IBM

Manage your Eclipse environment

Zen and the art of Eclipse maintenance

Chris Aniszczyk, Software Engineer, IBM
Phil Crosby (philc@philisoft.com), Student, University of Maryland, College Park

Summary: The continuing growth of Eclipse means that there will always be an increase in the number of projects and plug-ins to manage. As a developer, this management process can be frustrating when staying up to date with the latest Eclipse builds. As a new user, the concept of projects, plug-ins, workspaces, and installations may seem daunting at first. This article aims to show some best practices for managing your Eclipse environment.

Date: 14 Feb 2006
Level: Introductory
Activity: 4045 views
Comments: 0 (Add comments)

 Average rating

Managing your plug-ins

What's a plug-in and why should I care?

A *plug-in* (also known as a *bundle*) is a piece of functionality in Eclipse. There are plug-ins for everything seen in Eclipse, including:

- Perspectives and views
- Editors
- Modeling tools
- Logging and other core functions

In fact, the entire Eclipse IDE is built to be just a big collection of plug-ins. Other Eclipse-based products, like IBM Rational® Software Architect, enhance the base Eclipse by adding new plug-ins.

Sets of related plug-ins are grouped into *features*. The features and their plug-ins live in the Eclipse program directory (in this example, Eclipse is installed to /opt/eclipse). Here's a sample directory layout:

Listing 1. Eclipse feature and plug-in directory structure

```
/opt/eclipse/
   features/
      org.eclipse.jdt_3.1.1/
         feature.xml
         ...
   plugins/
      org.eclipse.jdt.ui_3.1.1.jar
      ...
```

Eclipse can have many of the same plug-ins, each a different version. It knows how to resolve plug-in dependencies and avoid version conflicts, so you don't need to ever worry about having two Subclipse plug-ins installed at the same time.

When using Eclipse for an extended period of time, you invariably add new functionality to it by downloading third-party plug-ins or creating your own. Managing these plug-ins across different versions of Eclipse can be a hassle. Since they live in the Eclipse program directory, they are lost if you install a new version of Eclipse. This means you must keep multiple copies of large numbers of plug-ins around if you have many Eclipse installations; or worse, you have to go through the trouble of reinstalling all those plug-ins each time you want to upgrade Eclipse.

If you have your plug-ins stored in a separate location outside of the Eclipse program directory, you don't have to reinstall them when you upgrade to a new version of Eclipse, and you can share plug-ins across multiple versions of Eclipse.

Take control: Method 1 -- Manual file system extensions

There are three ways to take control of your plug-ins. The first is manually creating a directory that can hold plug-ins (called a *product extension*), moving your plug-ins there, then telling Eclipse to look in this directory for features and plug-ins.

For our example, we'll be creating a location called /opt/eclipse-plugins that will house the plug-ins. To have Eclipse store plug-ins here, you must first create the following directory structure and files:

Listing 2. Eclipse product extension directory structure

```
/opt/eclipse-plugins/
   eclipse/
      .eclipseextension
      features/
      plugins/
```

Note that, in addition to creating those directories, you must also create a file called .eclipseextension in the eclipse directory (in our example, /opt/eclipse-plugins/eclipse). This file lets Eclipse know there are extensions to be found. It should have the following contents:

```
id=org.eclipse.platform name=Eclipse Platform
version=3.1.1
```

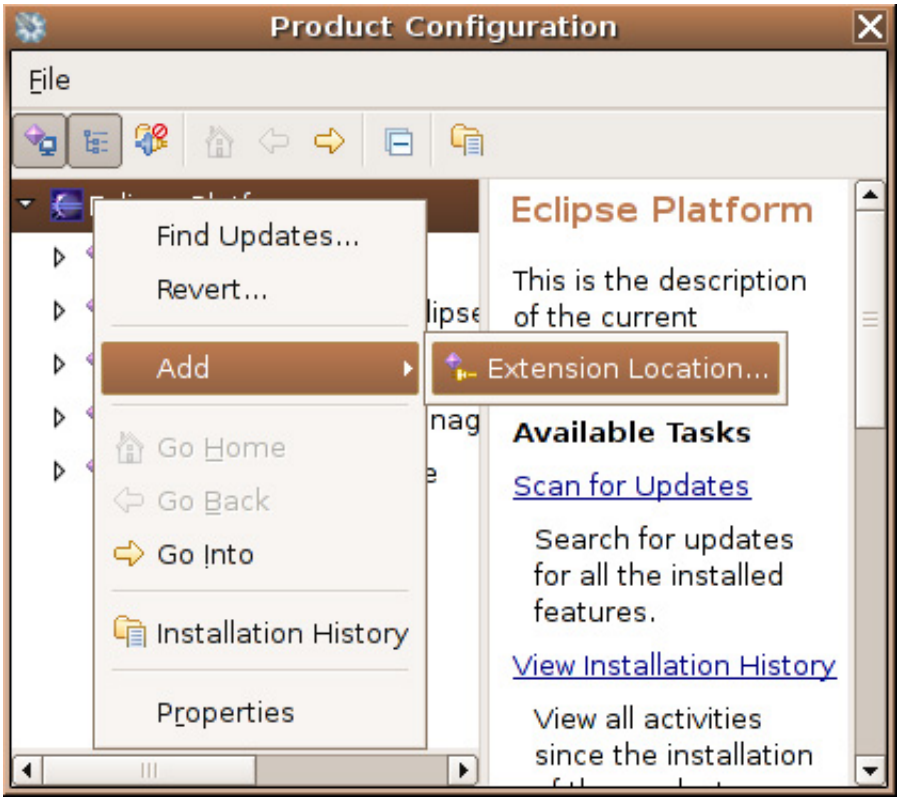
The `version` property in the .eclipseextension file should be set to the version of Eclipse that's using this product extension. This can be specific (3.1.1), more general (3.0.0) or very general (1.0.0). At the time of writing, the version number doesn't seem to have any effect on the functionality of the product extension.

Note that if you're using Windows®, you can't create an .eclipseextension file through the Explorer shell. You can create it by opening Notepad, entering the contents of the file, and saving it as .eclipseextension (ensure that "all files" is selected as the file type or Notepad appends .txt to the file).

The next thing to do is to tell Eclipse about this plug-in location, so it knows to look here for plug-ins in the future. This can be done with the Product Configuration Manager, accessible via **Help > Software Updates > Manager Configuration**.

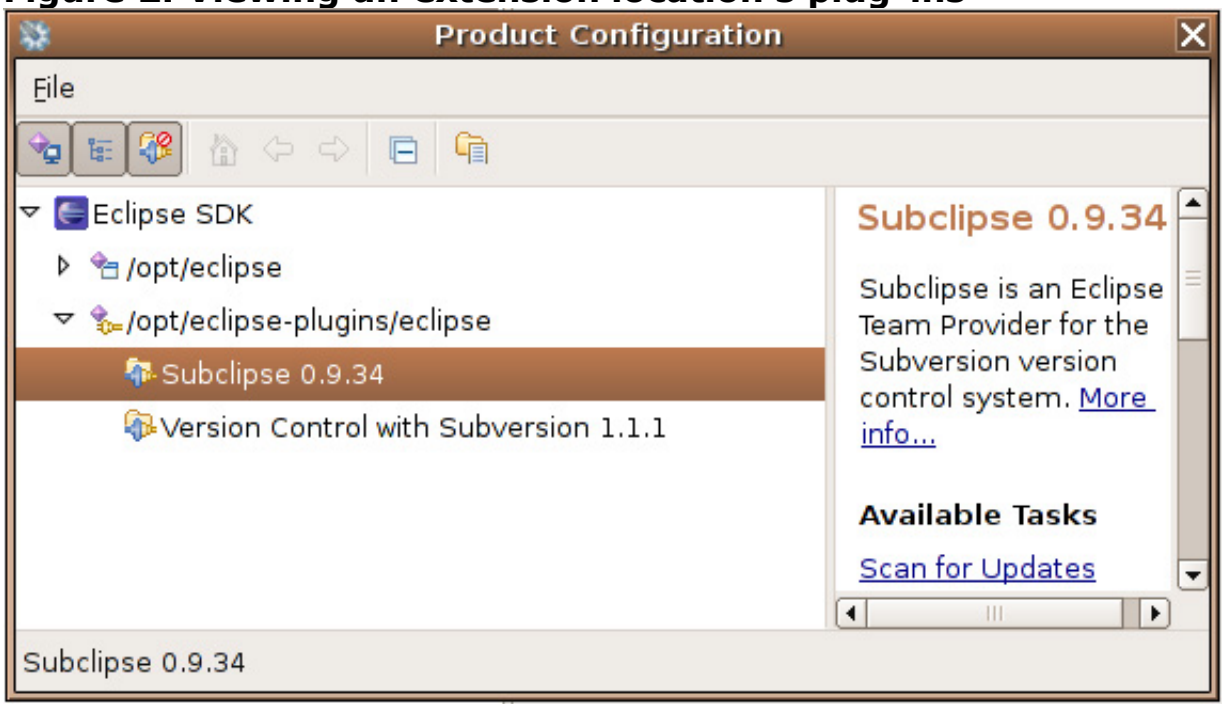
From the Product Configuration Manager, you can add new Eclipse extensions. To enable the one created above (/opt/eclipse-plugins), we need to add it as an extension location. Everyone already has one extension location, which is the plugins folder in your Eclipse installation. To add another, right-click on the Eclipse Platform and select **Add > Extension Location**.

Figure 1. Adding extension locations



Once you've selected the directory where the plug-ins reside, they appear in the list of product extensions. From here, you can verify that your plug-ins were found.

Figure 2. Viewing an extension location's plug-ins



What's nice about the Product Configuration screen is that you can disable entire

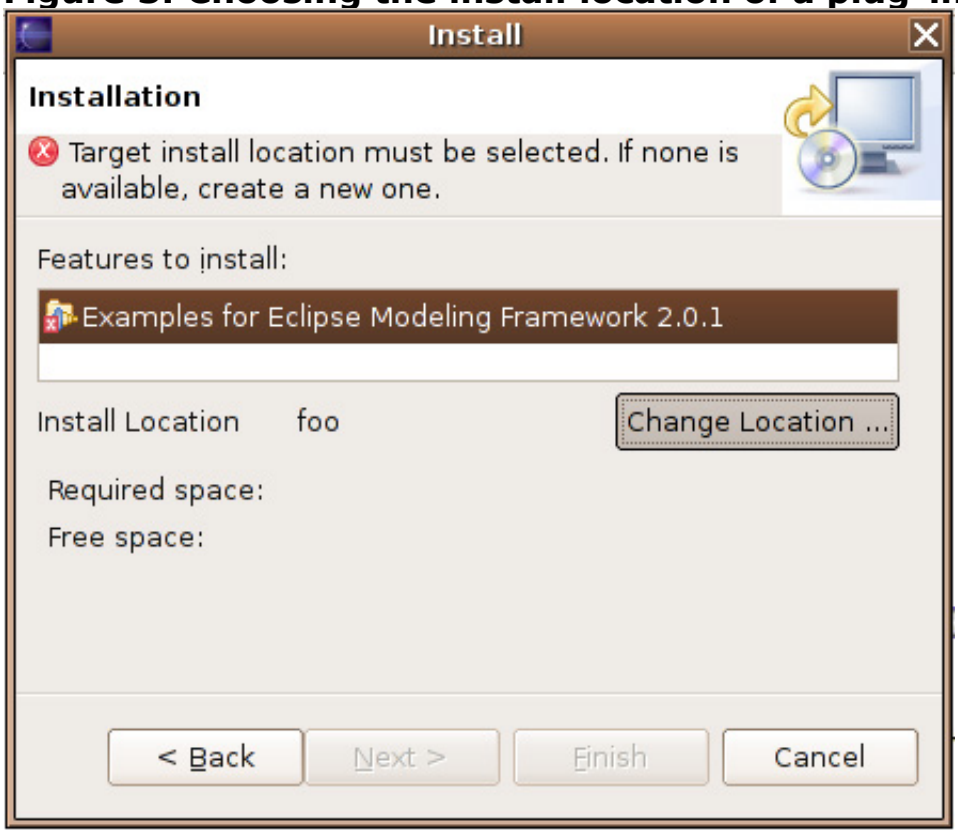
plug-in locations easily -- useful when doing plug-in development and testing various configurations.

Take control: Method 2 -- Adding product extensions through the Configuration Manager

Rather than creating folders and .eclipseextension files on the file system, you can have Eclipse create product extensions for you.

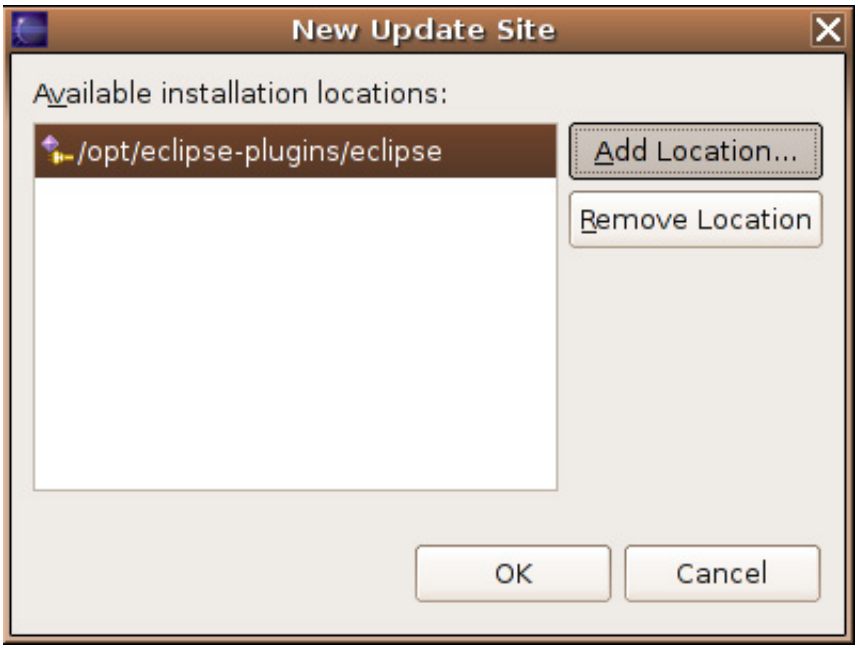
You can create new product extensions in the Update Manager (**Help > Software Updates > Find and Install**). When installing a new plug-in, Eclipse eventually prompts you for the location to install it to. Here, you can click on Change Location to choose a product extension.

Figure 3. Choosing the install location of a plug-in



Choose Add Location. When you select a directory, Eclipse will create a product extension there for you.

Figure 4. Creating a new product extension through Update Manager



When you install plug-ins in the future, ensure that they are being installed to the plug-in extension site you want (see the Install Location area in Figure 3).

Take control: Method 3 -- Creating a links folder to manage product extensions

If you have product extensions sitting on your file system, like the one we made in Method 1, you can create a few simple files in your Eclipse program directory to notify Eclipse that it needs to check these directories for plug-ins.

First, create a directory inside your Eclipse installation folder (for example, /opt/eclipse) called links. Within this folder, you can create *.link files (for example, emfPlugins.link). Each link file points to a product extension location. Eclipse will scan this links folder on startup and find the plug-ins in each product extension pointed to by a link file. Here is an example of an Eclipse installation layout using a links folder:

Listing 3. Eclipse installation layout using a links folder

```
/opt/eclipse/
     links/
          emfPlugins.link
          webtools.link
          updateManager.link
          ...
     ...
```

The contents of the link files should look like the following:

```
path=/opt/plugins/emf/
```

Windows

In this article, we have heavily focused on a Linux installation of Eclipse. All the sample listings provided are valid for Windows. You just have to replace the path with the proper windows path. Also note, in Windows, paths require double backslashes (for example, path=c:\\plugins\\emf).

Where /opt/plugins/emf/ has the directory structure of an Eclipse product extension, like in Listing 1.

The advantage of this method is that all of your plug-in locations are stored as text files in a folder. This means you can upgrade Eclipse and point it at your product extension folders by simply copying the links folder to the new Eclipse installation. You can also have one common links folder for all your Eclipse installs by making a symbolic link to a links folder from each Eclipse installation (if your file system supports symbolic links).

Managing Eclipse workspaces

In Eclipse, the concept of a workspace is simply represented as a container of resources that can be accessed by plug-ins. The workspace is the major point of interaction between the end user and the Eclipse platform. The end user is able to create projects and manipulate the content contained within the workspace. The workspace itself exists in the file system as a directory and has the limitation of only being used per one Eclipse instance. The workspace also contains a .metadata directory that persists private information, such as the state of a plug-in.

Why do I want multiple Eclipse workspaces?

The simple answer is performance. The more projects in your workspace, the greater chance of reaching a point where your development system can't handle the projects. To solve this problem, you can partition out your Eclipse workspaces via the `-data` parameter, which is passed to the Eclipse executable (/opt/eclipse/eclipse, for example):

Listing 4. Specifying different workspaces

```
/opt/eclipse-3.1/eclipse -data /opt/workspaces/web
/opt/eclipse-3.2M2/eclipse -data /opt/workspaces/web
/opt/eclipse-3.2M2/eclipse -data /opt/workspaces/dev -vmargs -Xmx512m
```

Workspace Tips

You can show workspace location in your Eclipse title bar by passing the `-showlocation` parameter to the Eclipse executable. In addition, you can assign different performance characteristics to different workspaces through the arguments `-vmargs -Xms` and `Xmx`.

You can also switch workspaces without restarting Eclipse by selecting **File > Switch Workspace** from within Eclipse.

There is also a prospect of having a "research" workspace, where you have a large code base loaded to search through the code via Eclipse, using Open Type (**Ctrl+Shift+T**), etc. This is incredibly useful if you're trying to learn from examples, or if you have ever wondered how an open source project approached a particular problem.

The downside of having multiple workspaces is the need to share development preferences among them. Since preferences are persisted on a workspace basis within Eclipse, you have to export your workspace preferences and import them to your desired workspaces (**File > Export > Preferences**).

Managing Eclipse installations

Why do I want multiple Eclipse installations?

By necessity, you have multiple Eclipse installations if you use more than one Eclipse-based product. For instance, if you use base Eclipse V3.1 for everyday Java™ language coding, and you use Eclipse with WebTools for authoring IBM WebSphere® applications, then you will have two completely separate Eclipse installations. Sharing plug-ins and workspaces between these Eclipse-based products can save you time and avoid some upgrade headaches.

You might also want multiple Eclipse installations if you're developing Eclipse plug-ins. When you have multiple Eclipse installations, you can test your plug-in's functionality across different versions of Eclipse. You can also associate a different set of plug-ins with different installations of Eclipse so you can test your plug-in across multiple environment configurations.

Note that you can also manage which plug-ins are being used by Eclipse by examining features within the Eclipse IDE (which enables and disables the plug-ins belonging to those features) by selecting **Help > Software Updates > Manage Configuration** from within Eclipse. You can also manage which plug-ins are enabled if you run test instances of Eclipse from within the Run Configuration Manager while

developing your plug-ins. It's generally been our experience that having multiple Eclipse installations is the most portable and reusable way of managing many Eclipse versions and configurations, especially for testing purposes.

Multiple Eclipse installations

Eclipse installations are self-contained within their own folder. To get multiple installations going, it's as simple as downloading the Eclipse products and versions you desire and extracting them to their own directories. Here's a sample layout used for testing plug-ins across different Eclipse versions:

```
/opt/eclipse-3.0
/opt/eclipse-3.1
/opt/eclipse-3.2-m1
```

It's a good idea to share as much as possible across Eclipse installations to save time installing plug-ins for all of the current installs and avoiding workspace duplication. As we've described above, you can share the following:

- **plugins** -- Have one (or many) common plug-in folders all of your installations use. The best way is by creating a links folder, as described in Take control: Method 3.
- **workspaces** -- As described in Managing Eclipse workspaces
- **workspace preferences** -- Preferences that are bound to workspaces. Use **File > Export > Preferences** from within Eclipse.

Please be aware that sharing workspaces and preferences across Eclipse installations can be problematic, especially if the versions of Eclipse differ by major numbers (3.1 and 3.2, for example).

Conclusion

Our goal was twofold: to give you an introduction to Eclipse's most basic units of work, which are plug-ins, projects, and workspaces; and to show you the benefits of managing multiple Eclipse environments, including some downsides. We hope you take this knowledge and apply it to save time in your Eclipse maintenance.

Resources

Learn

- Read "Managing Plugins in Eclipse," one of the original inspirations for this

article.

- Create your own Dependency Walker to find unresolved plug-in dependencies in the developerWorks article "Finding unresolved Plug-in dependencies in Eclipse."
- Find the answer to the question "Can I install plug-ins outside the main install directory?" in Eclipse FAQ 32.
- Get more details about Eclipse at Eclipse.org.
- Learn more about Eclipse by reading technical articles at the Eclipse Corner.
- Learn more about the Eclipse Corner Developer Community Resources.
- Attend the premier Eclipse conference, EclipseCON.
- Learn about Eclipse through Planet Eclipse, a window to the world of Eclipse hackers and contributors.
- Learn about Eclipse through the official Eclipse wiki. Alternatively, you can learn about Eclipse through the Unofficial Eclipse Wiki.
- Visit the developerWorks Open source zone for extensive how-to information, tools, and project updates to help you develop with open source technologies and use them with IBM's products.

Get products and technologies

- Innovate your next open source development project with IBM trial software, available for download or on DVD.

Discuss

- If you are new to Eclipse, you can find lots of information on the Eclipse newsgroups.
- Ask technical questions about Eclipse on the developer mailing lists.
- Get involved in the developerWorks community by participating in developerWorks blogs.

About the authors

Chris Aniszczyk is a software engineer at IBM Lotus focusing on OSGi related development. He is an open source enthusiast at heart, and he works on the Gentoo Linux distribution and is a committer on a few Eclipse projects (PDE, ECF, EMFT). He's always available to discuss open source and Eclipse over a frosty beverage.

Phil Crosby is an undergrad at the University of Maryland, College Park. He interned with the Visual Studio team at Microsoft and is a former IBM Extreme Blue intern. He currently works on pen-based user interface research and enjoys writing .NET GNOME desktop applications on top of Mono in Linux.

Trademarks | My developerWorks terms and conditions