Generating Help for Eclipse Plug-ins

This document describes the process used by the TechPubs department at Agitar Software (in Mountain View, CA) to generate help for an Eclipse plug-in. Feel free to use this information for your own projects. If you have questions, you can write to me (Martha Kolman-Davidson, martha@agitar.com or editrix@nemasys.com).

To support the integration of Agitator 2.0 with Eclipse, TechPubs has created Eclipse-style help from FrameMaker source files, converted to HTML using WebWorks Publisher Pro 2003 (WWP). Because WWP does not provide templates that generate help using the Eclipse-specific file formats for the TOC and for context-sensitive help topics, extra work was required to generate help in the required format, starting with WWP's Dynamic HTML template.

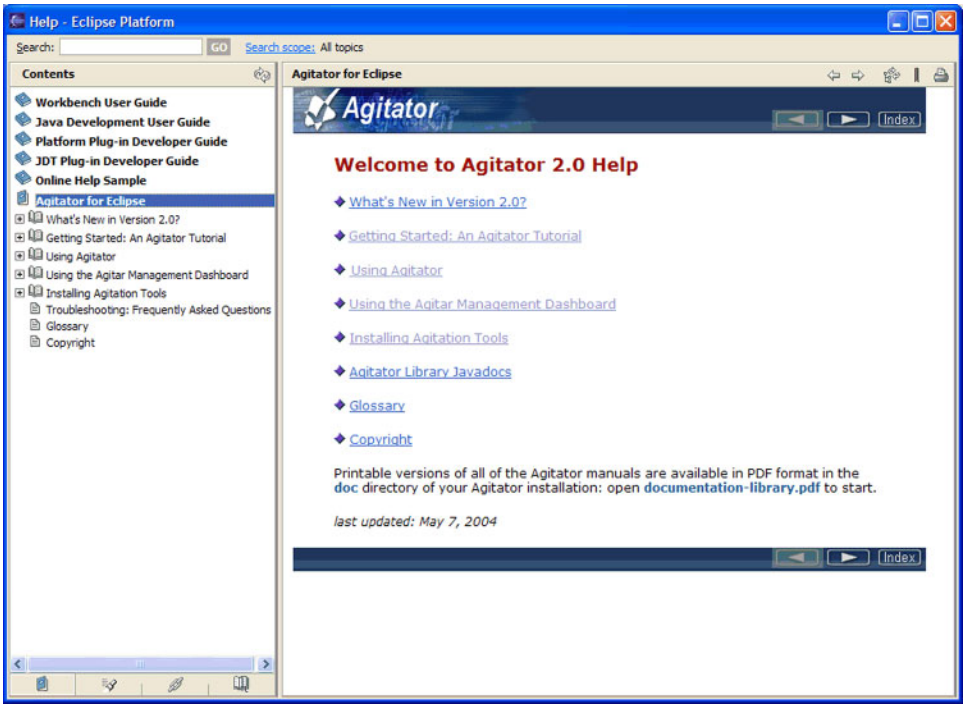
As part of the process, we chose to require specific FrameMaker formatting in order to simplify the WWP processing for context-sensitive help topics. Rather than delving deeply enough into the WWP macro language to buffer the content of these topics in an iterator and write them at the end of the generation process, we chose to add extra FrameMaker constructs and to map those directly to lines of generated XML text (using **@WRITE** macros).

The following topics describe the process we used to create the WWP template to generate Eclipse help:

About Eclipse Help

Help for Eclipse plug-ins uses HTML pages, displayed in Eclipse's custom help browser, with a Table of Contents pane on the left and help topic content on the right. The Table of Contents combines entries for all plug-ins, using content from XML files in a specific format (see "Generating the Table of Contents" on page 3). The left pane also shows results of full-text searches. Eclipse help provides no built-in support for "back of the book" type indexes, only full-text search.

The generated help plug-in for Agitator looks like this:



Eclipse uses special popup topics (known as *infopops*) for context-sensitive help, invoked only by F1, and not by **Help** buttons in dialog boxes. Each infopop has a brief description of the view, editor, or dialog box that has focus, followed by one or more links to topics elsewhere in the corresponding plug-in's help (see "Supporting Context-Sensitive Help (Infopops)" on page 5).

NOTE: Using build M8 of Eclipse 3.0, I notice that if I press F1 in a view or editor pane, and click a link from the infopop, the proprietary help browser displays the topic. If, however, I press F1 from a dialog box and then click a link from the infopop, my default browser (in this case, Mozilla 1.6) displays the help topic, including the TOC pane.

WebWorks Conversion Template Changes

To generate help for Eclipse, TechPubs started with WWP's Dynamic HTML template, which generates individual HTML pages for each topic, with bare-bones navigation through the topics in the sequence they appear in the FrameMaker book.

The following customization has been made to this template:

- Modified **normal.asp** to add page headers and footers to look like the corresponding topics in the JavaHelp for Agitator 1.1.
- Changed the default output directory from **Output** to **html** to make it easier to package the help as an Eclipse plugin (for more, see "Packaging and Installing the Generated Help" on page 9).

  NOTE: It turns out this isn't necessary, but since I've changed it, I'll leave it that way.
- Removed the **TOC** navigation button from **normal.asp**, since Eclipse displays the TOC in a separate pane; I left the **Index** button in and made it match the Agitator look. To make this work, I made the following changes:
  - Modified **BP80PageASP_Navigation** to remove image for TOC button and to point to the **Next**/**Previous** images used in the JavaHelp version.
  - Modified **BP80PageASP_SetPrevNext** to return **false** if the previous page is the TOC, so that the opening topic won't have a "previous" page to navigate to.
- Made the changes described in the next section, "Generating the Table of Contents," to generate **toc.xml** in the format expected by Eclipse.

Generating the Table of Contents

Eclipse uses the following format for the Table of Contents, by default in a file called **toc.xml**:

```
<toc label="Tasks">
    <topic label="Plain Stuff">
        <topic label="Task1" href="html/tasks/task1.html"/>
        <topic label="Task2" href="html/tasks/task2.html"/>
    </topic>
    <topic label="Fun Stuff" >
        <topic label="Task3_1" href="html/tasks/task3_1.html"/>
        <topic label="Task3_2" href="html/tasks/task3_2.html"/>
    </topic>
</toc>
```

To generate a TOC in this format, no changes were needed to the FrameMaker source. All changes happened in the WWP template; specifically:

- Changed the name of the generated TOC file from **toc.html** to **toc.xml** (in WWP project properties).

- Modified the following TOC building blocks based on corresponding ones in the JavaHelp template:
  - **BP80TOCWriteIterator**, to generate individual entries for the Eclipse TOC. Here's the code, with Eclipse-specific stuff in bold:

```
$COMMENT(
  Emit Eclipse TOC in valid XML format--modified from JavaHelp template.
);\

\
@INC_COUNTER(var_TOCIteratorDepth);\
\
$LOOP($GET_COUNTER(var_TOCIteratorDepth);, BP80_Spaces);\
<topic\
 label="$KEYEDLISTITER_ARRAY(3);" \
 href="$KEYEDLISTITER_ARRAY(2)[$CHARSET;, " \" " , "
BP80_EscapeIterator" , replace];" \
\
$IF_GREATER($GET_KEYEDLISTTOTAL(var_TOC,
$KEYEDLISTITER_ARRAY(1););, 0,
  >
$ITERATE_KEYEDLIST(BP80TOC_WriteIterator, var_TOC,
$KEYEDLISTITER_ARRAY(1););\
$LOOP($GET_COUNTER(var_TOCIteratorDepth);, BP80_Spaces);\
</topic>
,
  \ />
);\
\
@DEC_COUNTER(var_TOCIteratorDepth);\
```

  - **BP80TOCWrite**, to write the **toc.xml** file. Here's the code:

```
$COMMENT(
  Writes Eclipse TOC XML file--modified from JavaHelp template.
);\
\
$BP80TOC_Collapse;\
\
@WRITE(overwrite, host, $GET_PROJECTPROP(TOCName);,
<?xml version='1.0' encoding='$CHARSET;' ?>
<?NLS TYPE="org.eclipse.help.toc"?>
<toc label="$PROJECTNAME;" topic="$UMEclipseOpeningTopic;">
$ITERATE_KEYEDLIST(BP80TOC_WriteIterator, var_TOC, 0);</toc>
);\
```

    I created the user macro **UMEclipseOpeningTopic** with the value of the first generated topic in the project:

    `help-opening-topic-1.html`

    The purpose of this is to display the opening topic when a user selects **Agitator for Eclipse** from the combined Eclipse help TOC, rather than displaying no topic and having this be the first topic inside the Agitator help.

    NOTE: There is probably a more dynamic way to set the name of the opening topic to display; for now, I've hard-coded it in the user macro.

Supporting Context-Sensitive Help (Infopops)

Eclipse help uses popup windows, known as infopops, to display context-sensitive help. Users press F1 to get context-sensitive help for the Eclipse view, editor, or dialog box that has focus. Each infopop has a brief description of the GUI control, followed by a list of links to topics in the help for the associated plug-in. For example:

NOTE: For each infopop entry in the FrameMaker document, I'm using a sidehead paragraph (mapped to **NoOutput** in the WWP template) to identify which GUI control (Eclipse view or dialog box) the infopop is describing.

Required Output

Eclipse expects the definitions of infopop topics to be in a file called **contexts.xml**, with the following format:

```
<contexts>
   <context id="panic_button">
      <description>Brief description of this control.</description>
      <topic href="file_name_link1.html" label="Link1 Topic Title"/>
      <topic href="file_name_link2.html" label="Link2 Topic Title"/>
   </context>
   . . .
</contexts>
```

FrameMaker Setup

The modified WWP template can generate topics in this format using the following FrameMaker constructs:

- Markers:
  - **EclipseContextsStart** marker, to trigger deleting any existing **contexts.xml** file and write the opening lines:

    ```
    <?xml version='1.0' encoding='ISO-8859-1' ?>
    <contexts>
    ```

    The template ignores the content of this marker.
  - **EclipseContext** marker, analogous to **TopicAlias** markers.

    Encountering this marker opens a **context** element in the **contexts.xml** file, with the marker text as the value of the **id** attribute.

- **EclipseContextsEnd** marker, to trigger adding the closing **</contexts>** tag.
  The template ignores the content of this marker.

- **EclipseTopic** paragraph tag, whose content goes into a **description** element inside the **context** element.
- **EclipseLink** paragraph tag, with a cross-reference to a related topic, which becomes a **topic** element inside the context element.
  - The text of this paragraph would be the value of the **label** attribute.
  - The destination of the cross-reference would be the value of the **href** attribute.
- Separate paragraph tag **EclipseLinkLast**, to add the final link and also close the **context** element.

Using the FrameMaker Markers and Paragraph Tags

To create the source for Eclipse infopops, place the text at the end of an existing Frame chapter file, *not* in a separate file, and tag the entire section with the **Eclipse** condition tag. If it's in a separate file, WWP will generate an empty topic corresponding to that file, even if there's no content in it to populate the topic page.

Place all of the infopop topics together, in a block with nothing else in it, following these guidelines:

1. At the beginning of the infopop section, in an empty **Body** paragraph, place an **EclipseContextsStart** marker.

   You can put any text you want in the marker, because WWP only uses the marker as a flag and ignores the marker text.

2. For each infopop topic:
   a. As internal documentation, start with a **SideHead Text** paragraph, describing what view or dialog box this infopop is associated with.

      In the WWP template, **SideHead Text** paragraphs are mapped to **No Output**.

   b. Create a paragraph tagged **EclipseTopic**, with a brief description of the view or dialog box.

   c. Go back to the beginning of the **EclipseTopic** paragraph and add an **EclipseContext** marker with the ID for this view or dialog box—coordinate with the GUI developer responsible for that construct to get the ID.

   d. Add one or more **EclipseLink** paragraphs with cross-references to relevant topics elsewhere in the help.

   e. Use the last **EclipseLinkLast** paragraph tag for the last cross-reference paragraph. If there's only one cross-reference, use **EclipseLinkLast** for it.

3. At the end of the infopop section, in an empty **Body** paragraph, place an **EclipseContextsEnd** marker.

   You can put any text you want in this marker also, because WWP uses it as only a flag and ignores the marker text.

WWP Macros

To process the FrameMaker constructs listed in the previous section, the WWP template contains the following marker and paragraph macros. Notice the hard returns before the closing parentheses of each **@WRITE** macro, to ensure line breaks in the generated XML.

The details appear in the following subsections:

- Markers
- Paragraph Styles
- User Macros

Markers

The WWP template has the following new marker macros:

- **EclipseContextsStart** marker:

```
$COMMENT(
  Clears existing contexts.xml file and starts a new one.
);\
@DELETE(host,$UMEclipseContextsFileName;);\
@WRITE(append, host, $UMEclipseContextsFileName;, <?xml
version='1.0' encoding='ISO-8859-1' ?>
);\
@WRITE(append, host, $UMEclipseContextsFileName;, <contexts>
);
```

- **EclipseContext** marker:

```
@WRITE(append, host, $UMEclipseContextsFileName;, \ \ <context
id="$DATA;">
);
```

- **EclipseContextsEnd** marker:

```
$COMMENT(
  Adds final closing tag to $UMEclipseContextsFileName; file.
);\
@WRITE(append, host, $UMEclipseContextsFileName;, </contexts>);
```

Paragraph Styles

The WWP template has the following new paragraph style macros:

- **EclipseTopic** paragraph:

```
@WRITE(append, host, $UMEclipseContextsFileName;, \ \ \ \
<description>$DATA(raw);</description>
);
```

- **EclipseLink** paragraph:

```
@WRITE(append, host, $UMEclipseContextsFileName;,
$UMEclipseInfopopLink;
);
```

- **EclipseLinkLast** paragraph:

```
@WRITE(append, host,$UMEclipseContextsFileName;,
$UMEclipseInfopopLink;
);\
@WRITE(append, host,$UMEclipseContextsFileName;, \ \ </context>
);
```

User Macros

The marker and paragraph styles use the following new user macros:

- **UMEclipseContextsFileName**, with the name of the XML file containing the infopop topic definitions, defined as:

```
contexts.xml
```

- **UMEclipseInfopopLink**, with the code to generate the XML code for infopop links (following the description in each infopop entry), defined as:

```
\ \ \ \ <topic href="$LINKFILE(html, basename);#wp$LINKTAG;"\
label="$DATA(raw);"/>
```

Order of Processing

Here's the basic flow in pseudo-code.

Start writing Eclipse Infopop topics.

1. Encounter an **EclipseContextsStart** marker:
   a. Delete existing **contexts.xml** file.
   b. Write the XML declaration and opening **<contexts>** tag in a brand-new file.

Process each Eclipse Infopop topic.

2. Encounter **EclipseContext** marker:
   a. Write the opening **<context>** tag with marker text as **id** attribute.
   b. Encounter an **EclipseTopic** paragraph tag; write the **<description>** element with the paragraph contents.
   c. Encounter an **EclipseLink** paragraph tag, then write a **<topic>** element with the link destination and the paragraph text.

d. Repeat step 2c until encountering **EclipseLinkLast** paragraph tag, then write the last **<topic>** element, followed by the closing **</context>** tag.

3. Repeat step 2 until encountering an **EclipseContextsEnd** marker.

Finish writing Eclipse Infopop topics.

4. Write the closing **</contexts>** tag.

NOTE: This flow assumes that all Eclipse infopop source is in a block together, with nothing else intervening. It is the responsibility of the writer to ensure that this structure is maintained.

Packaging and Installing the Generated Help

For Agitator 2.0 the help will be packaged as a separate plug-in. We can write the **plugin.xml** file once by hand instead of trying to have WWP generate it.

NOTE: The following link http://www-106.ibm.com/developerworks/opensource/library/os-echelp/ was the most helpful in figuring out how to package and install the help plug-in.

Plug-in Definition

The plug-in description contains (*with substitutions for the right values where needed*), in a file called **plugin.xml**:

```
<?xml version="1.0"?>
<plugin name="Agitator Help"
       id="com.agitar.help"
       version="2.0"
       provider-name="Agitar.com">
   <runtime/>
   <extension point="org.eclipse.help.toc">
      <toc file="toc.xml" primary="true"/>
   </extension>
   <extension point="org.eclipse.help.contexts">
      <contexts name="contexts.xml"/>
    </extension>
</plugin>
```

The way to provide the help files is to zip the generated files into a **doc.zip** file. This file goes in the plug-in directory (named ***id_version***, based on the attributes in on the **plugin** element in **plugin.xml**). So, the plug-in home directory, in this case **com.agitar.help_2.0**, should contain:

- **plugin.xml**
- **toc.xml**
- **contexts.xml**
- **doc.zip** with the HTML files and supporting files

WWP Support

To package the generated help, I've a created a set of WWP building blocks and user macros, and added the building block **BAZipEclipseFilesForPlugin** to the end of **BZOnConvertAllEnd** in the customized WWP template.

Building Blocks

The building block **BAZipEclipseFilesForPlugin** copies the generated TOC and contexts files to a temporary directory called **plugin** (a sibling of the WWP output directory), zips the remaining files in the output directory to a file called **doc.zip**, and then moves **doc.zip** to the **plugin** directory. After testing, the contents of the **plugin** directory can be copied to the correct subdirectory of the Eclipse **plugin** directory.

Here's the code:

```
$COMMENT(
  Move toc and contexts files to plugin dir.
);\
@MOVE(overwrite, host, toc.xml, ..$SEP;plugin$SEP;toc.xml);\
@MOVE(overwrite, host, $UMEclipseContextsFileName;,
..$SEP;plugin$SEP;$UMEclipseContextsFileName;);\

$COMMENT(
  Zip remaining files into doc.zip.
);\
@EXECUTE($UMZipFilesCommand;);\

$COMMENT(
  Move doc.zip to plugin dir.
);\
@MOVE(overwrite, host, doc.zip, ..$SEP;plugin$SEP;doc.zip);
```

User Macros

And the supporting user macros:

- **UMEclipseContextsFileName**, described in "User Macros" on page 8
- **UMZipFilesCommand**, defined as:

  ```
  c:\\cygwin\\bin\\zip -r doc \*
  ```

Installing the Help Plug-in

To install the plug-in:

1. Close Eclipse.
2. Place the directory with the plug-in contents in the **plugin** directory of an Eclipse installation.
3. Restart Eclipse.
4. Select **Help>Help Contents**.

   If all went well, the help plug-in appears in the Eclipse help browser. (It took me a few tries to get the help to show up, mostly related to the name I chose for the plug-in directory under **eclipse/plugins**.)