

Developing an Eclipse Plugin



Paul Tremblett
Harmonia, Inc.





Do We Need Plugins?

- Eclipse 3.3 (Europa) is approximately 17 million lines of code
- So it can do everything, right?
- Mmmmmmm, no
- Eclipse was not designed as a “single, total solution”
- The majority of Europa's 17 million lines of code is found in plugins



The Plugin Mindset

- Eclipse consists of:
 - A very small core
 - A LOT of plugins
- The rules to follow are:
 - (1) If you want to extend and/or modify Eclipse, start thinking in terms of plugins
 - (2) See (1)



Advantages of Plugins

- Much of the work is already done for you
- The development effort is well-defined
- The work you do can be used by others



What Is a Plugin?

- Essentially a JAR file
- Has a special type of manifest and (optionally) a plugin.xml file
- May depend on other plugins
- May contribute to other plugins



Developing Plugins

- Plugins are developed using the Plugin Development Environment (PDE)
- PDE is (you guessed it) a plugin
- PDE is included in JEE, RCP/Plugin and Classic Eclipse packages
- Can be downloaded separately if you are using Java or C/C++ packages



Before You Start

- Make certain the plugin you plan to develop doesn't already exist
 - Check the following URLs:
 - <http://www.eclipseplugincentral.com/>
 - <http://eclipse-plugins.2y.net/eclipse/plugins.jsp>



Getting Started

- One of the best ways to start is to use the templates provided and study the code that is generated
- We will do this shortly
- Q. But how much can you possibly learn from HelloWorld?
- A. You'd be surprised (and we will go beyond HelloWorld)



Roll Up Your Sleeves

- Plugin development is more about doing than viewing
- So let's start doing
- If you have Eclipse 3.3 and PDE installed, you can follow along if you choose (but we can not pause to solve individual difficulties like we could if this were a lab)



Generated Code

- After using wizard, you should see:
 - MANIFEST.MF
 - plugin.xml (optional)
 - Activator (optional)
 - Plugin-specific code



MANIFEST.MF

- The Hitchhiker's Guide to The Plugin
- Use manifest editor to modify
- Can be edited manually but not recommended
- Divided into sections
 - Each section is a tab in the manifest editor



Plugin.xml

- Defines extension aspects of a plugin
 - Extensions: the mechanism whereby a plugin extends existing functionality
 - Extension points: the mechanism whereby a plugin allows others to extend it



The Manifest Editor

- Activated by double-clicking on either MANIFEST.MF or plugin.xml
- Displays 9 tabs
- Tabs correspond to sections in MANIFEST.MF or provide information required to build the plugin



Overview Tab

- Displays:
 - Plugin ID
 - Version
 - Name
 - Provider
 - Class



Dependencies Tab

- Describes the reliance of this plugin on other plugins in the system
- Corresponds to *Requires-Bundle* and *Import-Package* sections in MANIFEST.MF



Runtime Tab

- Defines which libraries are delivered and which libraries are used at runtime
- Controls package visibility
- Corresponds to *Bundle-Classpath* in MANIFEST.MF



Extensions Tab

- Describes how plugin extends existing functionality
- Corresponds to `<extension>... </extension>` elements in `plugin.xml`



Extension Points Tab

- Provides a way for others to extend this plugin
- Corresponds to
<extension-point>...
</extension-point> elements in
plugin.xml



Build Tab

- Displays a Build Configuration editor
- Used to control files and directories that are included in the build
- Manipulates *build.properties* file



Build.properties Tab

- Displays the contents of the *build.properties* file
- Used in conjunction with the Build tab



Plugin.xml Tab

- Displays the contents of the *plugin.xml* file
- Displayed file is directly editable
- Manual editing is not recommended



MANIFEST.MF Tab

- Displays the contents of the *MANIFEST.MF* file
- Displayed file is directly editable
- Manual editing is not recommended



Activator

- Activators are optional
- Extends *AbstractUIPlugin*
- Also called the plugin class
- When plugin is activated, this class is instantiated before any other class in the plugin is loaded



Resources

- The Eclipse community moves quickly so books tend to be perpetually out of date
- I recommend:
 - *Eclipse: Building Commercial-Quality Plug-Ins*
by Eric Clayberg and Dan Rubel
ISBN 0-321-42672-X