plug-in Project in Eclipse

To open the plug-in project in Eclipse, do the following:

1. Download the plug-in source locally, by running git clone https://github.com/mirswamp/swamp-eclipse-plugin/
2. Open an Eclipse instance
3. Follow the steps in 4.2

4.4 Building

Eclipse allows you to build from source. To run, either press ⟨c-F11⟩ or Run ⟩ Run in the menubar. This will create a new Eclipse instance with the plug-in installed.

4.5 Debugging

To help debug issues, you can print to STDOUT (using System.out.print()) or to STDERR (using System.err.print()). By default, both show up in the console, with the former appearing in black and the latter red. You can also use the Eclipse debugger. Set one or more break points by double-clicking locations in the left hand ruler. Then run the plug-in in Debug mode by either pressing ⟨F11⟩ or clicking Run ⟩ Debug in the menubar. When the plug-in reaches a break point, it will automatically open the Debug perspective, which provides information like the local variables and the current call stack. At this point, you can make modifications to the code in the current method, and Eclipse will "replay" starting at the top of the method. This is helpful for debugging and testing in real time.

4.6 Update Dependencies

All of the dependencies for the SWAMP Eclipse plug-in are found in the lib/ directory. There are a few steps involved in updating the dependencies.

1. Store the .jar in the lib/ directory - Note: you may have to refresh the directory in Eclipse before Eclipse recognizes that the lib/ directory has been modified.
2. Add the dependency to the build path - Right click on the Eclipse project and select "Build Path – Configure Build Path." In the Properties dialog that opens, select "Java Build Path." Switch to the "Libraries" tab, click "Add JARs..." and select your new JAR.
3. Add the dependency to the classpath - On the Eclipse plugin.xml, select the "Runtime" tab. In the classpath section click "Add..." and select the new JAR.

4.7 Package a Release

Note that the plug-in and feature projects both exist in the Github repository, whereas the update site project is located at /p/swamp/gits/swamp-eclipse-update-site.git. Run git clone on that location to download a local copy of the update site repository.

The update site has a site.xml with some information about the update site and the features it contains. It also has current and old versions of features and plugins.

1. Open the SWAMP Eclipse plug-in project in Eclipse's Plug-in Development Environment by clicking File ⟩ Open Project from File System.
2. In the Overview tab of the plugin.xml, update the version of the plug-in as appropriate.
3. Open the SWAMP Eclipse feature project in Eclipse's Plug-in Development Environment by clicking File ⟩ Open Project from File System.
4. Update the version of the "eclipseplugin" in the "Included Plug-ins" tab of the feature.xml to match the newly released version.
5. Open the SWAMP Eclipse update site in Eclipse's Plug-in Development Environment by clicking File ⟩ Open Project from File System.
6. Open the site.xml of the SWAMP Eclipse update site project.
7. Select the "SWAMP" category and click "Add Feature..."
8. Select the name of the SWAMP Eclipse feature project
9. Click on the feature and click "Synchronize..."
10. Select "Synchronize selected features only" and click "Finish"
11. Make sure "This feature is a patch for another feature" is selected
12. Click "Build"
13. Commit and push the changes to your local copy of the update site repository
14. To archive this update site, execute the following in the directory of the release script: ./releaseScript ⟨version-number⟩ (where version-number is the version number with qualifier of the feature). This will create an archive of the update site with documentation in the user's $HOME/Desktop directory. This is a zip for uploading releases to GitHub Releases or Google Drive. It will also create a tar of the update site without documentation for the actual swampinabox.org update site.
15. scp ⟨tarball-name⟩ to swa.platform-01.cosalab.org
16. run: sudo kupsch/install-eclipse_update_site ⟨tarball-name⟩

4.8 Signing JARs

Eclipse will warn users when they attempt to install unsigned content. If the content is self-signed, Eclipse will ask the user whether he or she trusts the certificate.

4.9 Signing the Plug-in

To sign the plug-in, follow the steps listed at http://timjansen.github.io/jarfiller/javabasics/jar/selfsigning.xhtm You will want to sign all of the JARs in the features/ and plugins/ directories, as well as the artifacts.jar and content.jar in the top level directory of the update site. For example, to sign a specific JAR file, you might do:

jarsigner -keystore malcolmKeystore -verbose /public/html-s/swamp-eclipse/artifacts.jar malcolmSWAMPKey

4.10 Modify Eclipse Marketplace Listing

To modify the Eclipse Marketplace listing, do the following:

1. Log into the Eclipse website https://www.eclipse.org with the credentials we have established for the SWAMP Eclipse plug-in
2. Select Eclipse Marketplace
3. Click "Edit"

5 Tips

5.1 Issue Tracking and Version Control

I used GitHub issues for issue tracking. Feel free to look at the issues to figure out how I made fixes and enhancements.

I tried to commit in units as small as possible and include helpful commit messages.

5.2 C/C++

The documentation on the Eclipse C/C++ plug-in is very limited. I've documented what I've learned in the plug-in.