Copyright © 2001, 2003 Object Technology International, Inc.

Eclipse Corner Article

Help – Part 1
Contributing a little help

Summary
The Eclipse Platform's help system defines two extension points (`"toc"` and `"contexts"`) that allow individual plug-ins to contribute online help and context-sensitive help for their components. In this article we will investigate the `"toc"` extension point and how you can use it to contribute documentation for your plug-in.

By Greg Adams, OTI and Dorian Birsan, IBM
Updated August 9, 2002

Editor's note: This article describes the help system for Eclipse release 2.1, which differs in minor ways from the previous Eclipse release. If you are working with Eclipse release 1.0, you should consult the original version of this article.

Introduction

The Eclipse Platform's help system allows you to contribute your plug-in's online help using the org.eclipse.help.toc extension point. You can either contribute the online help as part of your code plug-in or provide it separately in its own documentation plug-in. This separation is beneficial in those situations where the code and documentation teams are different groups or where you want to reduce the coupling/dependency between the documentation and code. The Platform's help facilities provide you with the raw building blocks to structure and contribute your help without dictating the structure or granularity of documentation. The Platform does however provide and control the integrated help viewers thus ensuring seamless integration of your documentation.

The org.eclipse.help.toc contribution specifies one or more associated XML files that contain the structure of your help and its integration with help contributed by other plug-ins. In the remainder of this article we will create a documentation plug-in, so by the time your're done you can browse your own documentation using the Eclipse Help System.

Making the plug-in and content

A content author supplies one or more HTML files containing the actual documentation. There are no restrictions imposed by the Platform on the granularity of the HTML files or on their content. We will start by assuming that you already have some HTML files that you want to integrate into the Eclipse Platform. Let's assume your content is arranged into the following directory tree.

```
html/
   overview.html
   concepts/
       concept1.html
       concept1_1.html
       concept1_2.html
   tasks/
       task1.html
       task2.html
       task3_1.html
       task3_2.html
   ref/
       ref1.html
       ref2.html
```

Create a plug-in directory called org.eclipse.helparticle and place the above `html/` sub-tree into it. Now create an initial plugin.xml file with the following content:

```
<?xml version="1.0"?>
```

```
<plugin
   name = "Online Help Sample"
   id = "org.eclipse.helparticle"
   version = "1.0"
   provider-name = "Eclipse.org>
</plugin>
```

To help you get started here is a zip file containing the above initial plug-in structure . This plug-in manifest file doesn't actually integrate anything yet but soon we will add our contributions to it.

Topics & HTML Content

Now that we have our sample content files we are ready to create our **toc** file. A toc file defines the key entry points into the HTML content files by defining labeled topics mapped to the individual HTML files. A toc file acts like a table of contents for a set of HTML content. Teams migrating onto the Eclipse Platform can quickly reuse existing HTML documentation by defining entry points into their existing documentation via the toc file. A plug-in can have one or more toc files. Toc files are sometimes referred to as navigation files since they describe how to navigate the HTML content. We have three main content areas, concepts, tasks and reference. Our obvious choices are one big toc file, or a toc file for each main content area. Keep in mind this decision is ours to make, and is not a decision the Platform dictates to us. If we were “really” writing our documentation we would have a larger number of files so, with that in mind we will try and keep things manageable by creating a toc file for each of the three content areas as follows:

toc_Tasks.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>
```

toc_Concepts.xml

```
<toc label="Concepts">
    <topic label="Concept1" href="html/concepts/concept1.html">
        <topic label="Concept1_1" href="html/concepts/concept1_1.html"/>
        <topic label="Concept1_2" href="html/concepts/concept1_2.html"/>
    </topic>
</toc>
```

toc_Ref.xml

```
<toc label="Refs">
    <topic label="Ref1" href="html/ref/ref1.html"/>
    <topic label="Ref2" href="html/ref/ref2.html"/>
</toc>
```

Topics are contributed as part of the `<toc>` container element. A `<topic>` can be a simple link to content (e.g. Task1) or a hierarchical grouping of sub-topics (e.g. Fun Stuff), or both (e.g. Concept1). When used as a link, the argument to href is assumed to be relative to the current plug-in. Later we will modify the plugin.xml to add the actual contributions consisting of these files. Notice that the hierarchical structure of topics is not the same as the file system structure: all the "Concepts" topics files in one directory, but the table of contents nests them differently.

Creating a book

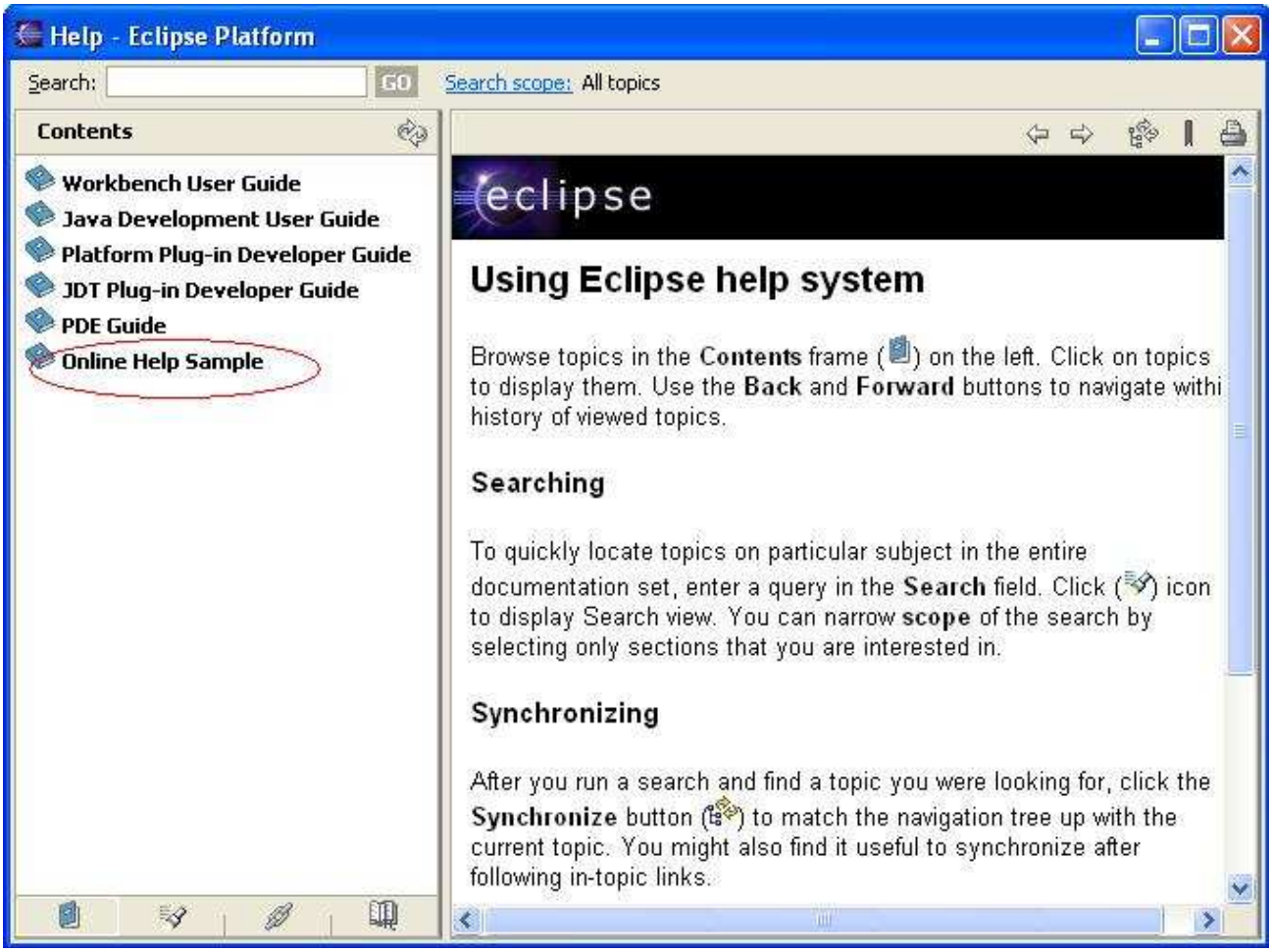
Now that we have our raw content and toc files it’s time to create our book. A book is just table of contents. In fact, we could make a book out of each of the three toc files above, but we will treat them as sections of a larger book instead. So, we will create another toc file that defines the main sections of the book and will link the three toc files above to create a larger table of contents (our book).

book.xml

```
<toc label="Online Help Sample"  topic="html/book.html">
    <topic label="Overview" href="html/overview.html"/>
```

```
        <topic label="Concepts">
            <link toc="toc_Concepts.xml"/>
        </topic>
        <topic label="Tasks">
            <link toc="toc_Tasks.xml"/>
        </topic>
        <topic label="Reference">
            <link toc="toc_Ref.xml"/>
        </topic>
    </toc>
```

The Eclipse Platform can display any number of books/tables of contents. Each table of contents contains a collection of topics. Sometimes a higher-level component or product team is responsible for weaving together the documentation and topics supplied by a number of its component teams. For our purposes we'll assume that our plug-in should supply both the topics and the book that integrates the topics. Towards the end of the article we will look at how to make your documentation plug-in live happily in both a component world and a product world.
The following figure shows the book called "Online Help Sample" in the list of all available books.



Finishing our plug-in

The one remaining piece of work is to update our plugin.xml to actually contribute the toc files that we created. Start with updating the plugin.xml to contribute our book. The important thing here is to define this table of contents as a primary <toc>.

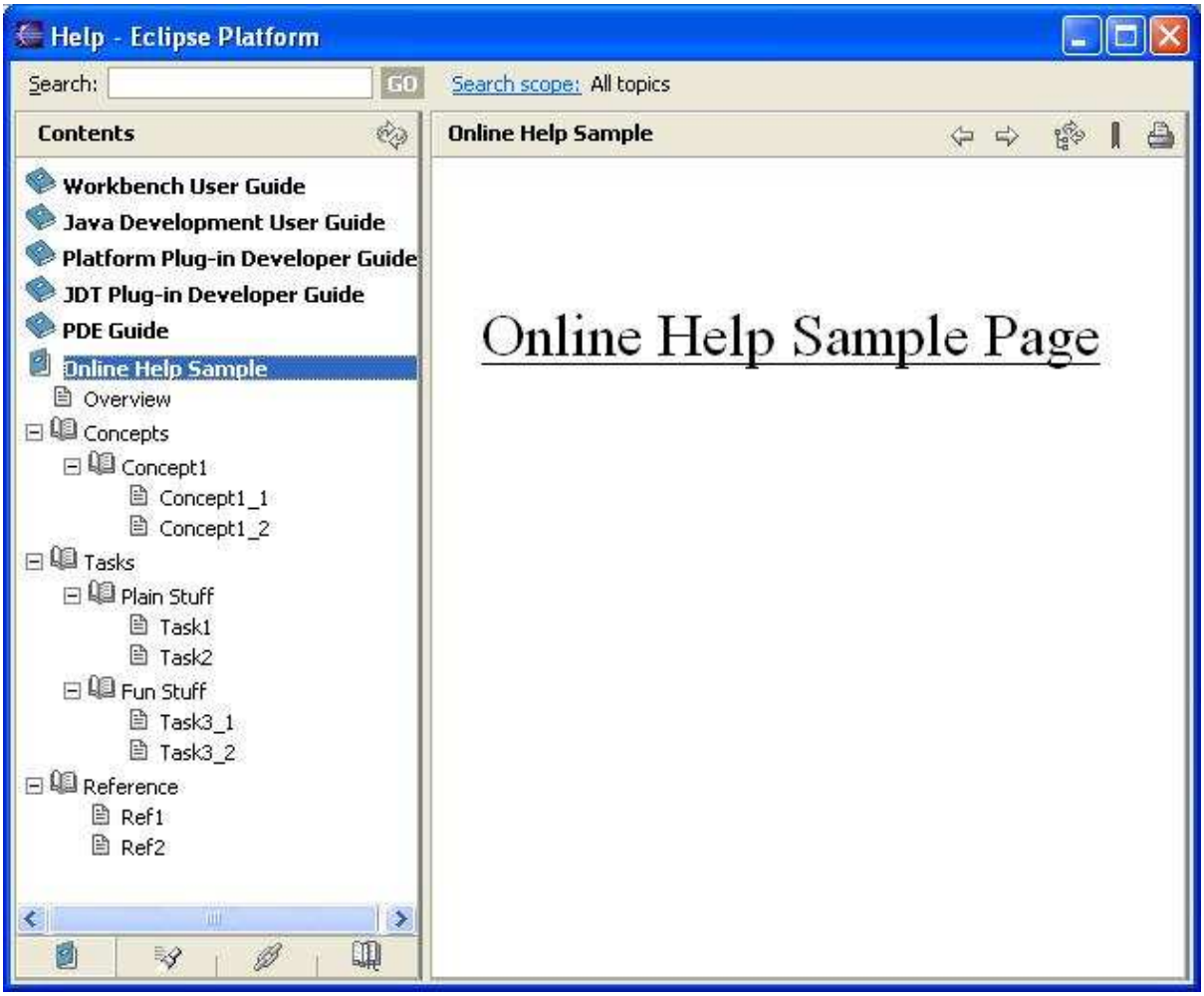
```
    <extension point="org.eclipse.help.toc">
        <toc file="book.xml" primary="true" />
    </extension>
```

Next we contribute the section toc files, and the important thing here is that primary is not set, so its default value (false) gets picked up:

```
    <extension point="org.eclipse.help.toc">
```

```
        <toc file="toc_Concepts.xml" />
        <toc file="toc_Tasks.xml" />
        <toc file="toc_Ref.xml" />
    </extension>
```

That's it, you're done. Now take your plug-in (click here for a zip file of the final plug-in) and drop it into the Platform's *plugins* directory, start Eclipse and choose Help -> Help Contents. Once you select the Online Help Sample and expand the topics you will see the following.



Before we continue a brief recap is in order:

- We started by creating our plug-in and document files.
- Next we created toc files to describe the structure/navigation of our content.
- We then created the toc file for the book and linked the other toc files as sections of the book.
- Finally we contributed all the files in the plugin.xml in the org.eclipse.help.toc extension point.

Integration of Tables of Contents

In our example we defined four tables of contents, of which one was viewed as the book, the other three as sections of the book. Since the book is aware of what sections to link we call this a top-down integration. It is possible to do it the other way around, i.e. bottom-up integration, where the sections specify which book (or section) to link to. A table of contents indicates its willingness to allow others to link to by providing anchors (<anchor id="..."/>) at the desired linking points.
Most often a plug-in defines a primary table of contents to which other plug-ins integrate their own table of contents. Tables of contents that are not primary or are not (directly or indirectly) integrated into other primary tables of contents are discarded from the final help navigation.
Linking is specified by using the fully qualified reference to a table of contents, such as

- top-down: <link toc="../the_other_plugin_id/path/to/toc.xml" /> or

- bottom-up: <toc link_to="../the_other_plugin_id/path/to/toc.xml#anchor_id"/>

Since the participating tables of contents are sometimes located in other plug-ins, and these plug-ins may not be installed, it is possible that some tables of contents remain unintegrated.
What if we expect our plug-in will sometimes be installed by itself, and in other cases it will be installed as part of a larger component or product. When deploying a free floating plug-in we want to ensure that our book is visible. When topics from a plug-in are integrated into a larger web it probably doesn't make sense for their standalone book to show up anymore. To support this nonintegrated or loosely integrated documentation, a plug-in can define a table of contents as a book by setting its **primary** attribute to true (inside plugin.xml) and having a **link_to** attribute pointing to the desired anchor in the larger web. This has the effect of the table of contents appearing as a book when the plug-in defining the anchor is not installed, and appearing as a section of the target book when the plug-in defining the anchor is installed.

Localization

Plugin manifest files externalize their strings by replacing the string with a key (e.g. %pluginName) and creating an entry in the plugin.properties file of the form:

```
pluginName = “Online Help Sample Plugin”
```

Strings from the contribution XML files and the topic HTML files are not externalized. The toc XML files and the HTML documentation must be translated and packaged in the appropriate NL directory structure or in an NL fragment, making sure files do not change their names. The NL directory structure inside a plug-in is

```
myplugin/
    nl/
        langCode/
            countryCode/
```

Here is how the plug-in with Japanese documentation is packaged (note: the files are not translated, just packaged in the appropriate location).

Server and zip files

The Platform utilizes its own help server to provide the actual web pages from within the document web. A help server allows the Platform to handle the wide variety of web browsers in a browser independent way while also providing plug-in aware support. The Platform's help server allows the documentation to also be packaged in zip files thus avoiding problems that may result when a large number of files are present. In our example, we put the HTML files in subdirectories of the plug-in directory. Alternatively, we could have placed them in a .zip file, called doc.zip, in the plug-in directory, maintaining the directory structure underneath. In general, the recommended way is to actually zip all the documentation. Here is how the plug-in is packaged.

Conclusions

We have seen how we can use the tables of contents to declare books the user can see. We then used the linking mechanism to integrate our topics. The Platform's mechanisms can be used to integrate new documentation, or to quickly wire in existing HTML based documentation without rewriting it. The mechanisms allow you to create multiple different views/books onto your documentation web. In our plug-in we only created a single book but additional books could easily be created. Lastly, we observed that the Platform provides the building blocks for documentation integration, provides minimal packaging guidelines, and lets the documentation authors have full control over structuring their HTML files.