5

Creating HelpSet Data and Navigation Files

HelpSet data and navigation files connect the HTML-format help topic files of your HelpSet. To show you how to create HelpSet data and navigation files, this chapter discusses the following topics:

- Understanding XML
- Creating the HelpSet file
- Assigning map IDs to help topics
- Specifying the navigation components

Understanding XML

The JavaHelp map, TOC, index, and HelpSet files are all written in the *Extensible Markup Language* (XML). Like HTML, XML is a web standard, but it's more flexible. You can use XML to format any kind of textual data, not just web pages. In addition, you can use XML to describe the *structure* of textual data, which is how XML is used in HelpSet data and navigation files.

The map, TOC, index, and HelpSet files for the Aviation sample project contain markup tags you don't find in HTML files. For example, look at the following line from the TOC file:

```
<tocitem target="intro" text="Introduction to Aviation"></tocitem>
```

The tags `<tocitem>` and `</tocitem>` are not standard HTML tags. A standard web browser simply skips over the entire line. But JavaHelp's HelpSet Viewer does know how to interpret these tags.

Each XML file contains a tree-structured hierarchy of *elements*. That is, an element can contain other elements. In particular, the *root* element of the tree contains all

the other elements in the file. Elements that contain other elements have this form:

```
<tag  name1="value1"  name2="value2">
    text and other elements go here
</tag>
```

This kind of element has a start-tag (for example, `<tocitem ...>`), and a corresponding end-tag (for example, `</tocitem>`).

Some elements, the *leaves* of the tree structure, don't contain any other element. Such elements can use this single-tag form:

```
<tag  name1="value1"  name2="value2"/>
```

Alternatively, you can use the separate start-tag/end-tag form for such elements:

```
<tag  name1="value1"  name2="value2"></tag>
```

An element can have any number of "name=value" pairs, which are called *attributes*. JavaHelp makes heavy use of attributes. The value of an attribute must be enclosed in quotes. In general, you can use either single quotes or double quotes; this book uses double quotes only.

The following excerpt from the Aviation TOC file contains both kinds of elements:

```
<tocitem text="Introduction to Aviation" target="intro">
    <tocitem text="Landing Gear" target="gear"/>
</tocitem>
```

The "Introduction to Aviation" element contains the "Landing Gear" element. Accordingly, the TOC displayed by the JavaHelp system contains "Landing Gear" as a subentry of "Introduction to Aviation."

NOTE When one or more elements are contained within another element, I use indentation to emphasize the relationship. But this is just a custom; indentation has no real meaning in an XML file. You can even place an element and its subelements on a single text line.

You might want to use comments in some of your XML files to point out information to other developers or help authors who might look at your files. An XML comment has this form:

```
<!--  text, possibly multiple lines  -->
```

The comment text itself must not include a double-hyphen (`--`).

For example, if you want to identify a section in the map file that contains map IDs for images, you might place the following line just before the actual image map IDs:

```
<!-- Map IDs for Images -->
```

Creating the HelpSet File

Take a look at the HelpSet file for the Aviation JavaHelp sample. I like to keep filenames simple, so I named it *HelpSet.hs*. Another good name would be *Aviation.hs*. Regardless of what you name the HelpSet file, you must use the `.hs` filename extension; the HelpSet Viewer won't open it otherwise.

Using a text editor, open the file *HelpSet.hs*. It contains the following code:

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

This code demonstrates both HelpSet data tags and navigation component tags. I explain how to use tags to merge HelpSets in Chapter 6, *Enhancing the HelpSet*.

Understanding the HelpSet Data Elements

The HelpSet file begins with two optional declarations. The XML declaration (`<?xml ...`) and the document type definition, or DTD, (`<!DOCTYPE ...`) identify this file as an XML document with a particular structure. This can be useful if an XML-aware program other than the HelpSet Viewer needs to access the data in this file.

The HelpSet file must contain an element hierarchy whose root element is named "helpset." Accordingly, there is a `<helpset>` start-tag at the top of the file and a `</helpset>` end-tag at the bottom. You can include an optional `version` attribute in the start-tag, to specify the version of this online help project.

Within the `<helpset>` element, you create:

HelpSet data elements
: A `<title>` element and a `<map>` element

Navigation elements
: One or more `<view>` elements

The data elements are described in the following paragraphs, and the navigation elements in the next section.

The `<title>` element specifies the name of the HelpSet, which appears in the HelpSet Viewer's titlebar.

```
<title>Aviation Information</title>
```

The `<maps>` element uses two subelements to specify essential data:

```
<maps>
  <mapref location="Map.jhm"/>
  <homeID>intro</homeID>
</maps>
```

- The `<mapref>` element uses a `location` attribute to specify the location of the map file, relative to the HelpSet file.
- The `<homeID>` specifies the map ID of the topic to be displayed when the JavaHelp system starts.

The JavaHelp system uses the data in the map file to convert map IDs into URLs. For example, the map ID `intro`, specified in the `<homeID>` element, maps to URL *Topics/introduction_to_aviation.htm*. This is the HTML file that contains the help topic "Introduction to Aviation." Thus, when the JavaHelp system first opens the Aviation HelpSet, it displays the "Introduction to Aviation" topic.

NOTE Since you code the map ID as the contents of the `<homeID>` element, not as an attribute value, you don't have to place the map ID (`intro`) inside quotes.

Understanding the Navigation Component Elements

Following the HelpSet data elements are the navigation component elements, coded as `<view>` elements. Each `<view>` element specifies one of the navigation components: TOC, index, and word-search index. The following subelements within the `<view>` element supply the details:

- `<name>` provides an internal identifier for the navigation component. It produces nothing visible in the HelpSet Viewer. As always, I use simple names:

  ```
  <name>TOC</name>
  ```

- `<label>` specifies the text that appears in the tool tip label when the cursor rests over one of the navigation component tabs in the navigation pane of the HelpSet Viewer. For example, the following line generates the text `Aviation TOC` in the TOC tool tip label:

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

  To see this label, run the Aviation JavaHelp sample. Point and hold the cursor over the TOC, index, and word-search index tabs.

- `<type>` provides internal information to the JavaHelp system, to specify the paths to the navigator Java class.
- `<data>` specifies the location, relative to the HelpSet file, of the file or directory that contains the data for the navigation component. For example, the data that defines the index for the Aviation HelpSet is in file *Index.xml*.

For the word-search index, the `engine` attribute specifies the search engine Java class.

Assigning Map IDs to Help Topics

The map file associates each help-topic map ID with the URL of a help topic HTML file. The map ID acts as a nickname for the file; the TOC file, index file, and word search-index file all use these nicknames to specify help topics.

In Chapter 1, *Understanding JavaHelp*, I showed you how a map file works within a HelpSet. In this section I expand on the map file by explaining how you create, edit, and structure it.

The Map File

Take a look at the map file, *Map.jhm,* for the Aviation JavaHelp sample. A portion of the contents are shown here:

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
<!DOCTYPE map
  PUBLIC "-//Sun Microsystems Inc.//DTD JavaHelp Map Version 1.0//EN"
         "http://java.sun.com/products/javahelp/map_1_0.dtd">

<map version="1.0">

  <mapID target="toplevelfolder" url="Images/toplevel.gif"/>

  <mapID target="intro" url="Topics/introduction_to_aviation.htm"/>

  <mapID target="aerodynamics" url="Topics/Aerodynamics/aerodynamics.htm"/>
  <mapID target="drag" url="Topics/Aerodynamics/drag.htm"/>
  <mapID target="lift" url="Topics/Aerodynamics/lift.htm"/>
  <mapID target="thrust" url="Topics/Aerodynamics/thrust.htm"/>
  <mapID target="weight" url="Topics/Aerodynamics/weight.htm"/>

</map>
```

As in the HelpSet file, the XML declaration (`<?xml ...`) and the document type definition (`<!DOCTYPE ...`) are optional. The root element of the map file is named "map," specified with the `<map>` and `</map>` tags. The `<map>` start-tag includes an optional `version` attribute, where you can specify the JavaHelp map-version number.

The `<map>` element contains a series of `<mapID>` elements, each of which defines one map ID. A map ID can be associated with a topic file or with an image file.

Following is a typical `<mapID>` element:

```
<mapID target="intro" url="Topics/introduction_to_aviation.htm"/>
```

`<mapID>` elements contain no text data, just attributes. The `target` attribute specifies the map ID. The `url` attribute specifies the URL of the topic file or image file, relative to the location of the map file. In specifying the `url` value, be sure to:

- Separate directory names with forward slashes (`/`).
- Begin the URL with a character other than forward slash, because the URL must be *relative* to the map file.
- Include the filename extension (for example, *.htm*, *.html*, *.jpg*).

Naming the Map File

When it introduced JavaHelp, Sun used the *.jhm* filename extension on map files. Some third-party help-authoring tools have adopted this convention. You can use any filename extension on a map file (for example, you might want to use *Map.xml* instead of *Map.jhm*). Just be sure that the name of your map file is specified accurately in the `<mapref>` element in the HelpSet file. For example:

```
<mapref location="Map.xml"/>
```

Structuring the Map File

When you create the map file, you should understand how to structure it to make it easier to work with. You can place the map IDs in any order in the map file, but you should use a strategy that keeps your JavaHelp project well organized. In previous chapters, I discussed separating HTML files into different directories, based on the subject matter of the help topics. I suggest using a similar approach with the map file. Group related topics together in the map file, and separate each group by spaces or by XML comments. It doesn't matter to JavaHelp how the map IDs are organized, but it will matter to you later if you try to locate certain map IDs but can't figure out where in the map file you placed them.

I usually put images and miscellaneous map IDs at the beginning of my map file, and I put pop-up and secondary window map IDs at the end. In the Aviation JavaHelp sample map file, the first `<mapID>` element specifies a map ID for the top-level folder image. Shown in Figure 5-1, the top-level folder image appears at the top of the TOC. TOC image files are the only image files for which you must set a map ID. (I discuss other TOC image files in Chapter 6.) The top-level image file in this example is referenced only by the TOC file. You can substitute any image for it, but I recommend using the top-level folder image Sun supplies with JavaHelp.

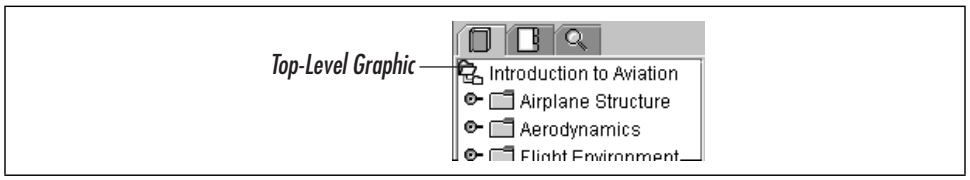

Figure 5-1. Top-level folder image

NOTE You may want to refer back to Chapter 2, *Creating Your First HelpSet*, where I discuss working with the top-level image file.

If you use images within help topics to launch pop-up windows and secondary windows, you might also want to define them in the map file. I discuss this topic more in Chapter 6.

After the map IDs for the image files, I list the map IDs for each help topic. Notice that I separate them into related subjects and separate each set with a space. Again, you don't have to place the map IDs in any particular order, but it helps if you organize them in a recognizable pattern (for example, the same way the TOC is organized).

Try to use map IDs that are meaningful. You could simply use numbers for each ID; JavaHelp doesn't care what you use. But if you use numbers, you can't identify the content of a topic by looking at its map ID. My map IDs usually contain some or all of the help topic's HTML filename. For example, I use the map ID `aerodynamics` for the HTML file *aerodynamics.htm*, and I use the map ID `gear` for the HTML file *landing_gear.htm*.

If you are creating a very large HelpSet, you might want to create a simulated map ID hierarchy, using multipart names. For example, I could have assigned the map ID `aircraft.gear` to the HTML file *landing_gear.htm* and the map ID `aircraft.wing` to the HTML file *wing.htm*. Then, whenever I notice a map ID starting with "aircraft," I would know that the topic has to do with aircraft structure. Similarly, to identify map IDs for images, I use `img.` with the map ID. For pop-up and secondary-window topic map IDs, I use `pop.` and `sec.` respectively.

Specifying the Navigation Components

The navigation components are an important part of an effective online help system. If users can't find the information for which they need help, they will lose faith in the help system. One disappointing fact is that users won't give your navigation system many chances to prove itself. Users generally search for a help topic only a few times before they give up on finding it.

This section provides some tips on creating an effective navigation facility. Its focus, however, is to show you how to specify the various JavaHelp navigation components: the TOC, index, and word-search index.

Creating the Table of Contents

A HelpSet's TOC should function the same way as a traditional book's TOC. Users will look in the TOC to find topics arranged by subject matter.

If you have been following my practices for planning a JavaHelp project, you are off to a good start planning a well-designed TOC. In discussing project planning and preparing topics in previous chapters, I encouraged you to group related sub-

jects within their own directories to keep them organized. You can now use this same structure to present the TOC for your HelpSet. Since you already organized the directories and topic files by subject, simply mimic the structure in the TOC. The only difference is that, in the TOC, you should not necessarily alphabetize the topics within a directory. Organize the topics in a logical order according the concepts the users must know and procedures they must perform.

When placing topic titles in the TOC, be sure to use the same title that you used in the topic file's `<title>` element. When users select a topic from the TOC, they expect the title to be the same in the topic file. If you use the title "Landing an Airplane" in the TOC, users will be confused if the actual topic title says "Landing Gear."

Building the JavaHelp TOC

Let's take a look at the TOC file, *TOC.xml*, for the Aviation JavaHelp sample. Here is a complete listing:

```
<?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" target="intro" text="Aviation Information">
    <tocitem target="structure" text="Airplane Structure">
      <tocitem target="empennage" text="Empennage"/>
      <tocitem target="fuselage" text="Fuselage"/>
      <tocitem target="gear" text="Landing Gear"/>
      <tocitem target="powerplant" text="Powerplant"/>
      <tocitem target="wing" text="Wing"/>
    </tocitem>
    <tocitem target="aerodynamics" text="Aerodynamics">
      <tocitem target="lift" text="Lift"/>
      <tocitem target="weight" text="Weight"/>
      <tocitem target="thrust" text="Thrust"/>
      <tocitem target="drag" text="Drag"/>
    </tocitem>
    <tocitem text="Flight Environment">
      <tocitem target="airports" text="Airports">
        <tocitem target="wind" text="Wind Direction Indicators"/>
          <tocitem target="runtaxi" text="Runways and Taxiways">
            <tocitem target="runways" text="Runways"/>
            <tocitem target="taxiways" text="Taxiways"/>
            <tocitem target="markings" text="Markings"/>
          </tocitem>
      </tocitem>
    </tocitem>
```

```
</tocitem>
</toc>
```

Figure 5-2 shows the TOC, as it appears in the HelpSet Viewer.

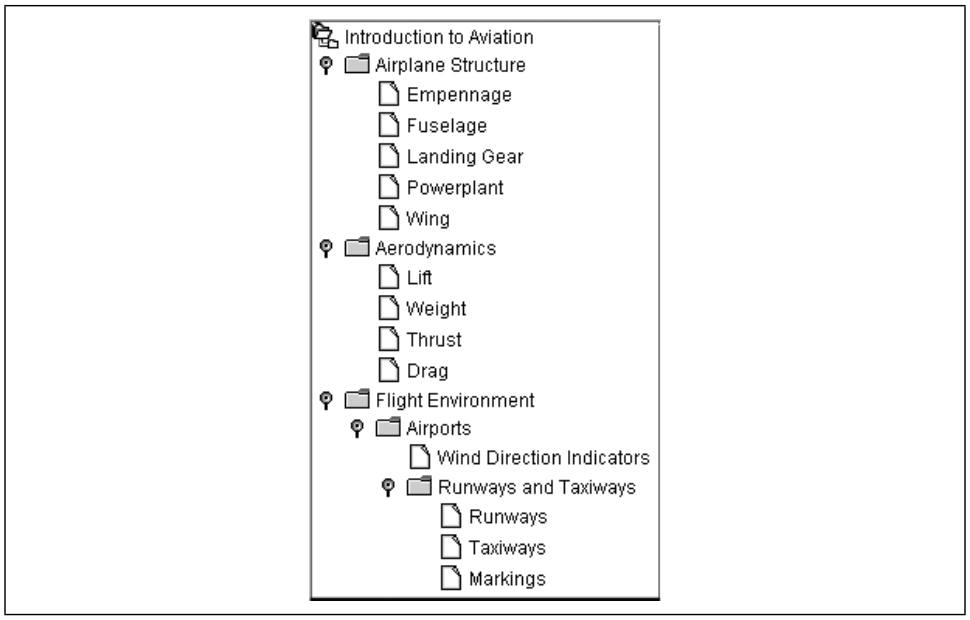

Figure 5-2. Aviation TOC

Creating the TOC elements

As in the HelpSet file, the XML declaration (`<?xml ...`) and the document type definition (`<!DOCTYPE ...`) are optional. The root element of the TOC file is named "toc," specified with the `<toc>` and `</toc>` tags. The `<toc>` start-tag includes an optional `version` attribute, where you can specify the JavaHelp map version number.

The `<toc>` element contains a set of `<tocitem>` elements. Each `<tocitem>` element uses attributes to specify one TOC entry. For example:

```
<tocitem target="intro" text="Introduction to Aviation"/>
```

In this example, when the user clicks the TOC entry "Introduction to Aviation," the topic with map ID "intro" appears. You can use these attributes in the `<tocitem>` tag:

- The `text` attribute specifies the character string to appear as the TOC entry.

- The `target` attribute specifies the map ID of the topic to be displayed when the user clicks the TOC entry. This attribute is optional; if you omit it, nothing happens when the user clicks on the TOC entry.
- The `image` attribute (not included in the above example) specifies the icon to be displayed along with the TOC entry. If you omit this attribute, the HelpSet Viewer displays an open/close control (for entries that have subentries) or a sheet-of-paper icon (for entries without subentries). Figure 5-2 shows these icons.

Creating help topic categories

When you have a group of related topics, create a separate category (TOC level) for them, to make them easier for users to find. (I also suggest that you place the topic files in a separate directory within your help project's *Topics* directory.)

You create multiple categories by nesting `<tocitem>` elements within each other. For example, the following excerpt from the Aviation TOC file defines the category entry "Aerodynamics," along with subentry topics titled "Lift," "Weight," "Thrust," and "Drag."

```
<tocitem target="aerodynamics" text="Aerodynamics">
  <tocitem target="lift" text="Lift"/>
  <tocitem target="weight" text="Weight"/>
  <tocitem target="thrust" text="Thrust"/>
  <tocitem target="drag" text="Drag"/>
</tocitem>
```

NOTE If a `<tocitem>` element defines a category (that is, if it has subentries), use the start-tag/end-tag form `<tocitem>...</tocitem>`. If a `<tocitem>` element defines a TOC entry with no subentries, you can use the single-tag form `<tocitem.../>`.

The user can open and close a category—make its subentries appear and disappear from the TOC display—by using the control that appears in front of the category entry. Figure 5-3 demonstrates this control.

Figure 5-4 shows a more elaborate TOC entry hierarchy with a more complex nesting of categories and topics.

The `<tocitem>` elements that define this structure are nested in the same way:

```
<tocitem text="Flight Environment">
  <tocitem target="airports" text="Airports">
    <tocitem target="wind" text="Wind Direction Indicators"/>
    <tocitem target="runtaxi" text="Runways and Taxiways">
      <tocitem target="runways" text="Runways"/>
      <tocitem target="taxiways" text="Taxiways"/>
```

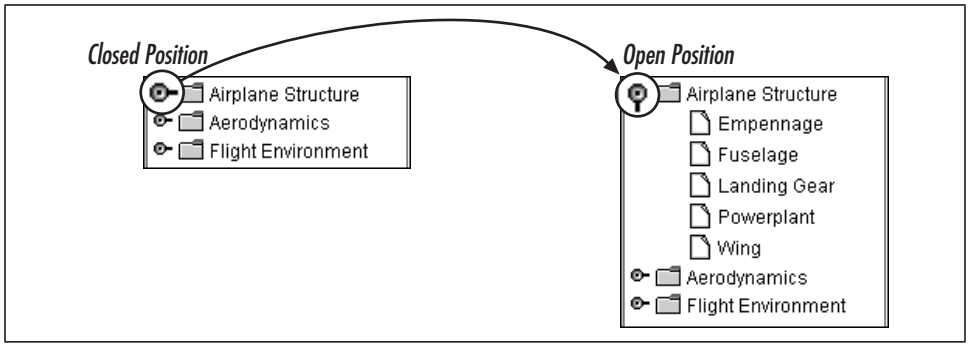

Figure 5-3. Opening and closing TOC categories

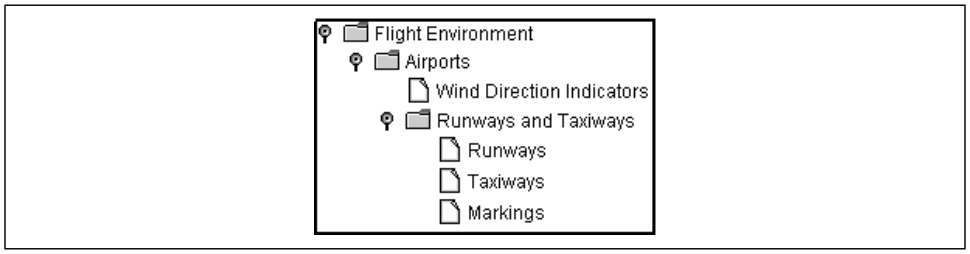

Figure 5-4. TOC categories, subcategories, and help topics

```
        <tocitem target="markings" text="Markings"/>
      </tocitem>
    </tocitem>
  </tocitem>
```

If all this nesting of XML elements seems confusing, just keep the nesting to a minimum.

Help topics for category entries

A TOC entry that defines a category can also have a topic associated with it. (Not all online help systems work this way.) For example, look again at the definition of the "Aerodynamics" category:

```
<tocitem target="aerodynamics" text="Aerodynamics">
```

When the user clicks on the TOC entry "Aerodynamics," the help topic with map ID "aerodynamics" appears. If you did not want this TOC entry to link to a help topic, omit the `target` attribute:

```
<tocitem text="Aerodynamics">
```

With this change, nothing happens when the user clicks on the TOC entry "Aerodynamics."

I recommend having a topic for every TOC entry, including categories. The most important thing, however, is to be consistent. If one category has a topic, all categories should have topics.

Creating a top-level TOC entry

You should create a top-level category to contain all other topics in the HelpSet. This entry provides an overall title for the HelpSet. Thus, the overall structure of the TOC file looks like this (using the Aviation example):

```
<toc>
<tocitem image="img.toplevelfolder" target="intro" text="Introduction to
Information">

    (all other <tocitem> elements, defining help topics
    and help topic categories)

</tocitem>
</toc>
```

Use the `image` attribute to specify an icon to be displayed for this top-level category. Sun supplies a standard icon for this purpose, in file *toplevel.gif*. It depicts a folder containing other folders: a top-level folder (see Figure 5-1). The map file for the Aviation HelpSet assigns map ID `img.toplevelfolder` to this icon file. Accordingly, the `image` attribute in the example above uses this map ID.

The `target` attribute specifies map ID `intro`. When the user clicks the top-level item in the TOC, the HelpSet Viewer displays the introductory help topic, which is assigned to that map ID. As always, the value of the `text` attribute specifies the title to appear in the TOC—in this example, `Aviation Information`.

There is no open/close control for the top-level category. The user can open or close the category by double-clicking it.

Creating the Index

Designing a good index differs from designing a good TOC, because users use an index in a much different way than they use a TOC. In constructing a TOC, you organize related topics into categories. With an index, however, you focus on each topic individually. The only time you consider how multiple topics fit together is when you have to organize related secondary index items under a primary index item.

To index a topic you must first identify the main ideas associated with that topic. You might even want to record this information in a simple text file. If you want a more sophisticated approach to indexing, you can create a spreadsheet or database to store your index information. Once you have determined a topic's main

ideas, you need to decide on words or short phrases that identify that topic's subject. These words and phrases should be either nouns or gerund phrases—phrases that act as nouns, usually by adding "ing" to the phrases' verbs. An example of a gerund phrase is "landing an airplane." The entire phrase acts as a noun. In this example, the gerund phrase represents the act of landing an airplane.

When you create the index, try to think of as many synonyms for your index items as possible. You have no way of knowing what word or phrase your readers are thinking of when trying to find a particular topic. One reader might look for the word "airport" while another reader might look for "landing field." Be prepared to create several index items for each help topic. Approaching the index from the viewpoints of many different users is challenging, but it improves the usability of the index.

Index items are arranged alphabetically and are not capitalized unless they are proper nouns (for example, the name of a window or control). Within this alphabetized list, certain index items might contain secondary index items. For example, a primary index item might be the word "airplane." However, this item may have several nested secondary items such as "single engine," "multiple engine," "mechanics," or "about." Don't use more than two levels of index items; it's confusing to your readers and could render the index useless.

Building the JavaHelp index

Let's take a look at the index file, *Index.xml*, for the Aviation JavaHelp sample:

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
<!DOCTYPE index
  PUBLIC "-//Sun Microsystems Inc.//DTD JavaHelp Index Version 1.0//EN"
         "http://java.sun.com/products/javahelp/index_1_0.dtd">

<index version="1.0">
  <indexitem target="aerodynamics" text="aerodynamics"/>
  <indexitem text="airplane structure">
    <indexitem target="empennage" text="empennage"/>
    <indexitem target="fuselage" text="fuselage"/>
    <indexitem target="gear" text="landing gear"/>
    <indexitem target="powerplant" text="powerplant"/>
    <indexitem target="wing" text="wing"/>
  </indexitem>
  <indexitem target="airports" text="airports"/>
  <indexitem target="drag" text="drag"/>
  <indexitem text="four forces">
    <indexitem target="drag" text="drag"/>
    <indexitem target="lift" text="lift"/>
    <indexitem target="thrust" text="thrust"/>
    <indexitem target="weight" text="weight"/>
```

```
  </indexitem>
  <indexitem target="empennage" text="empennage"/>
  <indexitem target="powerplant" text="engine"/>
  <indexitem target="environment" text="environment, flight"/>
  <indexitem target="environment" text="flight environment"/>
  <indexitem target="fuselage" text="fuselage"/>
  <indexitem target="gear" text="gear, landing"/>
  <indexitem target="wind" text="indicators, wind"/>
  <indexitem target="intro" text="introduction"/>
  <indexitem target="airports" text="landing field"/>
  <indexitem target="gear" text="landing gear"/>
  <indexitem target="lift" text="lift"/>
  <indexitem target="markings" text="markings, runway and taxiway"/>
  <indexitem target="intro" text="overview"/>
  <indexitem target="powerplant" text="powerplant"/>
  <indexitem target="markings" text="runway markings"/>
  <indexitem target="runways" text="runways"/>
  <indexitem text="structure, airplane">
    <indexitem target="empennage" text="empennage"/>
    <indexitem target="fuselage" text="fuselage"/>
    <indexitem target="gear" text="landing gear"/>
    <indexitem target="powerplant" text="powerplant"/>
    <indexitem target="wing" text="wing"/>
  </indexitem>
  <indexitem target="empennage" text="tail"/>
  <indexitem target="taxiways" text="taxiway markings"/>
  <indexitem target="taxiways" text="taxiways"/>
  <indexitem target="thrust" text="thrust"/>
  <indexitem target="weight" text="weight"/>
  <indexitem target="wind" text="wind indicators"/>
  <indexitem target="wing" text="wing"/>
</index>
```

The structure of the index file is almost identical to that of the TOC file but with different element names. The XML declaration (`<?xml ...`) and the document type definition (`<!DOCTYPE ...`) are optional. The root element of the index file is named "index," specified with the `<index>` and `</index>` tags. The `<index>` start-tag includes an optional `version` attribute, where you can specify the JavaHelp map version number.

The `<index>` element contains a set of `<indexitem>` elements. Each `<indexitem>` element uses attributes to specify one TOC entry. For example:

```
<indexitem target="intro" text="introduction"/>
```

This defines the index entry `introduction`. When the user selects this entry, the HelpSet Viewer displays the topic with map ID `intro`.

The code for the Aviation JavaHelp sample index creates the index shown in Figure 5-5.

Figure 5-5. Aviation index

Notice that some secondary index items are nested within primary items. For example, the following code shows how five index items are nested under the primary item `airplane structure`:

```
<indexitem text="airplane structure">
  <indexitem target="empennage" text="empennage"/>
  <indexitem target="fuselage" text="fuselage"/>
  <indexitem target="gear" text="landing gear"/>
  <indexitem target="powerplant" text="powerplant"/>
  <indexitem target="wing" text="wing"/>
</indexitem>
```

The first line, which defines the primary index item, doesn't contain a `target` attribute to link to a help topic. It is customary for an index item not to link to a help topic if it contains subitems. In this example, if you want to associate an overview help topic for airplane structure, you can create a secondary index item and give it an overview name such as "about." The "about" index item provides the link to the overall airplane structure help topic.

Notice that all the index items are alphabetized by the primary index items. In turn, any secondary index items are alphabetized within their corresponding primary item. JavaHelp doesn't alphabetize these items automatically. You must be sure to arrange the items in alphabetical order when creating the index file.

Creating the Word-Search Index

The word-search index is the most complex JavaHelp navigation component. But it is the easiest to create, since JavaHelp includes commands for automatically generating it. Creating the word-search index can be as simple as running these commands to create the default JavaHelp search index. However, you may want to take advantage of advanced features, such as hooking up another Java-based search engine to perform word search functions.

The *jhindexer* command generates a word-search database. *jhindexer* searches all the files in the directory you specify and generates a database using the words it finds in these files. When the user enters a word-search request, the JavaHelp system looks in the database to determine the topic files in which the word or phrase is used.

Try using the *jhindexer* command to create a new word-search database for the Aviation HelpSet. Using a Unix shell window or a DOS command prompt window:

1. Go to directory *Aviation*, the master project directory for the Aviation project.
2. Delete or rename the *JavaHelpSearch* directory.
3. Enter this command to run *jhindexer* against the files in the *Topics* directory:

   ```
   C:\jh1.1\javahelp\bin\jhindexer Topics   (Windows)
   ```

   or:

   ```
   /jh1.1/javahelp/bin/jhindexer Topics   (Unix)
   ```

   These examples assume that you installed JavaHelp in directory *C:\jh1.1* (Windows) or */jh1.1* (Unix).
4. View the Aviation HelpSet in the HelpSet Viewer to see the search index that you created.

jhindexer creates subdirectory *JavaHelpSearch* in current directory, populating it with the database files that make up your HelpSet's word-search index.

WARNING Don't change the name of any file in the *JavaHelpSearch* directory.

The directory name *JavaHelpSearch* is also specified in the HelpSet file. It's the contents of the `<data>` element within the "Search" `<view>` element. (See the listing

in the previous section, "Creating the HelpSet File.") You can change the directory name and the HelpSet file specification, but I strongly recommend you don't.

You entered the *jhindexer* command with a single argument, specifying the *Topics* directory. The indexer recursively descends through the *Topics* directory hierarchy, searching the files and adding words to the word-search database. This is exactly the data that is appropriate to search: all the help topic files, but none of the image files or HelpSet data and navigation files. The convenience of having the indexer recursively descend through a single directory hierarchy is one of the reasons I strongly suggest placing all your topic files and subdirectories under one *Topics* directory.

NOTE As I discussed in Chapter 4, *Preparing Help Topics*, the word-search index uses the information you place in the `<title>` tags of your help topic files to assign names to the search results returned to a user conducting a search.

The *jhindexer* command offers several advanced options, which are discussed in Chapter 6.