Ali Almossawi <almossawi@cmu.edu>
September 5, 2006

The Novice Eclipse Plug-in Developer's Help-sheet

1. Introduction

Becoming comfortable with Eclipse plug-in development in a short amount of time can be a daunting experience given its learning curve and the large amount of sometimes scattered information that is available on the Internet. This short help-sheet will hopefully aid the interested reader who would like to know how to quickly get up to speed with Eclipse plug-in development and make good use of the tutorials, articles and source code that have already been written by others. The workflow described here is one that I followed when writing my first Eclipse plug-in, which is discussed in [11].

2. Realize your universe's boundaries

I have found that there is great benefit when starting any learning experience to firstly digest large amounts of information on the subject in a preferably uninterruptible burst in order to get a feel for the domain, eliminate the possibility of resorting to hacking somewhere down the line and reduce the overall learning curve. Pacing across the field of interest to its very tips gives one a good amount of confidence and makes the later learning process mostly one of remembering things instead of one of finding them out for the first time. Clayberg and Rubel's text is a good read for the novice because it starts with the very basics by describing the Eclipse framework and putting plug-in development in context and then proceeds to cover a lot of ground.

I also found it useful to print out all the classes of the Java editor sample plug-in available from [4] and read through them in like manner to get a first-hand feel for how the implemented parts of a plug-in fit together. Information on classes, methods and sometimes concepts was frequently looked up in the Eclipse API at [1]. Looking through the source code of other plug-ins, many of which are available from [2], was also beneficial. In my case, an XML editor plug-in was particularly interesting since it supported the type of checking and reporting of errors that I had in mind for my editor.

3. Create a barebones plug-in

To avoid getting overwhelmed, a good second step is to create a barebones plug-in. Since I wanted to write a text-based editor, I followed the tutorial found at [3] while also referring to some of the plug-ins I had downloaded earlier. The tutorial covers an older version of Eclipse, but with a bit of tweaking it works just fine. The additional effort is in fact a blessing since it brings to light some things about Eclipse that perhaps one wouldn't have otherwise learnt. Alternatively, one may wish to just use the example plug-in that [10] progressively builds up instead.

After laying the groundwork with a barebones plug-in, the next step for me was to incorporate a few things from the Java editor example from [4] to see if those bits could be fitted into it without breaking anything. Of course, it's always nicer if something does break so that the experience becomes more interesting and educational; numerous breaks though do take their toll on the ego. Eclipse's Local History feature proved invaluable here as it allowed painless rolling back of files to previous versions.

4. Construct the rest piece by piece

A divide-and-conquer approach is always a good idea when dealing with a complex system. After deciding what my editor was going to include and prioritizing all of its functional requirements, each requirement was then implemented in a systematic way that involved following a tutorial or a section from [10] and looking up information on classes and methods in the API whenever necessary. For my editor, the main requirements were that it should:

- Display the results of syntax and semantics checks in the **problems view** with associated markers in the gutter.
- Display a tree representation of an active document's Abstract Syntax Tree (AST) in the **outline view**, link its nodes to lines of code and allow the user to toggle certain options like sorting of the nodes.
- Support **content assistance** for the language's keywords and tactics.
- Support **syntax highlighting**.
- Allow the user to set or toggle certain options via a **preference page**.

Now, this set is actually the result of the classification that was done on the initial set of requirements, which was simply an enumeration of all the things that the editor was expected to have. No attention was given to how the technology would support them all. After spending some time looking into that, it was possible to group requirements that map to a single part or close enough parts of the Eclipse framework. Now it was easy to build up and slowly expand the plug-in's functionality because one only needed to concern oneself with a more or less isolated part of the whole, as defined by Eclipse, when wanting to work on some requirement. In addition to [10], the tutorials and articles at [5], [6], [7] and [8] were all used to aid the implementation process. And that was really it.

5. Final thoughts

Simplifying complex systems by reducing them to smaller sub-systems is something that is perhaps very intuitive to the human mind, which may lead the reader to consider the references listed in the next section to be the most valuable part of this help-sheet. That may well be the case, but I would still suggest reading through the entire document to enjoy the few light-hearted attempts at humor.

References

[1] Eclipse SDK API, http://help.eclipse.org/help31/index.jsp

[2] Eclipse Plug-in Central, http://www.eclipseplug-incentral.com

[3] Creating a text-based editor for Eclipse 2.1, http://devresource.hp.com/drc/technical_white_papers/eclipeditor/index.jsp

[4] Eclipse example plug-ins download, http://download.eclipse.org/eclipse/downloads/drops/R-3.1.2-200601181600/index.php

[5] Simplifying Preference Pages with Field Editors, http://safariexamples.informit.com/0321268385/Eclipse%20materials/Eclipse%20online%20articles/Simplifying%20Preference%20Pages%20with%20Field%20Editors.pdf

[6] Preferences in the Eclipse Workbench UI, http://www.eclipse.org/articles/Article-Preferences/preferences.htm

[7] Equipping SWT applications with content assistants, http://www-128.ibm.com/developerworks/opensource/library/os-ecca/

[8] Mark My Words, http://www.eclipse.org/articles/Article-Mark%20My%20Words/mark-my-words.html

[9] Season's Text Editor Recipes, Tom Eicher, http://www.eclipse.org/eclipse/platform-text/eclipseCon/2005/poster.pdf

[10] Eclipse: Building Commercial-Quality Plug-ins (2nd ed.), Eric Clayberg and Dan Rubel, Addison-Wesley Professional, 2006

[11] Contributing to Rainbow's Stitch Adaptation Language Final Report, http://cyberiapc.com/rainbow/images/4/42/Almossawi_final_report.pdf