6

Enhancing the HelpSet

So far, you have learned how to create a basic HelpSet. With the information from the previous chapters, you can create a fully functional help system with help topics, a TOC, an index, and a word-search index. You might, however, want to enhance your HelpSet by adding advanced controls within your help topics or by customizing the navigation facility.

This chapter covers the following topics to show you how to enhance your HelpSet:

- Creating pop-up and secondary windows
- Customizing the navigation facility
- Merging HelpSets

Creating Pop-up and Secondary Windows

You can add pop-up and secondary windows to your HelpSet for many different reasons. You might use a pop-up window to define a word within a help topic or to provide expanded information for a procedure. A secondary window is similar to a pop-up window except you use it to provide longer and more detailed information, to supplement the information in the original help topic. The user launches a pop-up or secondary window by clicking the text, image, or button that has encoded in it a link to that pop-up or secondary window.

A pop-up or secondary window displays a help topic, stored in an HTML-format file. That is, the topic has essentially the same format as the "main" help topics you've seen in the preceding chapters. You can add text, images, and links to the pop-up or secondary window text. However, keep in mind the customary usage of

pop-up windows (for example, defining a word in the help topic), and limit them to displaying text and images.

Differences Between Pop-up and Secondary Windows

Pop-up and secondary windows are similar, but there are some differences. Consider the following points when deciding which kind of window to use in a particular situation:

- Pop-up windows always appear directly adjacent to the object the user clicks to display them. But you can specify where secondary windows appear on the screen.
- Users can't adjust or move pop-up windows. But they can minimize, maximize, resize, and move secondary windows.
- Pop-up windows don't have titlebars: they are simply windows containing text. Secondary windows have (empty) titlebars.
- Pop-up windows close automatically when the user clicks another area in the HelpSet viewer. Users close secondary windows manually, by clicking the title bar's close-window button.

Even though pop-up and secondary windows are functionally different, you program them nearly the same way.

Programming Pop-up and Secondary Windows

To specify a pop-up or secondary window at some location within a help topic, you define a link using the HTML element `<object>`. The attribute `classid` specifies that the link launches a pop-up or secondary window:

```
<object classid="java:com.sun.java.help.impl.JHSecondaryViewer">
```

Nested within this element are `<param>` tags that provide the details: the link text, the kind of window to be launched, the topic to be displayed in the window, and so on.

For example, the following code from the Aviation topic file *markings.htm* defines a link to a secondary window:

```
<p>Markings provide special information for pilots when they are taxiing,
taking off, and landing. There are two types of markings:

<ul>
  <li>
  <object classid="java:com.sun.java.help.impl.JHSecondaryViewer">
    <param name="viewerActivator" value="javax.help.LinkLabel">
    <param name="viewerStyle" value="javax.help.SecondaryWindow">
```

```
    <param name="viewerLocation" value="400,200">
    <param name="viewerSize" value="600,300">
    <param name="text" value="Runway markings">
    <param name="textFontSize" value="medium">
    <param name="textFontWeight" value="plain">
    <param name="id" value="sec.runwaymarkings">
  </object>
  </li>
```

The link is defined by all the code between the <object> and </object> tags. The <p>, <ul>, and <li> tags are part of the *markings.htm* help topic containing the link: a paragraph introduces a bulleted list, which contains the text that links to the secondary window. The link text, "Runway Markings", is specified as the text parameter within the <object> element.

The parameters of the link can appear in any order within the <object> element. Each <param> tag uses name and value attributes to specify one parameter setting. For example, it might set the parameter named textFontSize to the value medium.

Several of the link object's parameters specify Java classes. (So does the object's classid attribute.) The following sections discuss these parameters.

Specifying the form of the link

The viewerActivator parameter indicates how the link appears to the user: as a button, as a text string, or as an image.

Linking with a button. A button is a quick and obvious way to set up a link to a pop-up or secondary window. To specify a button link, set the viewerActivator parameter to the value javax.help.LinkButton:

```
<param name="viewerActivator" value="javax.help.LinkButton">
```

By default, the button displays a right angle-bracket (>). You can specify the character string to appear on the button with the text parameter. For example:

```
<param name="text" value="JavaHelp Note">
```

Avoid using a long button string, which results in a large, awkward-looking button. You can format the button string, using a variety of text-formatting parameters, as described in the next section.

Linking with a text string. If you are inserting the link in a sentence, or if you simply don't want to use a button for the link, you can use inline text to represent the link to the pop-up or secondary window. For example:

```
<param name="viewerActivator" value="javax.help.LinkLabel">
```

```
<param name="text" value="Taxiway markings">
```

You can format text using font properties (family, weight, style, and size) and color. The following example shows the text-formatting parameter names in bold:

```
<param name="viewerActivator" value="javax.help.LinkLabel">
<param name="text" value="Taxiway markings">
<param name="textFontFamily" value="serif">
<param name="textFontWeight" value="bold">
<param name="textFontStyle" value="italic">
<param name="textFontSize" value="medium">
<param name="textColor" value="red">
```

Table 6-1 lists the recognized values for these text-formatting parameters. If you are accustomed to formatting text strings in HTML documents, this table should look familiar. The default value for each parameter is indicated in bold.

NOTE This table applies both to links that are text strings (`javax.help.LinkLabel`) and to links that are button strings (`javax.help.LinkButton`).

Table 6-1. Text-Formatting Parameters

Parameter Name	Recognized Values	Comments
textFontFamily	Serif **SansSerif** Monospaced Dialog DialogInput Symbol	
textFontWeight	**plain** bold	
textFontStyle	**plain** italic	
textFontSize	xx-small (equiv: 0) x-small (1) **small (2)** medium (3) large (4) x-large (5) xx-large (6)	Absolute font-size levels, defined by the HelpSet Viewer
	*n*pt (*n* is an integer)	Absolute font size, expressed in points (12 pts = 1 inch)
	+*n* (*n* is an integer) bigger means +1	Increases the font size by *n* font-size levels.

Table 6-1. Text-Formatting Parameters (continued)

Parameter Name	Recognized Values	Comments
	-n (*n* is an integer) smaller means –1	Decreases the font size by *n* font-size levels.
textColor	**black** **blue** cyan darkGray gray green lightGray magenta orange pink red white yellow	black is the default for a button string; blue is the default for a text string

Typically, you want your text link to stand out. Therefore, use a color such as blue to represent a link.

Linking with an image. Sometimes a button or inline text might be too plain for your needs. You can use an image as the link to the pop-up or secondary window. You can place the image inline with the text, or you can place it on a button.

An inline image link is coded similarly to a text link; just use an iconByID parameter instead of a text parameter:

```
<param name="viewerActivator" value="javax.help.LinkLabel">
<param name="iconByID" value="img.structurepic">
```

The value of the iconByID parameter must be a valid map ID of an image file, defined in the map file.

To place the image on a button, change the viewerActivator parameter value to use the LinkButton Java class instead of the LinkLabel class. (There is no LinkImage class in JavaHelp.)

Whether it's inline or on a button, an image can be specified by URL, instead of by map ID:

```
<param name="viewerActivator" value="javax.help.LinkLabel">
<param name="iconByName" value="../../images/forces.jpg">
```

You set the iconByName parameter, instead of the iconByID parameter. The URL you specify as the value must be relative to the topic file in which the link occurs. I recommend using map IDs, since you use them for other files in your project, and consistency helps reduce errors. Also, if you later move or rename the image files, you have to make the update only once—in the map file.

Defining the window's properties

When the user clicks on the link (text, button, or image), JavaHelp launches a pop-up window or a secondary window, depending on the value of the `viewerStyle` parameter: `javax.help.pop-up` or `javax.help.SecondaryWindow`. For example:

```
<object CLASSID="java:com.sun.java.help.impl.JHSecondaryViewer">
  <param name="viewerActivator" value="javax.help.LinkLabel">
  <param name="viewerStyle" value="javax.help.Popup">
```

Pop-up and secondary windows permit different types of customization. You can define only the size of a pop-up window, but you can define the size, location, and name of a secondary window.

To specify the size of a pop-up or secondary window, you set the `viewerSize` parameter:

```
<param name="viewerSize" value="300,125">
```

This example specifies a window that is 300 pixels wide and 125 pixels high.

Setting the appropriate window size is mostly a trial-and-error process. After you set the window's size, run the help system and try the link to see if the topic fits in the pop-up or secondary window. If the window is too small, a scroll bar appears to enable scrolling through the entire window. Whenever possible, I prefer to set the window size so that the pop-up or secondary topic fits without the need for scroll bars.

Since a pop-up window automatically appears directly adjacent to the object the user clicks to display it, you can't define its location. You can, however, set the secondary window's location, using the `viewerLocation` parameter:

```
<param name="viewerLocation" value="400,200">
```

This example places the secondary window's upper-left corner 400 pixels from the left edge of the screen and 200 pixels down from the top edge.

A pop-up window automatically closes when the user clicks outside it. A secondary window stays open, though, until the user manually closes it.

By default, the JavaHelp system ensures that only one secondary window is open at a time. If the user clicks a link that launches a secondary window, any currently open secondary window closes automatically.

To implement the multiple-windows strategy, use *viewer names*. The link that defines a secondary window can have a `viewerName` parameter:

```
<param name="viewerName" value="MoreInfo">
```

More precisely, the JavaHelp system ensures that only one secondary window *with a given viewer name* is open at a time. But the user might have a `MoreInfo` secondary window open, along with a `GlossaryTerm` secondary window and a `TipOfTheDay` secondary window.

NOTE I don't recommend using the viewer names feature. If the user can open any number of secondary windows, navigation can easily become a nightmare.

Viewer names would be more useful if you could define properties only once, for all secondary windows with the same name. But unfortunately, all `<object>` links are independent, even if they have the same `viewerName` settings.

You can experiment with viewer names by modifying the secondary window links in the Aviation topic files. Use different names for the `value` attribute to see if the secondary windows remain open. Make backup copies of the original Aviation topic files before you start, so that you can restore the originals when you're done experimenting.

Keep in mind that certain properties that might seem like window properties are actually set in the HTML file used for the pop-up or secondary window. For example, the background color of a secondary window is in fact the background color of the topic. You would therefore define the color directly in that topic's HTML file.

Defining the window's content

When JavaHelp opens the pop-up or secondary window, it displays the topic you specify with the `id` parameter:

```
<param name="id" value="pop.jhhelpid">
```

The value of the `id` parameter must be a valid map ID of a topic file, defined in the map file.

Alternatively, you can specify the URL of the topic file, using a `content` parameter instead of an `id` parameter:

```
<param name="content" value="../../pop-ups/fourforces.htm">
```

The URL you specify as the value must be relative to the current topic file. As with image links, I recommend using map IDs, not URLs.

Customizing the Navigation Facility

The navigation facility is one of the most important features of any help system. Without it, users can't find the information they need. Because the navigation facility is so important, you might want to customize it to fulfill your users' needs. You can customize the navigation facility in many different ways.

Changing Navigation Components' Tool Tips

One feature that is easy to customize is the navigation components' tool tips. The user sees these tool tips when the cursor rests over any of the navigation tabs in the HelpSet Viewer's navigation pane. For example, when the cursor rests over the Aviation HelpSet's TOC tab, a tool tip pops up with the name **Aviation TOC**. The tool tips work the same way for the index and word-search index tabs, as well.

You can change the tool tips by modifying the contents of the `<label>` elements for each navigation component specified in the HelpSet file. Give it a try:

1. Open the Aviation HelpSet file (*HelpSet.hs*) in a text editor.
2. Modify the TOC tool tip by changing the following line in the TOC section:

   ```
   <label>Aviation TOC</label>
   ```

   to:

   ```
   <label>Table of Contents</label>
   ```
3. Modify the index tool tip by changing the following line in the index section:

   ```
   <label>Aviation Index</label>
   ```

   to:

   ```
   <label>Alphabetical Index</label>
   ```
4. Modify the word-search index tool tip by changing the following line in the word-search index section:

   ```
   <label>Aviation Word Search</label>
   ```

   to:

   ```
   <label>Word Search Index</label>
   ```
5. Save the HelpSet file and then load the Aviation HelpSet into the HelpSet Viewer.

   When the help system opens, pause the cursor over each navigation tab in the navigation pane. Note the changes in the tool tips.

Excluding Navigation Components

You can also customize the navigation components to which users have access. If for some reason you don't want to include the TOC, index, or word-search index, you can simply remove it from the HelpSet file. You may not have known it at the time, but you effectively performed this action in Chapter 2, *Creating Your First HelpSet*, when you created a simple HelpSet. You omitted the `<view>` element specifying a word-search index from the MyJavaHelp HelpSet file. The result was a HelpSet containing only the TOC and index in the navigation pane.

When you exclude a navigation component you must remove its entire `<view>` element: everything from the `<view>` start-tag to the `</view>` end-tag. Give it a try, by removing the word-search index from the Aviation HelpSet:

1. Open the Aviation HelpSet file (*HelpSet.hs*) in a text editor.
2. Delete the following lines:

```
<view>
  <name>Search</name>
  <label>Aviation Word Search</label>
  <type>javax.help.SearchView</type>
  <data engine="com.sun.java.help.search.DefaultSearchEngine">
    JavaHelpSearch
  </data>
</view>
```

3. Save the HelpSet file, and then load the Aviation HelpSet into the HelpSet Viewer.

   When the help system opens, note that there is no index tab in the navigation pane.

Rearranging Navigation Component Tabs

In each HelpSet you've seen so far in this book, the navigation pane has opened with the TOC as the active navigation component. The order of the navigation tabs, from left to right, has been the TOC tab, the index tab, and the word-search index tab.

You are not, however, limited to this navigation structure. You can change the structure to accommodate any needs you or your users might have. The most common alternative is to make the navigation pane open with the index active, instead of the TOC. This is common because users of online help systems typically know what information they are trying to find. Instead of wading through a TOC, users frequently turn to the index to find a short word or phrase that summarizes the topic on which they want information. For this reason, it might make sense for you to make the navigation pane open with the index active.

In JavaHelp, you don't specify the navigation component you want to display when a particular HelpSet loads. Instead, if you want to make the index the default navigation component, just make it the first navigation component in the HelpSet file. The order of <view> elements in the HelpSet file defines the order of the navigation tabs in the HelpSet Viewer.

You can experiment with this feature by modifying the Aviation HelpSet file. Using cut-and-paste, move the entire <view> element that defines the index above the <view> element that defines the TOC:

```
<helpset version="1.0">
  <title>Aviation Information</title>
  <maps>
    <mapref location="Map.jhm"/>
    <homeID>intro</homeID>
  </maps>
  <view>
    <name>Index</name>
    <label>Aviation Index</label>
    <type>javax.help.IndexView</type>
    <data>Index.xml</data>
  </view>
  <view>
    <name>TOC</name>
    <label>Aviation TOC</label>
    <type>javax.help.TOCView</type>
    <data>TOC.xml</data>
  </view>
  ...
```

View the Aviation HelpSet. Note that the index appears first, not the TOC, and note that the order of the navigation tabs corresponds to the order of the <view> elements.

You can also create additional navigation tabs. What if you wanted to present two different TOCs—one for novice users and the other for expert users? You simply create a second TOC file and add a <view> element defining the other TOC to the HelpSet file. When you run the JavaHelp system, the navigation pane contains four navigation tabs instead of the usual three.

Try this with the Aviation HelpSet. There's a file named *AnotherTOC.xml* in the main project directory. It contains only information about airplane structure. Modify the HelpSet file, adding the <view> element printed in bold:

```
  ...
  <view>
    <name>TOC</name>
    <label>Aviation TOC</label>
    <type>javax.help.TOCView</type>
```

```
    <data>TOC.xml</data>
  </view>
  <view>
    <name>TOC</name>
    <label>Airplane Structure TOC</label>
    <type>javax.help.TOCView</type>
    <data>AnotherTOC.xml</data>
  </view>
  <view>
    <name>Index</name>
    <label>Aviation Index</label>
    <type>javax.help.IndexView</type>
    <data>Index.xml</data>
  </view>
  ...
```

Note that the secondary TOC's tool tip, "Airplane Structure TOC" differs from the tool tip of the primary TOC. This helps the user understand the difference between the two TOCs. Run the Aviation JavaHelp sample, and notice the changes.

Customizing the TOC

So far you have seen the icons shown in Figure 6-1 in all the TOCs with which you have worked.

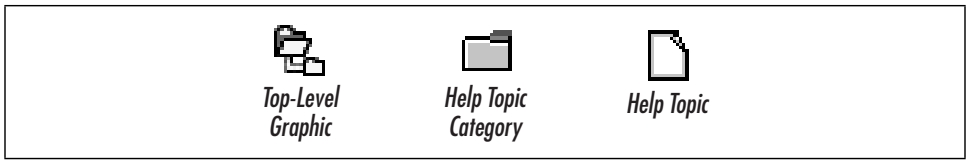

Figure 6-1. TOC icons

I have already explained the top-level image icon in previous chapters. It is an image you specify in both the map and TOC files, and it appears at the top of the TOC. The other two icons, the help topic category and the help topic icons, are default icons set by the HelpSet Viewer.

You can change any or all of these icons if you want. In fact, there are a couple of situations in which you might want to change them. One situation would be if you wish to adhere to company standards that might require the same "look and feel" as older help systems. Many of these older help systems use book and page icons. Another situation might be if you want to use custom icons for different types of help topics. For example, you might want to use an image of a regular page for conceptual topics, an image of a number for step-by-step procedures, and an image of tools for troubleshooting topics.

To use a custom icon, place the image file in the project's *Images* directory and specify a map ID and URL in the map file. For example, the beginning of your map file might look like this:

```
<mapID target="img.toplevelfolder" url="Images/toplevel.gif"/>
<mapID target="img.bookicon" url="Images/category.gif"/>
<mapID target="img.pageicon" url="Images/topic.gif"/>
```

You can then add `image` attributes to `<tocitem>` elements in the TOC file, specifying the appropriate icon map IDs. For example:

```
<tocitem image="img.bookicon" target="structure" text="Airplane Structure">
  <tocitem image="img.pageicon" target="empennage" text="Empennage"/>
  <tocitem image="img.pageicon" target="fuselage" text="Fuselage"/>
</tocitem>
```

Try customizing the icons in the Aviation JavaHelp sample. Before you start, download the TOC icon files from the "Examples" section of this book's web page. Once you have the icons on your computer, use the following steps to customize the TOC:

1. Copy the three image files you downloaded from the book's web site (*toplevel.gif*, *category.gif*, and *topic.gif*) to the *Images* subdirectory of the *Aviation* project directory). The *toplevel.gif* file from the web site should replace the original *toplevel.gif* file.
2. Add the following lines to the map file:

   ```
   <mapID target="img.bookicon" url="Images/category.gif"/>
   <mapID target="img.pageicon" url="Images/topic.gif"/>
   ```

3. In the TOC file, add the `image` attribute along with the appropriate image map ID to each category and topic line. When you are finished, the TOC file should look like the following example:

   ```
   <?xml version='1.0' encoding='ISO-8859-1' ?>
   <!DOCTYPE toc
     PUBLIC "-//Sun Microsystems Inc.//DTD JavaHelp TOC Version 1.0//EN"
            "http://java.sun.com/products/javahelp/toc_1_0.dtd">

   <toc version="1.0">
     <tocitem image="img.toplevelfolder" text="Aviation Information">
       <tocitem image="img.pageicon" target="intro" text="Introduction to Aviation"/>
       <tocitem image="img.bookicon" target="structure" text="Airplane Structure">
         <tocitem image="img.pageicon" target="empennage" text="Empennage"/>
         <tocitem image="img.pageicon" target="fuselage" text="Fuselage"/>
         <tocitem image="img.pageicon" target="gear" text="Landing Gear"/>
         <tocitem image="img.pageicon" target="powerplant" text="Powerplant"/>
         <tocitem image="img.pageicon" target="wing" text="Wing"/>
       </tocitem>
       <tocitem image="img.bookicon" target="aerodynamics" text="Aerodynamics">
   ```

```
      <tocitem image="img.pageicon" target="lift" text="Lift"/>
      <tocitem image="img.pageicon" target="weight" text="Weight"/>
      <tocitem image="img.pageicon" target="thrust" text="Thrust"/>
      <tocitem image="img.pageicon" target="drag" text="Drag"/>
    </tocitem>
    <tocitem image="img.bookicon" text="Flight Environment">
      <tocitem image="img.bookicon" target="airports" text="Airports">
        <tocitem image="img.pageicon" target="wind" text="Wind Direction
Indicators"/>
          <tocitem image="img.bookicon" target="runtaxi" text="Runways and
Taxiways">
            <tocitem image="img.pageicon" target="runways" text="Runways"/>
            <tocitem image="img.pageicon" target="taxiways" text="Taxiways"/>
            <tocitem image="img.pageicon" target="markings" text="Markings"/>
          </tocitem>
      </tocitem>
    </tocitem>
  </tocitem>
</toc>
```

4. Run the Aviation JavaHelp sample and notice the changes. Your TOC should resemble the TOC shown in Figure 6-2.

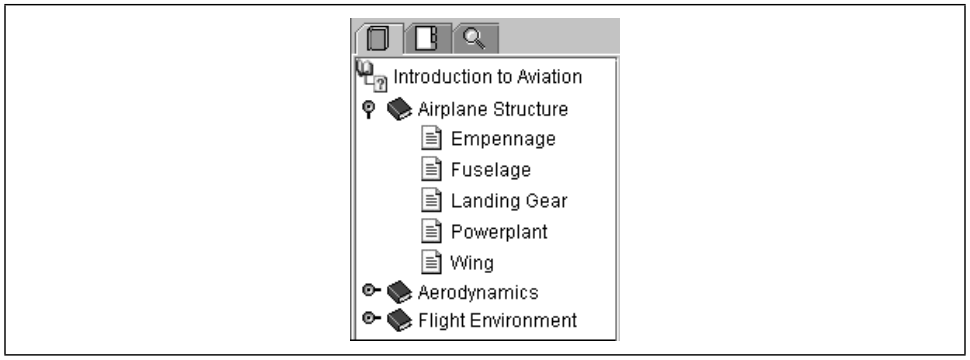

Figure 6-2. TOC with custom icons

Applying Advanced Word-Search Index Features

In Chapter 5, *Creating HelpSet Data and Navigation Files*, you learned how to create a basic word-search index. However, you are not limited to the default options you accepted when you created this search index; you can manipulate it to accommodate custom needs.

Customizing a word-search index generally means using a configuration file to modify different aspects of the search index. A configuration file is simply a text file with instructions on how the JavaHelp indexer should create the word-search index. To keep matters simple, I name my configuration file *Config.txt*.

You use the configuration file by using the `-c` option to the *jhindexer* command. For example, with the Aviation HelpSet, you enter this command:

```
jhindexer -c Config.txt Topics
```

The rest of this section explains the different uses for the configuration file, including changing the pathnames of topic files, identifying specific topic files to index, and modifying stopwords.

Changing pathnames of topic files

You will most likely never need to change the pathnames of topic files, especially if you encapsulate your HelpSet files into a JAR file. However, at the risk of confusing you, I discuss this topic only because you might come across it in Sun Microsystems' documentation and wonder what it is about. But keep in mind that if you follow the strategies presented in this book for directory and file management, you will never have to worry about this topic.

If the path to the topic files you create during development is different from that used later by the help system to find the topic files during searches, you must use the configuration file to account for this change. For example, let's say you store your *Topics* directory (the directory containing all your help topic files) in a separate directory on your computer, away from your other HelpSet files, or on another computer on a network. Then, you later reassemble the help system so that the *Topics* directory is in the same directory structure as the remaining HelpSet files. You then have to provide information to the JavaHelp indexer so it can record the location of the topic files at search time.

If you deviate from my guidelines in this book and should find the need to change the path names of topic files, you can do so either by removing a portion of the pathname or by adding a portion of it to the beginning of the existing path (known as *prepending*).

Use the `IndexRemove` command in the configuration file to remove the beginning portion of the path. For example, to change the path from *Development\Documentation\OnlineHelp\MyJavaHelp* to *OnlineHelp\MyJavaHelp*, use the following *IndexRemove* command in the configuration file:

```
IndexRemove Development\Documentation\
```

If you want to add to a pathname, use the `IndexPrepend` command in the configuration file. For example, to change a path from *\MyJavaHelp* to *\OnlineHelp\MyJavaHelp*, use the following command in the configuration file:

```
IndexPrepend OnlineHelp\
```

Identifying specific topic files to index

If you want to include only certain topic files in the word-search index instead of all the files in the topic directories, you must specify these files in the configuration file. To specify only the files you want to include in the word search, use the `File` command in the configuration file, as shown in this example:

```
File Topic1.htm
File Topic2.htm
File Topic3.htm
```

Be sure to place each filename on separate lines with its own `File` command.

Modifying stopwords

Stopwords are words excluded from the word-search index and not stored in the word-search database. The purpose of stopwords is to eliminate the redundancy of smaller, more common words such as “a” or “the” in the word-search index’s list of hits.

By default, the JavaHelp indexer excludes the following words from the word-search index:

a	did	his	now	than	way
all	do	how	of	that	we
am	does	if	off	the	what
an	etc	in	on	them	when
and	for	is	or	then	where
any	from	it	our	there	which
are	goes	let	own	these	who
as	got	me	see	this	why
at	had	more	set	those	will
be	has	much	shall	though	would
but	have	must	she	to	yes
by	he	my	should	too	yet
can	her	nor	so	us	you
could	him	not	some	was	

You can modify the use of stopwords in the following two ways:

- You can ignore the stopwords so that the indexer indexes every word in your help topics.
- You can specify your own custom stopwords to exclude from the word-search index.

To ignore stopwords, you don’t have to use a configuration file. Instead, use the *jhindexer* command’s `-nostopwords` option:

```
jhindexer –nostopwords Topics
```

To specify your own stopwords, you can use either of two methods:

- In the configuration file, include the `StopWords` command followed by a comma-separated list of stopwords:

  ```
  StopWords a, an, the, and, but, or, nor, for, so, yet
  ```

- List the stop words, one per line, in a separate text file, and specify the filename in the configuration file:

  ```
  StopWordsFile Stopwords.txt
  ```

  You don't have to name the stopword file *Stopwords.txt*, but this filename keeps your naming conventions simple. Save this text file in the main project directory, along with the configuration file, so that *jhindexer* can locate it.

Regardless of which method you use, when you specify your own stopwords, the JavaHelp indexer doesn't use the default stop list. The words you specify replace the default list. For this reason, you might want to use your stopwords in a text file that already contains Sun's default list. That way you have both the default list and your own custom list in one file.

NOTE To save time typing all the default stopwords into a text file, a stopword list is posted under "Examples" on this book's web page. This stopword list contains all of the default stopwords. You can use this file as a starter and add your own custom stopwords.

Combining multiple configuration commands

Each command in a configuration file must be on a separate line. For example:

```
IndexRemove Development\Documentation\
File Topic1.htm
File Topic2.htm
File Topic3.htm
StopWordsFile Stopwords.txt
```

Merging HelpSets

You can merge HelpSets, so that two or more independent HelpSets appear together in the HelpSet Viewer, almost as if they were a single HelpSet. On the HelpSet Viewer's TOC navigation tab, all the HelpSets' TOCs are concatenated. Data is combined similarly on the index and word-search index tabs.

A "product suite" provides an ideal opportunity for using this facility. For each component in the suite—such as a word processor, spreadsheet application, or

database application—you can develop a separate HelpSet. Then, you can merge all the HelpSets to provide a unified online help system for the entire suite.

Using the <subhelpset> Element

To merge multiple HelpSets, add one or more `<subhelpset>` elements to the HelpSet file whose data should appear first (the *master* HelpSet file). Add these elements just before the `</helpset>` end-tag, as shown in the following example:

```
<helpset>
  ...
  <subhelpset location="../OtherProject/HelpSet.hs"/>
  <subhelpset location="../ThirdProject/HelpSet.hs"/>
</helpset>
```

The `location` attribute specifies the location of another HelpSet file, relative to the location of the current HelpSet file. Data from the merged HelpSets appears in the navigational controls in the order of the `<subhelpset>` elements.

The TOC navigation components to be merged must all have the same view name:

```
<view>
  <name>TOC</name>      view name of this  component is "TOC"
  ...
</view>
```

Similarly, all index views should have the same view name, and all word-search index views should have the same view name.

Try merging the MyJavaHelp HelpSet you created in Chapter 2 with the Aviation HelpSet, as follows:

1. Note the location of the *MyJavaHelp* project directory. (In this example, assume that the *MyJavaHelp* directory is at the same level as the *Aviation* directory.)
2. Verify that the TOC and index navigation components have the same view name ("TOC" and "Index") in both HelpSet files.
3. In the Aviation HelpSet.hs file, add the following line right before the `</helpset>` end-tag:

   ```
   <subhelpset location="../MyJavaHelp/HelpSet.hs"/>
   ```

   You can also specify the location of a HelpSet with an absolute pathname, such as:

   ```
   <subhelpset location="/net/boron/apps/Blivet/HelpSet.hs"/>
   or
   <subhelpset location="R:\apps\Blivet\HelpSet.hs"/>
   ```

4. Save the HelpSet file, and then view the Aviation HelpSet. Note that the navigation components are appended.
5. When you finish looking at the changes, revert to the original HelpSets from the backups you made previously.

Considering the Drawbacks of Merging

There are a few drawbacks with merging HelpSets. Several of them result from the fact that the HelpSet Viewer simply appends navigation component data; it doesn't combine the data in a more sophisticated way.

Your TOCs appear exactly as they would if the HelpSets were not merged. If you include a top-level folder image for each TOC, the icon appears multiple times in the merged TOC—once for each TOC that originally contained the image (see Figure 6-3).

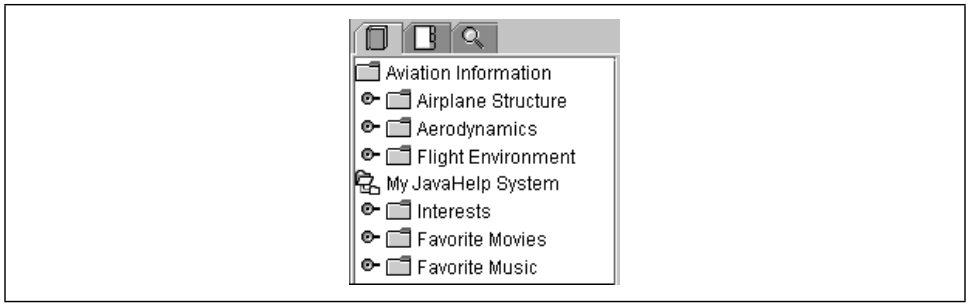

Figure 6-3. Awkwardly merged TOC

The indexes are simply appended, so even if the individual indexes are in alphabetical order, the merged index isn't. This drawback doesn't apply to the word-search index. When the user enters a word or phrase, the word-search index searches all HelpSets' databases and displays all the topic titles together.

The tool tip that appears when you rest the cursor over a navigation tab displays the text that was defined in the master HelpSet file. When you merged the MyJavaHelp and Aviation HelpSets, you probably noticed that the tool tips for the tabs were **Aviation TOC**, **Aviation Index**, and **Aviation Word Search**.

In addition to limitations with the navigation controls, you are also limited by the `location` attribute in the `<subhelpset>` tag. To make the location of the secondary HelpSet file relative to the master HelpSet file, you must place the entire secondary HelpSet in a disk location adjacent to the master HelpSet file. This limitation is why you moved the *MyJavaHelp* directory in the previous example to merge it with the Aviation HelpSet. If the secondary HelpSet is located in a separate

directory structure, you have to use an absolute location with the protocol type, as shown in the following example:

```
<subhelpset location="file:/c:/MyJavaHelp/HelpSet.hs"/>
```

The problem with using an absolute location is that if the HelpSet resides on an end user's computer, you won't necessarily know the absolute location to specify in the master HelpSet file. You then need an installation utility to determine this information and change the master HelpSet file during installation.

Overcoming the drawbacks

There are certain actions you can take to overcome most of these limitations. If you are planning a HelpSet that might later be merged with another, you should keep in mind some development alternatives.

If you don't like having the top-level image recur throughout the TOC, avoid using the top-level image and heading in all TOC files. That way the image doesn't reappear throughout the TOC when the HelpSets are merged. For example, with the TOC in Figure 6-3, you eliminate the lines in the TOC files responsible for generating the top-level images and the titles "Aviation Information" and "My JavaHelp System."

The index presents problems that are difficult to resolve because there is no way to alphabetize the index items after they are merged. The best way to work around this limitation is to add a top-level index item in all merged indexes with a phrase that identifies the particular HelpSet. This could mean that your index ends up with three levels of index items. For example, the index file for MyJavaHelp would resemble the following example:

```
<index version="1.0">
<indexitem text="My JavaHelp System">
  <indexitem target="computers" text="computer interests"/>
  <indexitem text="favorites">
    <indexitem target="movies" text="movies"/>
    <indexitem target="music" text="music"/>
  </indexitem>
  <indexitem target="fitness" text="fitness interests"/>
  <indexitem text="interests">
    <indexitem target="computers" text="computers"/>
    <indexitem target="fitness" text="fitness"/>
  </indexitem>
  <indexitem target="movies" text="movies, favorite"/>
  <indexitem target="music" text="music, favorite"/>
  <indexitem target="overview" text="overview"/>
</indexitem>
</index>
```

Notice that the first-level index item ("My JavaHelp System") doesn't have a target because you don't want it to launch an actual help topic. Also, notice that its tags contain the rest of the index nested within it. The biggest setback with using this approach is that the index will seem awkward if it is ever used independently (as opposed to being merged with other indexes).

Managing Projects with Merged HelpSets

Planning for a merged HelpSet begins before any HelpSet is even created. You should look at the application for which you are creating online help and decide if it offers the potential for add-on or supplemental applications. For example, if you are creating online help for a word-processing application, it's possible you might want to later merge it with other office suite products. Also, if you are creating online help for different JavaBean™ components, you should consider how HelpSets for the individual beans might eventually be merged.

The best way to keep merged HelpSets organized is to use a dataless master HelpSet file. A dataless master HelpSet file keeps all the merged HelpSets organized. It doesn't contain map or specific navigation view data for individual HelpSets. Instead it provides a container for the `<subhelpset>` elements that specify the HelpSet files to be merged. The following example shows a dataless master HelpSet file that merges the MyJavaHelp and Aviation HelpSets:

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
<!DOCTYPE helpset
  PUBLIC "-//Sun Microsystems Inc.//DTD JavaHelp HelpSet Version 1.0//EN"
         "http://java.sun.com/products/javahelp/helpset_1_0.dtd">

<helpset version="1.0">
  <title>Merged JavaHelp System</title>
  <view>
    <name>TOC</name>
    <label>TOC</label>
    <type>javax.help.TOCView</type>
  </view>
  <view>
    <name>Index</name>
    <label>Index</label>
    <type>javax.help.IndexView</type>
  </view>
  <view>
    <name>Search</name>
    <label>Word Search</label>
    <type>javax.help.SearchView</type>
  </view>
  <subhelpset location="Aviation/HelpSet.hs"/>
  <subhelpset location="MyJavaHelp/HelpSet.hs"/>
```

```
</helpset>
```

In this example, I placed the master HelpSet file in a master project directory and placed the secondary HelpSet files within *Aviation* and *MyJavaHelp* directories, adjacent to the master *HelpSet.hs* file. The title and tool-tip labels in the master HelpSet file use generic wording so that they are appropriate for any merged HelpSet. Also, the navigation view names in all HelpSet files are the same; the tool tips appear the same whether the HelpSets are viewed alone or are merged.