1. Documenting your project using the Eclipse help system

Build easy-to-use and searchable help documentation

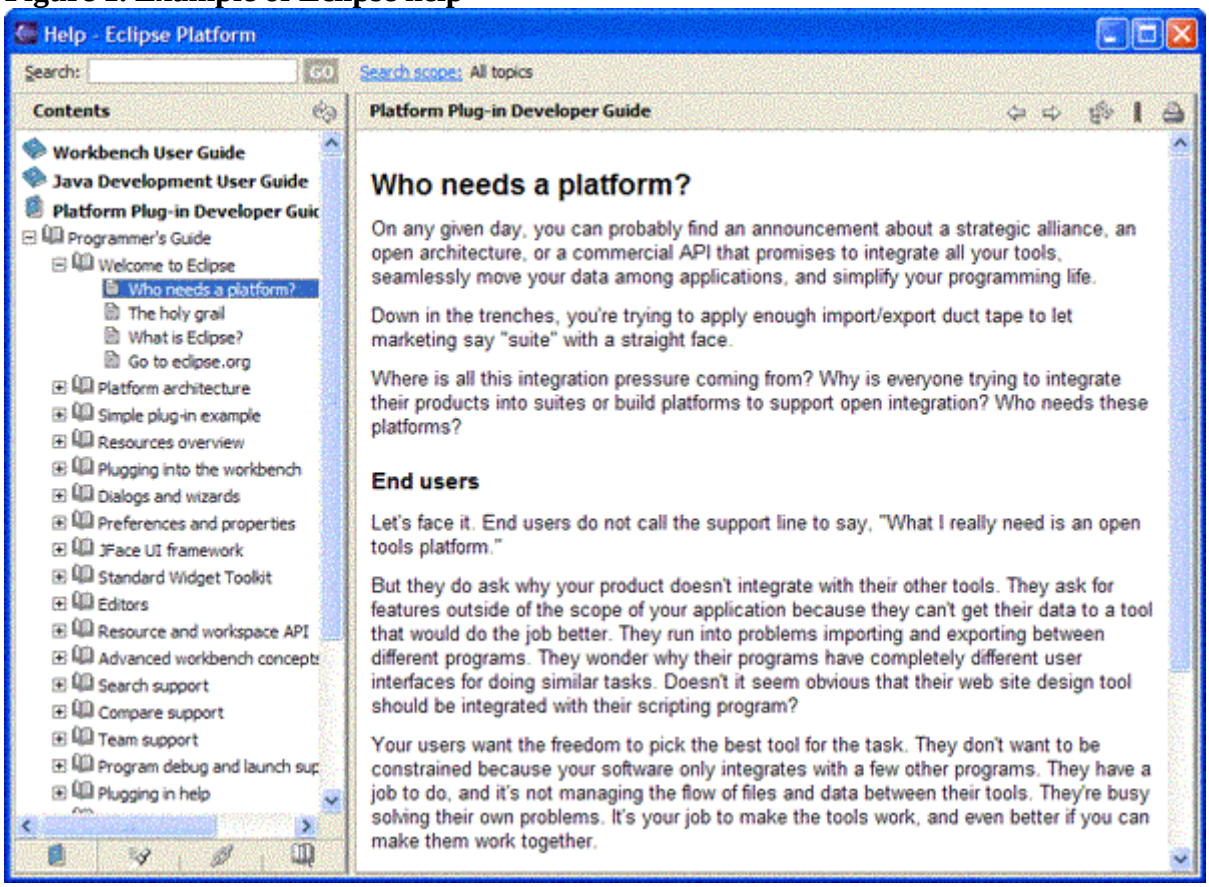
Arthur Barr, Software engineer, IBM

Summary: The Eclipse Platform, which provides a very powerful IDE, includes its own help system based on an XML table of contents referencing HTML files. What isn't immediately obvious is that you don't have to write Eclipse plug-ins to use it. Any project can use a cut-down version of the platform to provide professional, easy-to-use, and searchable documentation. This documentation system has been successfully used on a number of IBM projects, including those as large as the IBM® WebSphere® Application Server.

Tags for this article: documentation, eclipse, research, xml

When you access the Eclipse help system (through **Help > Help Contents**), you are actually starting up an embedded Apache Tomcat server. A window based on a Web browser is then opened, pointing to the correct page on that server (see Figure 1). Documentation is provided with a collapsible index on the left side and HTML documentation on the right, and can be searched (thanks to the Apache Lucene search engine). Since Tomcat is used, you are not limited to HTML. For example, you can use JSPs to make your documentation change dynamically (though we will discuss later a possible reason to avoid doing this).

Figure 1. Example of Eclipse help



The Hello, World of documentation plug-ins

Documentation is split into "books," and you can have as many books as you like in one instance of the help system. Each book is written as an Eclipse plug-in, but thankfully, the work involved here is minimal. To write a simple plug-in, you will need a plugin.xml file to describe your plug-in, which should look like Listing 1.

Listing 1. Plug-in definition

```
<plugin name="Sample Documentation Plug-in" id="com.ibm.sample.doc"
   version="1.0.0" provider-name="IBM">
   <extension point="org.eclipse.help.toc">
      <toc file="toc.xml" primary="true" />
   </extension>
</plugin>
```

Change the plug-in's `name`, `id`, `version`, and `provider-name` to values appropriate to your project. The extension point of `org.eclipse.help.toc` identifies this as a plug-in to the help system. The file toc.xml is referenced as being the table of contents for this plug-in. This file will provide the data for the hierarchical information in the left pane of the Eclipse help window. A simple file contains something like that shown below.

Listing 2. Table of contents definition

```
<toc label="Sample Documentation">
    <topic label="My Section" href="mySection.html">
        <topic label="Foo" href="foo.html"/>
        <topic label="Bar" href="bar.html"/>
    </topic>
</toc>
```

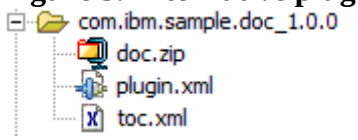
Packaging the plug-in

Each topic element is represented in the final documentation by an entry in the navigation list. These topics can be nested (they can contain more topics), and each one points to an HTML or JSP file. Once you've done this, all you need to do is package everything in the structure shown in Figure 2 (notice that the plug-in directory name matches the `id` and `version` attributes of the plug-in defined in the plugin.xml).

Figure 2. Plug-in directory structure

As a convenience, and to reduce file size, Eclipse allows you to keep all your actual documentation (the HTML files) in a ZIP file called doc.zip, so you could use the directory structure shown in Figure 3.

Figure 3. Alternative plug-in directory structure

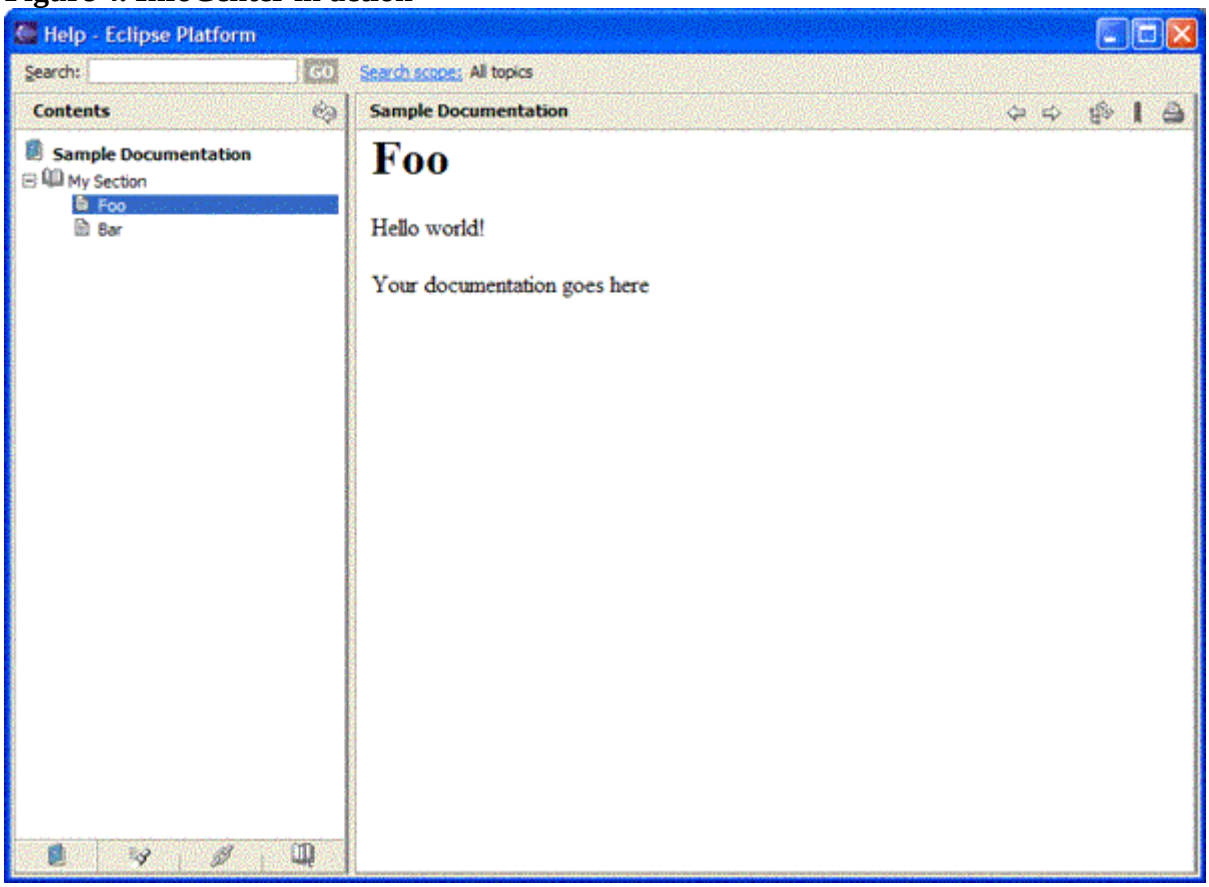


Viewing your documentation

The easiest way to test your plug-in is to simply drop the entire directory (as above) into the plugins directory of an installed Eclipse Platform, then launch Eclipse and select **Help > Help Contents**. You will get a help window with your plug-in added (similar to the one in Figure 1).

Using the IDE is all very well for testing, but to be useful without the IDE, the documentation needs to be more accessible, so what we really want is to run a process in the background that lets us connect to it with a browser. This mode of operation is known as an InfoCenter (see Figure 4). Instructions for starting an InfoCenter process (basically Apache Tomcat) are included with the Eclipse help system documentation (see Resources). Note that there also instructions on how to pare down the Eclipse system to give you just the bits you need.

Figure 4. InfoCenter in action



Handling large tables of contents

If your project has more than a few people working on it or has a large documentation set, updating a single table of contents (toc.xml) file can become impractical. You can change this by adding a `link` element into your topic in the main toc.xml file (see Listing 3 for an example).

Listing 3. Table of contents definition

```
<toc label="Sample Documentation">
    <topic label="My Section" href="mySection.html">
        <topic label="Foo" href="foo.html"/>
        <topic label="Bar" href="bar.html">
            <link toc="bar-toc.xml" />
        </topic>
    </topic>
</toc>
```

The file bar-toc.xml is just another table of contents, and should take exactly the same format as any other toc.xml file. When the documentation is viewed, there will be no difference between using this method and simply including the additional `topic` elements directly.

Back to top

Generating a stand-alone documentation set

Of course, using the Eclipse help system is all well and good if you don't mind distributing the 20-plus MB of code required, but this isn't realistic for smaller projects. Hosting an InfoCenter on a central server allows people to connect remotely. People receive all the benefits of using the Eclipse help system (such as searching), but people without connectivity are left stranded. So, in addition to using a hosted InfoCenter, it's useful to include the plain HTML in a downloadable package. As long as you haven't used any server-side technologies such as JSPs, you can easily generate an HTML table of contents to replace the XML one used by Eclipse. Which is why we have eXtensible Stylesheet Language Transformations (XSLT).

XSLT is a technology used to transform one form of XML to another, such as XHTML (a stricter, XML version of HTML). XSLT provides a rich and powerful language to perform transformations, and is the topic of many books and articles on its own, so we won't go into detail here. Listing 4 shows an example of a simple transformation of a toc.xml file, rendering the entries as nested HTML lists. Note that this particular transformation creates a single HTML file for the contents of the whole documentation set, which will be unwieldy for large numbers of files. Therefore, this XSLT will not work if you have split your table of contents across multiple files.

Listing 4. Sample XSLT to generate HTML table of contents

```
<?xml version="1.0"?>
<xsl:stylesheet
   version="1.1"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="html" indent="no" encoding="ISO-8859-1" />

<xsl:template match="toc">
   <html>
      <head />
      <body>
         <h1><xsl:value-of select="@label" /></h1>
         <ul>
            <xsl:apply-templates />
         </ul>
      </body>
   </html>
</xsl:template>

<xsl:template match="topic">
   <li>
      <xsl:choose>
         <xsl:when test="@href">
            <!-- Only add a hyperlink when there is something to link
to ->
            <xsl:element name="a">
               <xsl:attribute name="href">
                  <xsl:value-of select="@href" />
                  </xsl:attribute>
               <xsl:value-of select="@label" />
            </xsl:element>
```

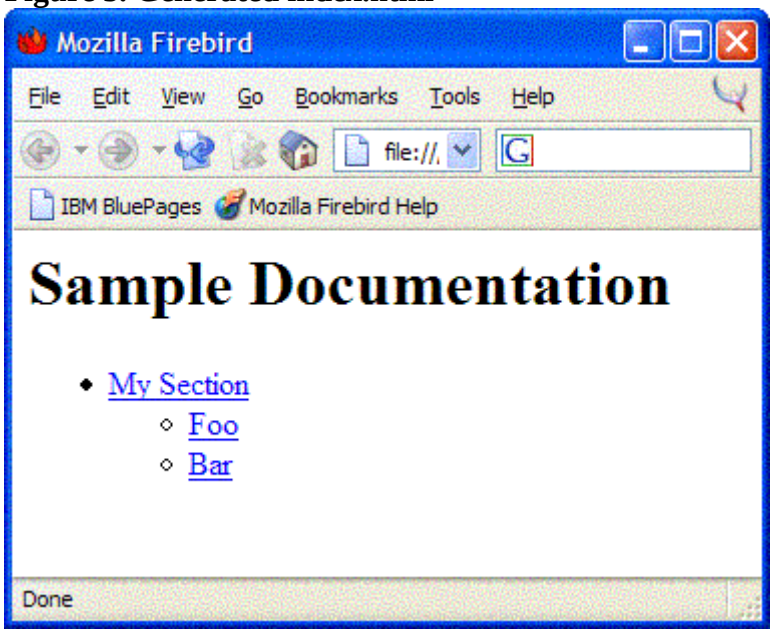
```
         </xsl:when>
         <xsl:otherwise>
            <xsl:value-of select="@label" />
         </xsl:otherwise>
      </xsl:choose>

      <!-- If there are any nested topics, then start a new sub-list -
>
      <xsl:if test="descendant::topic">
         <ul>
            <xsl:apply-templates/>
         </ul>
      </xsl:if>
   </li>
</xsl:template>

</xsl:stylesheet>
```

Processing the toc.xml file through an XSLT processor, such as Apache Xalan using the above XSLT, yields an HTML file that looks something like Figure 5, when viewed with a browser.

Figure 5. Generated index.html



Back to top

Conclusion

Using the Eclipse help system is a fairly painless way to develop professional-looking, searchable documentation that will amaze your friends and colleagues. If you don't have a requirement for a stand-alone documentation set, then you don't even need to go near

XSLT; you can write just two simple XML files and be on the road to documentation happiness. Off you go.

2. Installing the help system as an infocenter

You can allow your users to access the help system over the Internet or an intranet, by installing the infocenter and the documentation plug-ins on a server. Clients view help by navigating to a URL, and the help system is shown in their web browser. The infocenter help system can be used both for client applications and for web applications, either of which can have their help accessed remotely. All features of help system except infopops and active help are supported.

The infocenter help system allows passing number of options that can be used to customize various aspects of the infocenter. The following options are supported:

- **-eclipsehome** *eclipseInstallPath* - specifies Eclipse installation directory. This directory is a parent to "plugins" directory and eclipse executable. The option must be provided, when current directory from which infocenter is launched, is not the same as Eclipse installation directory.
- **-data** *instanceArea* - specifies a path that Eclipse can use to write instance data. The value can be an absolute path of a directory, or a path relative to Eclipse installation directory. The option must be provided when Eclipse is installed in the read only location, or has been customized to override osgi.instance.area or osgi.instance.area.default properties.
- **-host** *helpServerHost* - specifies host name of the interface that help server will use. It overrides host name specified in the application server plugin preferences.
- **-port** *helpServerPort* - specifies port number that help server will use. It overrides port number specified in the application server plugin preferences.
- **-locales** *localeList* - specifies a list of locales that infocenter will recognize and provide a customized content for. If the option is not specified, infocenter will build navigation, and index documents for each preferred locale of the browsers accessing the infocenter. When the option is present, locales from browser requests will be matched with locales in the list. If browser preferred locale does not exist in the list, but its language part does, it will be used. Subsequently, additional browser locales in decreased order of preference will be matched against the list. If none of the browser locales (or its language part) matches any locale on the list, the client will be served content in the default locale - server locale or locale passed with -nl option. For example using options

  ```
  -nl en -locales de en es fr it ja ko pt_BR zh_CN zh_TW
  ```

  will cause infocenter operating in 10 locales. All other locales will receive content for en locale.

- **-dir ltr** or **-dir rtl** - forces left-to-right or right-to-left rendering direction of help UI in the browser for all languages. By default direction is determined based on the browser locale.
- **-noexec** - indicates that Eclipse executable should not be used. You may need to use this option when running on a platform for which Eclipse executable is not available.
- Additionally, most options accepted by Eclipse executable can be passed. They are especially useful during debugging and for applying customization to Eclipse. For example, passing options

  ```
  -vmargs -Xmx256M
  ```

  increases memory available to the infocenter and will allow serving a larger book collection.

3. Installation/packaging

These steps are for the help system integrator and are not meant to address all the possible scenarios. It is assumed that all your documentation is delivered as Eclipse plug-ins and, in general, you are familiar with the eclipse help system.

1. Download the Eclipse Platform Runtime Binary driver from www.eclipse.org.
2. Install (unzip) the driver in a directory, *d:\myApp*. This will create an eclipse sub-directory, d:\myApp\eclipse that contains the code required for the Eclipse platform (which includes the help system).

4. How to start or stop infocenter from command line

The org.eclipse.help.standalone.Infocenter class has a main method that you can use to launch infocenter from a command line. The command line arguments syntax is:

```
-command start | shutdown | [-eclipsehome eclipseInstallPath] [-data
instanceArea] [-host helpServerHost] [-locales localeList] [-port
helpServerPort] [-dir rtl] [-noexec] [platform options] [-vmargs
JavaVMarguments]
```

To start an infocenter on port 8081 issue a start command by running

```
java -classpath
d:\myApp\eclipse\plugins\org.eclipse.help.base_3.1.0.jar
org.eclipse.help.standalone.Infocenter -command start -eclipsehome
d:\myApp\eclipse -port 8081
```

To shut the infocenter down issue a shutdown command by running

```
java -classpath
d:\myApp\eclipse\plugins\org.eclipse.help.base_3.1.0.jar
org.eclipse.help.standalone.Infocenter -command shutdown -eclipsehome
d:\myApp\eclipse
```

5. Using the infocenter

Start the web server. Point a web browser to the path "help" web application running on a port specified when starting the infocenter. On the machine the infocenter is installed, this would be http://localhost:8081/help/.

6. How to start or stop infocenter from Java

When including infocenter as part of another application, it may be more convenient to start it and stop using Java APIs instead of using system commands. Follow the steps if it is the case:

1. Make sure d:\myApp\eclipse\plugins\org.eclipse.help.base_3.1.0.jar is on your app classpath. The class you use to start, and shut down the infocenter isorg.eclipse.help.standalone.Infocenter.
2. Create an array of String containing options that you want to pass to the infocenter. Typically, the eclipsehome and port options are needed.

   ```
   String[] options = new String[] { "-eclipsehome",
   "d:\\myApp\\eclipse" , "-port", "8081" };
   ```

3. In your application, create an instance of the Help class by passing the options.

   ```
   Infocenter infocenter = new Help(options);
   ```

4. To start the help system:

   ```
   helpSystem.start();
   ```

5. To shut the infocenter down:

   ```
   helpSystem.shutdown();
   ```

7. Making infocenter available on the web

Eclipse contains a complete infocenter and does not require other server software to run. However, in unsecure environment like Internet, it is recommended infocenter is not accessed directly by clients, but is made available through an HTTP server or an application server. Most servers come with modules or servlets for delegating certain request to other web resources. For example, one may configure a proxy module of Apache HTTP Server to redirect requests made to *http://mycompany.com/myproduct/infocenter* to *http://internalserver:8081/help* that runs an infocenter. Adding the lines

```
LoadModule proxy_module modules/ApacheModuleProxy.dll
ProxyPass /myproduct/infocenter http://internalserver:8081/help
ProxyPassReverse /myproduct/infocenter http://internalserver:8081/help
```

to conf/httpd.conf file of Apache server running mycompany web site accomplishes this.

Some versions of Apache HTTP server, may contain AddDefaultCharset directive enabled in configuration file. Remove the directive or replace with

```
AddDefaultCharset Off
```

to have browsers display documents using correct character set.

8. Running multiple instance of infocenter

Multiple instances of infocenter can be run on a machine from one installation. Each started instance must use its own port and be provided with a workspace, hence `-port` and `-data` options must be specified. The instances can serve documentation from different set of plug-ins, by providing a valid platform configuration with `-configuration` option.

If `-configuration` is not used and configuration directory is shared among multiple infocenter instances, with overlapping set of locales, it must be ensured that all search indexes are created by one infocenter instance before another instance is started. Indexes are saved in the configuration directory, and write access is not synchronized across infocenter processes.

9. [Optional] Installing a minimal set of plug-ins

The infocenter does not require the entire Eclipse Platform package. It is possible to run the infocenter with the following plug-ins (located in the eclipse\plugins directory):

```
org.apache.lucene
org.eclipse.core.runtime
org.eclipse.help
org.eclipse.help.appserver
org.eclipse.help.base
org.eclipse.help.webapp
org.eclipse.osgi
org.eclipse.tomcat
org.eclipse.update.configurator
```

Some documentation plug-ins may have dependencies on other plug-ins, usually by specifying required plug-ins in their plugin.xml. The dependent plug-ins need to be installed on the infocenter as well. Additionally, plug-ins that were designed for earlier than 3.0 version of Eclipse implicitly require an `org.eclipse.core.runtime.compatibility` being present plug-in to work.

Infocenter plug-ins can be updated without restarting the infocenter, using commands explained in Updating a running infocenter from command line topic. To use this functionality, the minimal set of plug-ins must include `org.eclipse.update.core` plug-in.

See Help System Preferences for more information on customizing help system.