JavaHelp™ 2.0
System User's Guide

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054 U.S.A.
650−960−1300

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON−INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software Clause at DFARS 252.227−7013 (Oct. 1988) and FAR 52.227−19 (c) (June 1987).

Table of Contents

Table of Contents

3 JavaHelp System Overview

Table of Contents

4 Authoring Help Information

Table of Contents

1 JavaHelp^{TM} System User's Guide

JavaHelp 2.0 − October 2004

This user's guide contains the following sections:

Release Information

A description of the contents of the JavaHelp release.

JavaHelp System Overview

A general overview of the JavaHelp system. Describes features, usage scenarios, and other technical details.

Authoring Help Information

A guide for help authors. Describes how to: set up a help system, use popups and secondary windows, use context−sensitive help, use full−text search, and package help information for delivery to users.

Programming with the JavaHelp System

A guide for developers. Describes how to: add JavaHelp to applications, implement context−sensitive help, embed JavaHelp components into applications, and develop lightweight Java components that can be added to help topics.

Localizing Help Information

A guide for localizers of JavaHelp systems.

- This guide is available in PDF format at:
  `doc\jhug.pdf`
- The JavaHelp helpset for this guide can be found at:
  `doc\jhug`

1.1 Keeping in Touch

The following is a list of ways to obtain and share information about the JavaHelp system.

1.1.1 Feedback

Comments and questions about how the JavaHelp system software works are welcome. Please review the FAQ at our home page, and if your question is not covered, send email by using the following web page:

`http://java.sun.com/docs/forms/javahelp−sendusmail.html`

If you have comments on the JavaHelp specification, instead of the web page above, please send comments to:

`jsr−97−comments@jcp.org`

Your email messages will be read. However, due to the large volume of email, we might not be able to respond personally.

1.1.2 Mailing List

The JavaHelp team maintains a mailing list for disseminating information about JavaHelp system updates and events.

To subscribe:

1. Send mail to `listserv@javasoft.com`.

2. In the body of the message type: `SUBSCRIBE JAVAHELP−INFO`.

1.1.3 Discussion group (JAVAHELP−INTEREST)

Sun maintains a mailing list as a JavaHelp community resource where interested parties can post and exchange information and inquiries about the JavaHelp system in a public forum. Subscribers to this list can receive inquiries either as they are posted or in regular digest versions.

To subscribe:

1. Send mail to: `listserv@javasoft.com`.

2. In the body of the message type: `SUBSCRIBE JAVAHELP−INTEREST.`

To view archives, manage your subscription, or to unsubscribe, go to:

`http://archives.java.sun.com/archives/javahelp−interest.html`

Web Site

Other information can be obtained at our web site at:

`http://java.sun.com/products/javahelp`

We hope to hear from you!

2 The JavaHelp 2.0 Release

This chapter describes the contents of the JavaHelp 2.0 release. In addition to the JavaHelp system libraries, the release contains a variety of demos, examples, and documentation.

How you explore and use the release depends on your interests. For example:

- If you are a developer interested in adding the JavaHelp system to an application, you might:
  - Run the demonstration programs and examine the source code for those programs.
  - Read the chapters of the *JavaHelp System User's Guide* that pertain to developers, especially Programming with the JavaHelp System.
  - Review the specification and the APIs.
- If you are an online help author you might:
  - Run the demonstration programs.
  - Read the chapters of the *JavaHelp System User's Guide* that pertain to help authors, especially Authoring Help Information.
  - Examine the sample helpsets.
  - Create a helpset and view it using the `hsviewer` command.

2.1 Contents of the Release

The single zip file installation contains the following files:

README file	Information about this release (text document).
LICENSE.html	The JavaHelp license agreement (html document).
JavaHelp System User's Guide (this document)	Available in both PDF (`doc\jhug.pdf`) and JavaHelp/HTML format (`doc\jhug`).
Specification	Defines the API between the application and the help system and the formats of the underlying files used by the JavaHelp system. (`doc\spec\JavaHelp_V2_0_Specification.pdf`)
API	`javadoc` generated documentation of the JavaHelp API. Can be viewed by using the JavaHelp API viewer or using a web browser (`doc\api\index.html`)
Libraries and tools	The libraries (`javahelp\lib`) and tools (`javahelp\bin`) used to create online help systems.
Demonstration programs	Programs that demonstrate JavaHelp system functionality. Source code for these programs illustrate how you can implement these features in your JavaHelp systems. (`demos\bin` and `demos\src`)
Sample Helpsets	Helpsets that you can use to familiarize yourself with the capabilities of the JavaHelp system. You can also use these helpsets as templates for creating your own help systems. (`demos\hs` and `demos\hsjar`)
JavaHelp source files	Source files for the JavaHelp system (except the full−text search engine). (`src.jar`)
DTDs	The DTDs (Document Type Definitions) that define the XML−based metadata files (helpset, map, TOC, Index) are included in `javahelp\lib\dtd`.

Default Style Sheet	The default style sheet for the J2SE 2.0 viewer is included in `doc\css`.

For a detailed list of the files included in the release, see List of Files in the Release.

2.2 Requirements

You must install the Java^{TM} 2 Platform, Standard Edition SDK (J2SE SDK) to be able to use JavaHelp 2.0.

The JavaHelp 2.0 release is an optional package of the J2SE SDK (JDK 1.2.2 and later). As a result, it works well with all versions of the J2SE. JavaHelp 2.0 does not work with JDK 1.1.x. We have tested this release against J2SE 1.2.2, 1.3.1, and 1.4.1 on the following systems:

- Windows 2000
- Solaris 2.6/SPARC
- Solaris 7/SPARC
- Solaris 8/SPARC
- Solaris 9/SPARC

2.3 New Features and Changes in JavaHelp 2.0

This page describes the most significant changes since the JavaHelp 1.1.3 release.

2.3.1 Native Browser Support

Release 2.0_02 enables you to use a native browser to display your help topics in the topic pane of the help window. This alternate content viewer uses the Java Desktop Integrated Component (JDIC) browser component `BasicNativeContentViewerUI` to render content by using the native browser. The use of `BasicNativeContentViewerUI` requires that the JDIC libraries be installed. See http://jdic.dev.java.net for details on how to integrate the JDIC components into your application.

In your applications, you use this browser by adding the following import statement and then putting the call to `setContentViewerUI` before the code that creates the `HelpBroker` or `JHelp` component (for a code example that creates a `HelpBroker`, see Adding the JavaHelp System to Applications).

```
import javax.help.SwingHelpUtilities;
...
SwingHelpUtilities.setContentViewerUI("BasicNativeContentViewerUI");
```

Additionally, you can use the command−line parameters `−ID` and `−contentViewer` to cause hsviewer to use the alternate browser and display a given ID in the helpset, as follows:

```
java −jar hsviewer.jar −helpset HolidayHistory −classpath\
../hsjar/holidays.jar −ID easter −contentViewer BasicNativeContentViewerUI
```

2.3.2 Installation Packages

JavaHelp is now released in a single zip file installation package. You must uninstall prior releases before unpacking V2.0 into a directory.

2.3.3 Running the JavaHelp Viewer

The JavaHelp viewer is now in the JAR file `hsviewer.jar`. To run it you must use a command similar to the following one:

```
   java −jar c:\jh20\demos\bin\hsviewer.jar
```

For more information, see Viewing Helpsets.

2.3.4 JDK 1.1 is no longer supported

The earliest version of the JDK that is supported is the Java™ 2 Platform, Standard Edition (J2SE™) JDK 1.2.2.

2.3.5 Change to API for Accessing Frames

In previous versions of JavaHelp, you could directly access the frame in which JavaHelp is displayed if you extended the DefaultHelpBroker. After extending DefaultHelpBroker, you were able access the protected JFrame frame field.

In JavaHelp 2, the frame field is no longer accessible. In applications that use JavaHelp 2, you will have to rewrite code that does this kind of frame access. Use the following methods:

```
WindowPresentation DefaultHelpBroker.getWindowPresentation();
Window WindowPresentation.getHelpWindow();
```

You will still be able to access the frame with the following method calls:

```
WindowPresentation pres = hb.getWindowPresentation();
Window win = pres.getHelpWindow();
```

2.3.6 Multi−Topic Printing

It is now possible to print more than one help page at a time. You can select a group of topics in a navigation pane, such as the TOC or index, and then choose the Print option to print them all out.

2.3.7 Comprehensive Merging Options

When helpsets are merged, there are new merge options available: UniteAppendMerge and SortMerge. UniteAppendMerge causes like items to be merged, making it possible, for example, to produce fully merged TOCs. SortMerge allows you to produce fully merged and canonically sorted indexes. For more information, see Merging Helpsets.

2.3.8 New Views in Help Viewer

In addition to the existing TOC, Index, and Search navigators, there are two new navigators that can be added to the navigation pane of the help viewer:

- **Glossary**. Short technical descriptions of terms can be put in an XML−based file with a format similar to the index file.
- **Favorites**. An XML−based collapsible and expandable display of the user's favorite topics.

2.3.9 Specifying View (Navigator) Icons or Text

When you specify a navigator by using the `<view>` tag in the helpset file, you can also specify the icon that displays in the tab above the navigator by using the `<image>` tag. For more information on setting view images, see the description of the `<view>` tags in The Helpset File.

You can display text instead of icons in the navigator tabs by setting the presentation tag's `displayviewimage` attribute to `"false"`. For more information on setting this attribute, see the description of the `<presentation>` tag in The Helpset File.

2.3.10 Presentation Controls

It is now possible to display help in various kinds of windows (called *presentations*) from a Java application or from a helpset's table of contents. There are presentation controls in the helpset file and navigator files, a `Presentation` class, and context−sensitive help changes that support the class. These features provide you with more options for displaying your help content in the JavaHelp Viewer, in a secondary window, or in a popup window.

Not only can you specify the type of presentation window, but for each type of window you can designate the buttons that appear in the toolbar. In addition to the standard Back, Forward, and Print buttons, you can display buttons like the Reload button (reloads the current topic) and the Home button (goes to the main topic defined for the helpset).

For more information, see the Presentation feature in Helpset File and Implementing Context−Sensitive Help.

2.3.11 Customizable Toolbar Support in Helpset File

When you define a presentation window, you can specify if it has a toolbar, and, if so, which controls appear on the toolbar. For more information, see the `<toolbar>` tag in Helpset File.

2.3.12 Server−Based JavaHelp

The JavaHelp V1.0 API provided an initial foundation for developing online help for server−based applications. JavaHelp 2 extends support for server−based applications with a standard for a JavaHelp bean and a Java Server Pages^{TM} (JSP) tag library for accessing helpset data. For more information, see Server−Based JavaHelp.

2.3.13 Helpset File has an Implementation Section

The implementation section of the helpset, enclosed in the `<impl>` tag, enables you to use special viewers based on a MIME type. It can also simplify using an external browser to display your helpsets rather than using the default helpset viewer included in the JavaHelp system.

Specifying an `<impl>` section in a helpset file creates a registry that provides key data mapping to define the HelpBroker class to use in the `HelpSet.createHelpBroker` method. The registry also determines the content viewer to user for a given MIME type.

`JEditorPane` uses this mechanism to link a given MIME type with an editor kit.

For example, the JavaHelp system gets its default content pane viewer for HTML by linking the `text/html` MIME type to `com.sun.java.help.impl.CustomKit`, which is an extension of the Swing `HTMLEditorKit`. If you wanted to display PDF in the content pane, you could write your own PDF editor kit, and then you could map the MIME type for PDF (`application/pdf`) to your new editor kit.

Another use for this feature might be to replace the current `HTMLEditorKit` (the content viewer) with a web browser without having to change code as you do in JavaHelp 1.

To see how to declare the tag, see the `<impl>` tag in Helpset File.

2.3.14 Dynamic Context−Sensitive Help for Components

You can assign help IDs based on cursor position, selection, or some other mechanism inherent in an object. For more information, see Dynamic Map ID Assignment in "Implementing Context Sensitive Help."

2.4 Demonstration Programs

This release includes a number of demo programs (including source code) and examples. These programs demonstrate JavaHelp system functionality and provide code examples that illustrate how you can implement these features in your JavaHelp systems. Many of these demonstration programs use the sample helpsets.

IDE demo
A mockup of an Integrated Development Environment application. The application includes a complete, functional help system.

Object demo
A demonstration of how the `<OBJECT>` tag can be used to embed Java components directly into HTML topic pages. The JavaHelp system includes two components, a popup window and a secondary window.

API viewer
A specialized help viewer that includes a navigator that displays JavaHelp system API documentation.

Merge demo
A small application that demonstrates the dynamic merging of helpset information.

Newmerge demo
An application and a set of help files that demonstrate new features for merging of helpset information, including SorteMerge merging for indexes and glossaries and UniteAppendMerge merging for TOCs.

Browser demo
A demonstration of the JavaHelp viewer running in an applet on a web browser.

Search example
Source file that describes a simple, alternate implementation of a full−text search engine.

Localized Helpsets
Localized demonstration helpsets (English, German, Japanese.

Demo source files
Source files for all of the demos can be found in:

JavaHelp_Home/demos/src

In addition to the locations described below, most of the demo programs are available by executing the JAR files. Depending on your operating system, you might execute a JAR file from the command line or by double−clicking the file's icon in the Windows Explorer. For instance, if your JAR files are set to execute with the Java runtime program `java.exe` on a Windows operating system, you can double click the jar file in the Explorer to run it.

The demo programs' jar files are located in the `demos/bin` folder.

2.4.1 IDE Demo

The IDE Demo is a mockup of an IDE (Integrated Development Environment) application that contains a complete, fully−featured help system.

The IDE demo demonstrates the following functionality:

Standard navigators
TOC, keyword index, and full−text search navigators.

Synchronization
The TOC display is synchronized with the content pane. The topic displayed in the content pane is always highlighted in the TOC.

Content pane

Help topics displayed in the content pane contain images, hyperlinks, and other 3.2 HTML tags.

Lightweight Java components

The top−level topic "Building Projects" uses the JHSecondaryViewer lightweight component to implement a *popup window*, and a *secondary window*. In this topic, click on the blue term "project" to see the popup window and click on the button at the end of the first paragraph to see the secondary window. These components are also demonstrated and described in the Object demo.

Presentation settings

The IDE demo has code in its helpset file (`IdeHelp.hs`) that sets up two presentation windows. It also has entries in its TOC file (`IdeHelpTOC.xml`) and its index file (`IdeHelpIndex.xml`) that specify the type of window (the presentation) in which to display the help topic. (For example, see the TOC entry "Beans In JDE − in SecondaryWindow".) For more information on presentations, see the presentation feature in Helpset File and in Implementing Context−Sensitive Help.

Context Sensitive Help

The JavaHelp system supports three types of context−sensitive help:

Window−Level Help

Place the cursor in different areas of the application and press the F1 key. This activates the help viewer and displays help that describes the GUI object that currently has focus.

Field−Level Help

Click the ↖? button, then move the field−level cursor ↖? over an object in the GUI and click select.

Help Button

Choose Edit > Find. Click on the Help button in the dialog box to display a topic about the dialog box.

Custom Navigator

Choose Help > Java API Reference to activate the help viewer and load a helpset that includes the customized `ClassViewer` navigator. The `ClassViewer` is an example of a navigator that is customized to assist in navigating through `javadoc`−generated API documentation. The behavior of the `ClassViewer` is described at API Viewer. It is a fully functional navigator with its own specialized format.

Embedded Help

Choose Window > Class Inspector to activate the custom `ClassViewer` navigator described above and embed it in the application. Note that as you navigate, the corresponding topic is displayed in the lower pane of the application.

▶ To run the IDE demo, enter the following command at the command line:

```
java −jar JavaHelp_Home/demos/bin/idedemo.jar
```

After the demo starts, choose Help > Demo JDE − Help to activate the help viewer.

2.4.2 Object Demo

Lightweight Java components can be added to HTML topic pages using the `<OBJECT>` tag. The JavaHelp system includes popup and secondary windows implemented using the JHSecondaryViewer component. The object demo shows how popup and secondary windows work.

▶ To run the Object demo, enter the following command at the command line

```
java −jar JavaHelp_Home/demos/bin/object.jar
```

2.4.3 API Viewer

The JavaHelp release includes an *API Viewer*. This simple application is similar to the helpset viewer, except that the `CLASSPATH` includes a special ClassViewer navigator. The API Viewer presents `javadoc`−generated information derived from the JavaHelp source files.

The TOC in the API Viewer (the ClassViewer navigator) includes a top pane and a bottom pane. The top pane displays a list of the classes, interfaces, and exceptions that comprise the JavaHelp system. The bottom pane displays a hierarchy of components of a selected class, interface, or exception. Select a class in the top pane to view its constituent components in the bottom pane. Choose a component in the bottom pane to view it in the content viewer.

The data used by the ClassViewer Navigator is generated using a *doclet* (doclets are a feature of the Java^{TM} 2 SDK). The API doclet is not included in this release of the JavaHelp system.

▶ To run the API Viewer demo, enter the following command at the command line:

```
java −jar JavaHelp_Home/demos/bin/apiviewer.jar
```

2.4.4 Merge Demo

Demonstrates JavaHelp system helpset merging capabilities. For details, see the online help available from the Help menu in the `merge` demo program.

▶ To run the Merge demo, enter the following command at the command line:

```
java −jar JavaHelp_Home/demos/bin/merge.jar
```

2.4.5 Newmerge Demo

Demonstrates JavaHelp system helpset merging capabilities. For details, see the online help available from the Help menu in the `newmerge` demo program.

▶ To run the Newmerge demo, enter the following command at the command line:

```
java −jar JavaHelp_Home/demos/bin/newmerge.jar
```

2.4.6 Browser Demo

Demonstrates how the JavaHelp system can be used with applet−based applications running in web browsers. You can run demonstration by using either Netscape Navigator or Internet Explorer. The demo creates an applet button on an HTML page. When you click the button, the JavaHelp viewer displays the Holiday helpset. This demo requires some setup. You can find instructions for the demo in the file `demos/browser/demo_instructions`.

2.4.7 Search Example

The file `demos/src/sunw/demo/searchdemo/ClientSearch.java` shows how to extend the `HelpSearch` class to implement an alternate search engine.

2.4.8 Localized Helpsets

Localized demonstration helpsets (English, German, Japanese) for the IDE demo are included in the `demos/hsjar` folder. Be sure to choose the appropriate font in the JavaHelp viewer's Set Fonts dialog box (choose Options > Set Font).

You must have installed the appropriate Unicode fonts to be able to view the Japanese helpset.

Also, you use a different command to run helpsets. The command is:

```
java -jar hsviewer.jar [-helpset hs_name]
```

For example (on a Windows system):

```
C:\> java -jar c:\JavaHelp\demos\bin\hsviewer.jar
          -helpset c:\JavaHelp\demos\hs\newmerge\MergeHelp.hs
```

2.5 Sample Helpsets

This release includes a variety of helpsets packaged in different ways. Many of these helpsets are used by the demonstration programs, but you can also use the `hsviewer` executable jar file to view them.

Helpsets are described in detail in Authoring Help Information.

You can view these helpsets to familiarize yourself with the capabilities of the JavaHelp system. You can also use these helpsets as templates for creating your own help systems. The helpsets are described below.

2.5.1 JavaHelp System User's Guide

The *JavaHelp System User's Guide* (this document) is also available as a JavaHelp helpset. It is located in `doc\jhug`.

2.5.2 History of the Holidays

This helpset is interesting because it shows how a help system can be set up with two different TOCs. This helpset can be found in the following location:

```
JavaHelp_Home/demos/hsjar/holidays.jar
```

▶ To view this helpset on a UNIX system, use the following command:

```
java -jar JavaHelp_Home/demos/bin/hsviewer.jar \
     -helpset JavaHelp_Home/demos/hsjar/holidays.jar
```

▶ To view this helpset on a Windows system, use the following command in a Command window:

```
java -jar JavaHelp_Home\demos\bin\hsviewer.jar -helpset JavaHelp_Home\demos\hsjar\holidays.jar
```

2.5.3 IDE Demo

This helpset is used in the IDE Demo demonstration program. It is located in

```
demos/hsjar/idehelp.jar
```

▶ To view this helpset on a UNIX system, enter the following command at the command line:

```
java -jar JavaHelp_Home/demos/bin/hsviewer.jar \
     -helpset JavaHelp_Home/demos/hsjar/idehelp.jar
```

▶ To view this helpset on a Windows system, use the following command in a Command window:

```
java -jar JavaHelp_Home\demos\bin\hsviewer.jar -helpset JavaHelp_Home\demos\hsjar\idehelp.jar
```

2.5.4

2.5.5 Localized Helpsets

Two localized helpsets, one in German and another in Japanese, are included in the release. For more information about localizing helpsets, see Localizing Help Information.

To view these helpsets, you must have installed the correct fonts on your system.

2.5.5.1 German

This helpset is a portion of the `IDE demo` helpset that has been localized in German. It is located in

```
JavaHelp_Home/demos/hsjar/idehelp_de.jar
```

▶ To view this helpset on a UNIX system, enter the following command at the command line:

```
java -jar JavaHelp_Home/demos/bin/hsviewer.jar \
     -helpset JavaHelp_Home/demos/hsjar/idehelp_de.jar
```

▶ To view this helpset on a Windows system, use the following command in a Command window:

```
java -jar JavaHelp_Home\demos\bin\hsviewer.jar -helpset JavaHelp_Home\demos\hsjar\idehelp_de.jar
```

2.5.6

2.5.6.1 Japanese

This helpset is a portion of the `IDE demo` helpset that has been localized in Japanese. To view the Japanese helpset, you must have installed the correct Unicode fonts. This helpset is located in

```
JavaHelp_Home/demos/hsjar/idehelp_ja.jar
```

▶ To view this helpset, enter the following command at the UNIX command line:

```
java -jar JavaHelp_Home/demos/bin/hsviewer.jar \
     -helpset JavaHelp_Home/demos/hsjar/idehelp_ja.jar
```

▶ To view this helpset on a Windows system, use the following command in a Command window:

```
java -jar JavaHelp_Home\demos\bin\hsviewer.jar -helpset JavaHelp_Home\demos\hsjar\idehelp_ja.jar
```

2.6 The JavaHelp Libraries and Tools

The JavaHelp libraries and tools are located in the `javahelp` folder of the release. All classes work with the Java 2 Platform, Standard Edition (J2SE) and use Swing 1.2 or later (included in the J2SE).

2.6.1 Libraries

The JavaHelp classes are distributed in the following four JAR files located in the `javahelp\lib` folder:

`jh.jar` The standard library, which includes everything needed to use the standard navigator types (TOC, index, full−text search).

`jhbasic.jar` A subset of `jh.jar` that does not include support for the full−text search engine. This subset might be useful for very simple help systems that do not require a full−text search database or for help systems in which size is critical.

`jhall.jar` Contains all the JavaHelp system classes, including the tools required to create a search database.

`jsearch.jar` Contains the default full−text search engine used in the JavaHelp system.

2.6.2 Tools

The JavaHelp tools are used to view helpsets and to build and query the full−text search database. These tools are located in the `javahelp\bin` folder.

jhindexer Command−line program that creates the full−text search database used by the JavaHelp system full−text search navigator to locate matches.

jhsearch Command−line program that queries the JavaHelp system full−text search database that is created with the `jhindexer` command. You can use `jhsearch` to test a search database without invoking the help viewer.

2.7 Limitations and Bugs

This release of JavaHelp has the following limitations and bugs.

2.7.1 HTML Viewer

The JavaHelp HTML viewer is based on the Swing JEditorPane component. HTML rendering can differ depending on which version of Swing your application uses. Differences between versions are noted below.

2.7.1.1 Images Distorted

Occasionally, images are distorted (stretched). Redisplaying the page corrects the problem.

You can sometimes avoid this problem by explicitly specifying "height" and "width" attributes with the `<img>` tag. For example,

```
   <img src="../../images/hg_note.gif" width="18" height="13">
```

2.7.1.2 Classpath Limitations

Due to Java security protocols, it is not possible to reference images and files from your topics that are outside the `CLASSPATH` of your application (or `hsviewer`).

For example, you start `hsviewer` with the following command:

```
   java -jar c:\JavaHelp\demos\bin\hsviewer.jar -helpset
C:\my_app\help\myhelpset.hs
```

The `hsviewer` application sets the `CLASSPATH` to be:

```
   C:\my_app\help
```

You cannot reference files above the `C:\my_app\help` folder. For example, in the following code an image in `C:\my_app\images` referenced as follows cannot be displayed:

```
   <IMG SRC="../../images/foo.gif">
```

You can work around this problem by using the `-classpath` parameter of `hsviewer.jar`. The parameter allows you to specify a `CLASSPATH` separately from the helpset file, enabling you to set the `CLASSPATH` to include the folder that contains the image and specify the helpset file relative to that folder. For example, you could enter the following command (all on one line) at the command line:

```
    java -jar c:\JavaHelp\demos\bin\hsviewer.jar -helpset C:\my_app\help\myhelpset.hs
    -classpath C:\my_app
```

2.7.1.3 Duplicate Lines Displayed (J2SE 1.2.2)

If a TOC, or index entry points to an anchor target specified at or near the top of the page (in the first scroll zone), the viewer can position the lines incorrectly, resulting in lines' being displayed twice.

2.7.1.4 Anchor Targets

There are two problems with anchors:

- On J2SE 1.2.2 systems only, if the TOC or index is used to access a topic file that contains anchor targets in the first scroll zone in the viewer, text will be duplicated in the display.
- Named anchors cause a space to be added at the beginning of the object that follows them.

The best way to work around this problem is to nest the text of the target within the anchor tag. For example:

```
  <H2><a name="widgets">Working With Widgets</a></H2>
```

2.7.1.5 Cascading Style Sheets

Tag names in styles and style sheets *must* be specified with lowercase letters or they will be ignored.

2.7.1.6 <sup> and <sub> Tags (J2SE 1.2.2)

The `<sup>` `<sub>` tags are ignored on J2SE 1.2.2 systems.

2.7.1.7 The Width Attribute of the `<td>` Tag

The width attribute of the <td> tag is ignored in J2SE 1.2. That version of the viewer assigns its own width to table columns.

On J2SE 1.2.2, the width attribute works when specified in absolute pixels (px). The use of percentages (%) is not supported in that version of J2SE.

2.7.1.8 Named Anchors in Ordered and Unordered Lists

If the first item after a list tag (`<ul>`, `<ol>`, or `<dl>`) is a named anchor (`<a name>`), the list shown in the following example is rendered incorrectly:

```
<ul>
  <a name="17539"> </a>
  <li>Transmitter reports
  <a name="17540"> </a>
  <li>Channel reports
</ul>
```

The following list is rendered correctly:

```
<ul>
  <li><a name="17539">Transmitter reports</a></li>
  <li><a name="17540">Channel reports</a></li>
</ul>
```

2.7.1.9 TABS in `<pre>` Tag not Recognized

TABS used in text enclosed in `<pre>` tags are not recognized. Space characters are recognized correctly.

2.7.1.10 Viewer Cannot Load Image Files Directly

The help viewer aborts if you attempt to load a graphic file (*.gif, *.jpg) directly. You must include the images in an HTML file by using the `<img>` tag.

2.7.1.11 Page Setup Settings not Preserved (Printing)

Changes made to the default settings in the Page Setup dialog box are not preserved between activations. The default settings are always set upon activation.

2.7.2 Full−text Search

The text search feature, implemented by running `jhindexer` on your helpset, has the following limitations and bugs.

2.7.2.1 Parsing of Asian Languages

The J2SE word−break iterator that the JavaHelp search indexer and search navigator use to parse Asian (Japanese, Chinese, Korean, Thai) languages uses a heuristic that is not well suited to searching. As a result, topic files are not parsed into words that users are likely to enter into the Find input field.

However, because the parser works on the same model used to highlight words when the user double−clicks in the content pane, as a workaround (albeit an inconvenient one), the Asian language user can conduct a full−text search as follows:

1. Double−click a word in the content pane.
2. Copy and paste the word into the search navigator Find field.

3. Press Return.

2.7.2.2 Match Limit

To enhance full−text search performance, the search navigator reports the 100 most relevant matches. For example, in the `idedemo` program, if you search for the word "build", you see that different forms of the word (builder, built, builds) are not highlighted because the 100 match limit was met with the exact match "build". This limit should not be a problem with more complex, multi−word, natural language queries.

2.7.2.3 jhindexer Does Not Parse `"."` Correctly

The `jhindexer` does not treat the `"."` character correctly. As as result, a search for `"javax.help"` in the `apiviewer` returns no matches.

2.7.3 Context Sensitive Help

2.7.3.1 F1 Help (Solaris OpenWindows)

On Solaris OpenWindows manager the F1 key does not get help on the the component with focus.

2.7.4 Other Bugs

2.7.4.1 Copy/Paste on Solaris

On Solaris systems, follow these steps to copy and paste text from the help viewer:

1. Highlight text in the viewer.
2. Type Control−C to copy the text.
3. With focus in the target Solaris window, press the Paste key.

2.7.4.2 `jar:` Protocol

Due to a bug, the Java^{TM} 2 SDK `jar:` protocol does not permit relative references to JAR files. Instead, they must be fully qualified. For example, the following code works correctly:

```
jar:file://c:/my_app/help.jar!map.jhm
```

There is no way to make that reference relative from the location of a helpset file. For that reason, you must include the helpset and map files in the JAR file with the rest of the helpset.

2.7.4.3 Index Navigator

If an index entry contains more that two hierarchical levels, a "turner" mechanism (like the one used in the TOC) is added to the second +*n* levels.

2.7.4.4 Popup Window Accessibility

JavaHelp popup windows are not as accessible as they should be due to a bug in the underlying AWT classes that prevents the popups from obtaining focus. Popup windows can be dismissed by pressing the F10 key − the Esc key does not work because the window cannot obtain focus. In addition, this same bug prevents scrollbars in popup windows from being accessible from the keyboard; therefore, it is important to set the size of popups to enable all the information to be displayed in a single scroll zone.

2.7.4.5 Fonts and Localization

There are limitations in this release on the ability to display fonts in the help viewer content pane. Due to a bug in the J2SE, the only character encoding that can be displayed in the HTML content pane is the system default. Different locales that use that encoding are rendered correctly. `

2.8 List of Files in the JavaHelp 2.0 Release

The libraries included in this release support the Java^{TM} 2 Platform. They run on any Java platform that is compliant with Java 2 . Executable JAR files are included for tools and demonstration programs. They can be executed only on the Java 2 Platform. They will not work with JDK 1.1 systems.

`README` − Initial README file
`LICENSE.html` − License file
`src.jar` − JavaHelp system source files

`doc\` − Documentation
`doc\images` − Images used in documentation
`doc\jhug` − Helpset folder for the JavaHelp System User's Guide
`doc\jhug.pdf` − PDF version of the JavaHelp System User's Guide

`doc\api` − JavaHelp 1.1 javadoc API documentation
`doc\css\default.css` − `JEditorPane` default stylesheet
`doc\spec\JavaHelp_V2_0_Specification.pdf` − Latest version of the spec

`javahelp\` − JavaHelp system binaries and libraries system release
`javahelp\lib\` − JavaHelp libraries
`javahelp\lib\jh.jar` − Standard JavaHelp libraries
`javahelp\lib\jhbasic.jar` − Subset of jh.jar that does not include search engine
`javahelp\lib\jhall.jar` − Everything
`javahelp\lib\jsearch.jar` − The default search engine only

`javahelp\lib\dtd` − DTDs for JavaHelp system XML metadata files
`javahelp\lib\dtd\helpset_2_0.dtd` − Helpset file DTD
`javahelp\lib\dtd\index_2_0.dtd` − Index file DTD
`javahelp\lib\dtd\map_2_0.dtd` − Map file DTD
`javahelp\lib\dtd\toc_2_0.dtd` − TOC file DTD

`javahelp\bin\` − JavaHelp executable programs
`javahelp\bin\jhindexer` − Creates the search database
`javahelp\bin\jhsearch` − Queries the search database

`demos\` − Demos
`demos\browser` − JavaHelp viewer running in an applet on a web browser
`demos\README` − Instructions for building the demos
`demos\bin` − Demonstration programs
`demos\bin\apiviewer.jar` − JavaHelp API viewer
`demos\bin\hsviewer.jar` − Helpset viewer
`demos\bin\idedemo.jar` − Starts a mockup of an IDE
`demos\bin\merge.jar` − JH v1 helpset merging demo
`demos\bin\newmerge.jar` − Demonstrates new features of helpset merging
`demos\bin\object.jar` − Demonstrates popup and secondary window functionality
`demos\bin\UserGuide.jar` − Displays the JavaHelp System User's Guide

`demos\lib` − JARs containing different extensions
`demos\lib\classviewer.jar` − Classviewer (used in apiviewer and idedemo)

`demos\lib\searchdemo.jar` − Alternate search engine (includes documentation)

`demos\hs` − Expanded (unJARed) helpsets
`demos\hs\merge` − For the merge demo
`demos\hs\newmerge` − For the newmerge demo

`demos\hsjar` − Demo helpsets in JARs
`demos\hsjar\animals.jar` − animals.hs helpset used by newmerge demo
`demos\hsjar\apidoc.jar` − api.hs helpset
`demos\hsjar\holidays.jar` − HolidayHistory.hs helpset
`demos\hsjar\idehelp.jar` − IdeDemo helpset
`demos\hsjar\idehelp_de.jar` − Localized helpset (German)
`demos\hsjar\idehelp_en.jar` − Localized helpset (English)
`demos\hsjar\idehelp_ja.jar` − Localized helpset (Japanese)
`demos\hsjar\invertebrates.jar` − invertebrates.hs helpset used by newmerge demo
`demos\hsjar\object.jar` − Object demo helpset
`demos\hsjar\vertebrates.jar` − vertebrates.hs helpset used by newmerge demo

`demos\serverhelp` − Server help demo

`demos\src` − Sources for the demos (J2SE 1.2)
`demos\src\sunw\demo\browser` − Browser demo
`demos\src\sunw\demo\classviewer` − Classviewer Navigator used in apiviewer
`demos\src\sunw\demo\idedemo` − Mockup of an IDE using JavaHelp
`demos\src\sunw\demo\jhdemo` − hsviewer demo `.java` files
`demos\src\sunw\demo\merge` − JH v. 1 example of how to merge helpsets
`demos\src\sunw\demo\newmerge` − Uses new merge features to merge helpsets
`demos\src\sunw\demo\object` − Object demo
`demos\src\sunw\demo\searchdemo` − Alternative search engine

3 JavaHelp System Overview

This overview consists of the following sections:

Introduction

General introduction to the JavaHelp system.

JavaHelp System Features

A brief overview of the main features of the JavaHelp system.

Descriptive Scenarios

Scenarios that illustrate many ways the JavaHelp system can be used with different applications in a variety of network environments.

JavaHelp System Lightweight Components

A brief description of the lightweight component functions provided with the JavaHelp system.

3.1 Introduction

Most interactive applications require online help and Java applications are no exception. The JavaHelp^{TM} system is specifically tailored to the Java^{TM} platform. The JavaHelp system provides developers and authors a standard, fully−featured, easy−to−use system for presenting online information to Java application users. Providing a help system that is a standard extension to the Java^{TM} 2 Software Development Kit, Standard Edition (the J2SE^{TM} SDK), relieves developers and authors of having to implement their own proprietary help systems.

The JavaHelp system consists of a fully featured, extensible specification and API, and a reference implementation of that specification and API that is written entirely in the Java language.

- The JavaHelp reference implementation, based on the Java Foundation Classes (JFC, also known as *Swing*), provides a standard interface that enables both application developers and authors to add online help to their applications.
- The specification and API enable developers to customize and extend the help system to fit the style and requirements of their applications.

In addition, the JavaHelp system has been designed to work especially well in a variety of network environments. The JavaHelp system is platform independent and works in all browsers that support the Java platform.

The JavaHelp system enables Java developers to provide online help for:

- Applications (both applet and standalone)
- JavaBeans^{TM} components
- Applets in HTML pages
- Server−based Java applications

Authoring support for the JavaHelp system is available through online help authoring tool vendors. Tool vendors, including Software 7, Quadralay, eHelp, Brown Inc, Paradigm Systems, SolutionSoft, ComponentOne, and Oracle, have products that provide authoring support for the JavaHelp system.

⏩ **Next Overview Topic:** JavaHelp System Features

3.2 JavaHelp System Features

This section describes the main features of the JavaHelp system. For a list of new features in this release, see New Features in JavaHelp 2.0.

3.2.1 Help Viewers

There are three types of windows in which you can display your help topics. These windows can be specified in the Java program when it makes a call to the help system. The help author can set various attributes of these windows in the helpset (`.hs`) file. These windows cannot be invoked from a link in a help topic (although there is a way to link to a popup window that uses a different technique, described in Opening Popup and Secondary Windows From an HTML Topic). However, it is possible to open a topic in one of these windows from the table of contents in the tri−paned viewer (see the Presentation feature in Helpset File, the `presentationtype` and `presentationname` descriptions in the sections on the TOC, index, and glossary navigators, and the section Implementing Context−Sensitive Help.

The windows are:

- Main window (by default, a tri−paned help viewer)
- Secondary window
- Popup window

3.2.1.1 Main Window

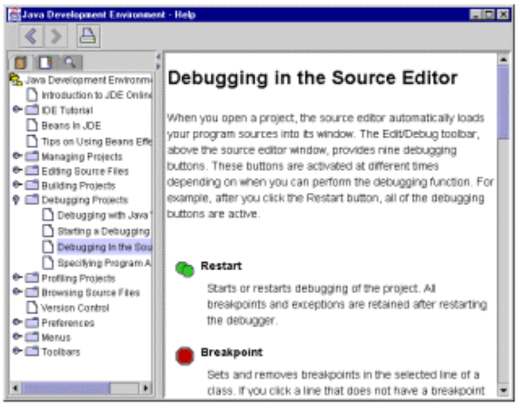
The standard JavaHelp system main window has three panes, is not destroyed when you exit the window, and is configurable (see the presentation feature in Helpset File). By default, a main window has the following three panes:

Toolbar — A bar over the navigation and content panes that can be configured to display various toolbar buttons, such as Back, Forward, and Print.

Navigation pane — A tabbed interface appearing on the left that allows users to switch between the table of contents, index, and full text search displays.

Content pane — A pane on the right that displays help topics formatted with HTML 3.2 or later, plus embedded lightweight Java components.

The following figure shows a help window with a toolbar that has three buttons in it, a table of contents in the navigation pane on the left, and in the content pane, a help topic, "Debugging in the Source Editor":



Debugging in the Source Editor

When you open a project, the source editor automatically loads your program sources into its window. The Edit/Debug toolbar, above the source editor window, provides nine debugging buttons. These buttons are activated at different times depending on when you can perform the debugging function. For example, after you click the Restart button, all of the debugging buttons are active.

Restart

Starts or restarts debugging of the project. All breakpoints and exceptions are retained after restarting the debugger.

Breakpoint

Sets and removes breakpoints in the selected line of a class. If you click a line that does not have a breakpoint

3.2.1.2 Secondary Window

By default, this window contains a single pane, a help content viewer that shows a help topic. It is similar to the content pane of the tri−paned viewer. A secondary window has a name and can be configured to have a navigation pane and a toolbar. If the window is already open, its contents are replaced if the Java program uses it again. When the user closes a secondary window, it is destroyed (unlike the main window, which is not destroyed on exit).

3.2.1.3 Popup Window

This type of window stays open as long as it has focus. When the user clicks elsewhere, the window is destroyed. A popup has only one pane, a content viewer.

3.2.2 Table of Contents

Provides a collapsible and expandable display of topics in the help system. Supports unlimited levels and merging of multiple TOCs. The underlying file format follows World Wide Web Consortium (W3C) Extended Markup Language (XML) standards. The TOC display is synchronized with the content viewer: The topic being displayed is highlighted in the TOC. For more information, see Table of Contents File.

3.2.3 Index

Supports merging of multiple indexes. The underlying file format follows W3C XML standards. The index display is synchronized with the content viewer: The topic being displayed is highlighted in the index. For more information, see Index File.

3.2.4 Full−Text Search

The full−text search engine can be used in a variety of network environments. Matches returned from searches are ranked for relevancy by using "relaxation rules." For more information, see Full−Text Search.

3.2.5 Compression and Encapsulation

The standard JAR format is used to encapsulate the help information into a single, compressed file. The JavaHelp system works equally well with help information that is not compressed into JAR files − this flexibility allows authors to view files during development without taking the time to compress them.

3.2.6 Embeddable Help Windows

Help windows (individually or in combination) can be embedded directly into application interfaces.

3.2.7 Context−Sensitive Help

Help can be activated from Java programs through a number of different mechanisms. For more information, see the help authoring section Context−Sensitive Help and the Java developer section Implementing Context−Sensitive Help.

3.2.8 Flexible Packaging

Flexible packaging of help information for product delivery makes it easy to incrementally update help information in the field.

3.2.9 Customization

The JavaHelp system is designed to permit great flexibility in customizing both the user interface and functionality.

3.2.10 Merging

Help information from different sources can be combined and presented to the end user. For more information, see Merging Helpsets.

3.2.11 JavaBeans Support

The JavaHelp API enables a JavaBeans component to specify help information that can be presented to the end user (perhaps merged with additional information).

⏩ **Next Overview Topic:** Descriptive Scenarios

3.3 Descriptive Scenarios

The following scenarios illustrate some of the many ways the JavaHelp system can be used to provide online help for different types of Java programs in a variety of network environments. These scenarios attempt to illustrate the JavaHelp system's flexibility and extensibility.

Scenarios are presented in three areas:

Invocation mechanisms	Scenarios that describe different ways that the JavaHelp system can be invoked from applications.
Presentation and deployment	Scenarios that describe different ways that the JavaHelp system can be used to present help information. These scenarios also illustrate different methods for deploying the JavaHelp system classes and help data.
Full−text search	Scenarios that describe different ways that full−text search of help information can be implemented.

⏩ **Next Overview Topic:** Invocation Mechanisms

3.4 Invocation Mechanisms

Users invoke online help from within applications in a number of ways. This section describes invocation methods available through the JavaHelp system.

3.4.1 Menus and Buttons

Online help is often invoked when a user chooses an item from a Help menu or clicks on a Help button in an application GUI.

The JavaHelp system provides a simple interface by which an application requests that a topic ID be displayed. The JavaHelp system then associates the topic ID with the appropriate URL and displays it. IDs are mapped to URLs in a JavaHelp system metadata file called the *map file*.

For example, when coding a file chooser dialog box, a developer requests that the topic ID `fc_help` be displayed when the Help button at the bottom of the dialog box is clicked. In the map file the ID `fc_help` is defined to be a file named `FileChooser.html` using the following XML syntax:

```
<mapID target="fc_help" url="html/help/FileChooser.html" />
```

Separating the specification of file names (or URLs) from the program code provides content authors the freedom to control the information that is associated with the topic ID.

3.4.2 Tooltips

A *tooltip* is a brief message presented to the user when the cursor remains over a button for an interval longer than a given threshold.

Although tooltip information could be included in the JavaHelp system data, it will usually be delivered as part of the application and will be co−located with the code. Tooltip functionality is provided as a component in Swing.

3.4.3 Context−Sensitive Help

The JavaHelp system provides the ability to invoke online help that describes graphical components in an application GUI. The user makes gestures that activate context−sensitive help and then specifies the component in question. The ID associated with the component is displayed.

3.4.4 Viewer Initiated Help

You can display help topics from a TOC or index navigator or from the content pane of the main window. By default, an index entry and a TOC entry display help in the main window's content viewer. You can also define an entry that displays help in a popup or a secondary window. In the content pane, you can define an <object> tag that displays help in a separate popup or secondary window. If you use a standard HTML link in the content pane, the linked help topic replaces the current one in the content pane.

3.4.5 System Initiated Context−Sensitive Help

The following invocation would display system−initiated help in a main window.

```
mainHelpBroker.setCurrentID(helpID);
```

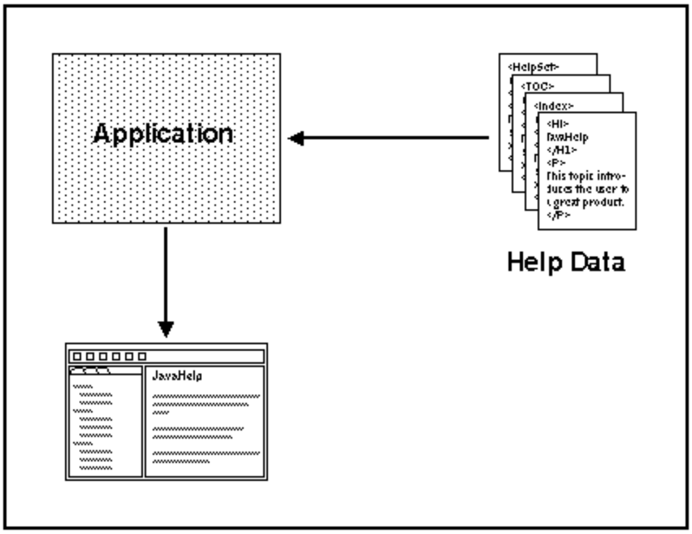
Next Overview Topic: Presentation and Deployment

3.5 Deploying and Presenting JavaHelp Helpsets

The JavaHelp system is designed to be deployed in a number of different types of applications and in a variety of different network environments. The following scenarios illustrate some of the different ways that the JavaHelp system can be used to present and deploy information.

3.5.1 Standalone Application

A standalone Java application runs independently of a web browser. In this scenario the Java application runs locally and accesses help data installed on the same machine.

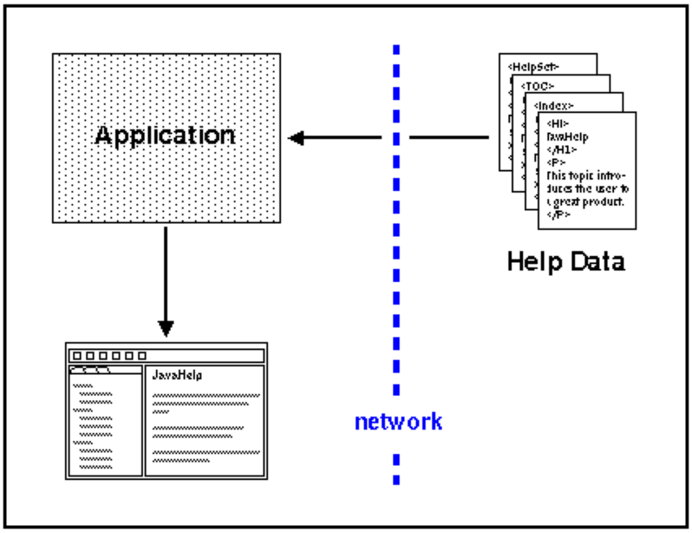


The application:

1. Requests the creation of a `JavaHelp` instance.
2. Loads the help data in that instance.
3. Presents the requested help topic.

3.5.2 Network Application

The JavaHelp system enables an application to transparently load help data from networks (intranet and Internet). When the help data is accessed across a network, the scenario is essentially the same as in the standalone scenario − the location of the data is transparent to the application.

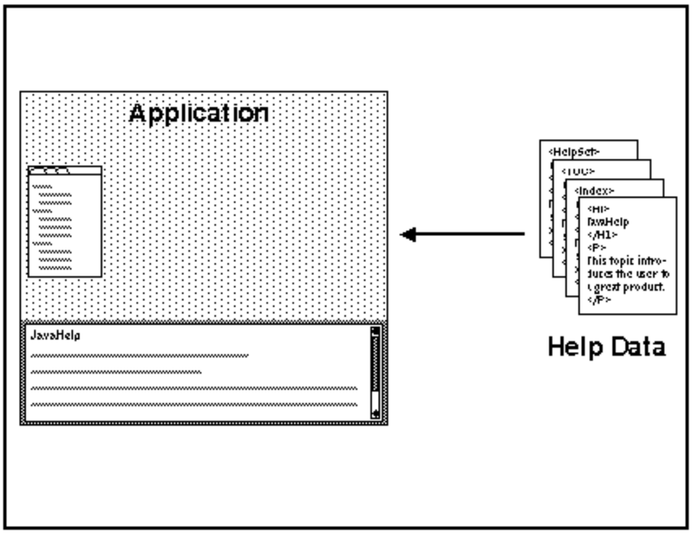


The application:

1. Requests the creation of a `JavaHelp` instance
2. Loads the help data from the network
3. Presents the requested help topic

3.5.3 Embedded Help

Both navigational and content information can be embedded directly in application windows. Embedding is accomplished by adding the JFC components that implement JavaHelp system components directly into the application frame.



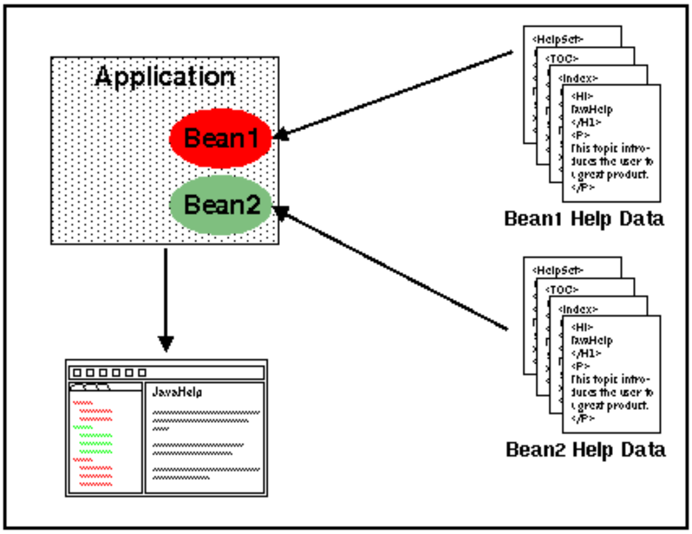
In this illustration, the content viewer is embedded along the bottom of the application window, and the navigation viewer is embedded in a different portion of the window.

The application can directly control the contents of the content viewer by programmatic means. Likewise, JavaHelp system navigators can be used to control information displays other than the JavaHelp system content viewer.

3.5.4 Component Help

Many modern applications are composed of a collection of interacting components. Examples range from large applications like Netscape Navigator™ (with plugins), to applications where JavaBeans components are connected together using JavaScript™ or Visual Basic™.

In the case of JavaBeans, each component can be shipped with its own help data. The following illustrates such a case.



In this case, the help information from the red JavaBean (Bean1) and from the green JavaBean (Bean2) is merged in the help viewer table of contents. The merge operation can be performed by the developer ahead of time, or completed when the application or JavaBeans component is installed by the user.

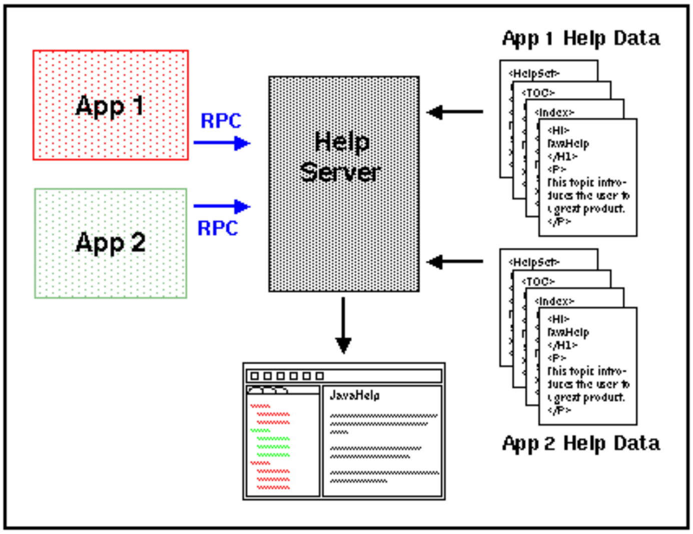
In version 1 of the JavaHelp software, merging is accomplished by appending TOC and index information and searching merged full−text search databases.

3.5.5 Help Server

In some environments, it is useful to separate the process that presents the help information from the application. For example:

- Applications that are written in a language other than the Java language (for example, C, C++, Visual Basic) can use the JavaHelp system to display online help when deployed on diverse computing platforms.
- A suite of applications might be installed together or separately. In this case the help server can be used to display help for the entire suite, rather than each of the constituent applications providing their own help system.

In the following scenario, applications not written in the Java language make requests to a JavaHelp system process (help server) through an RPC mechanism (the RPC might be wrapped in a library and be invisible to the application developer).

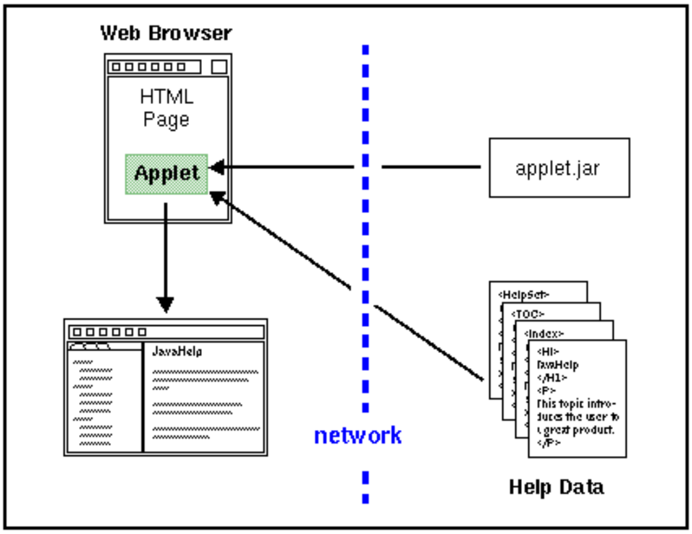


3.5.6 Browser−Based Applications (Applets)

Applications that run in browsers have a number of unique deployment issues that the JavaHelp system addresses. The following three scenarios illustrate how the JavaHelp system can be used in three of the most common cases. In the following scenarios an applet or some other triggering entity on an HTML page requests the JavaHelp system to display help information.

3.5.6.1 Applet(1)

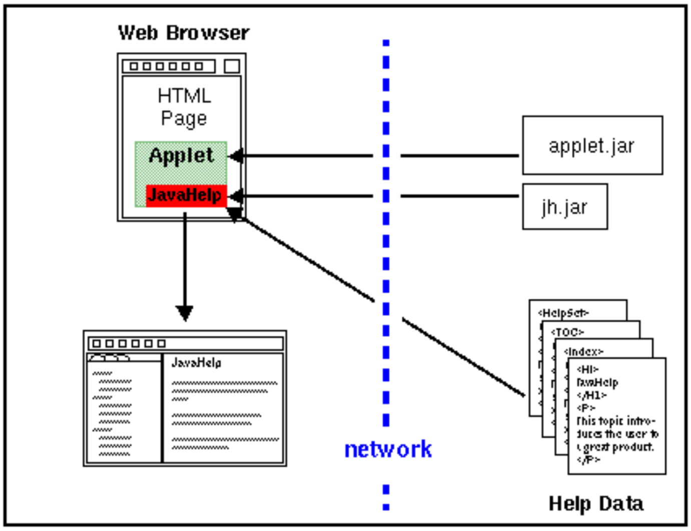
In the first scenario, the browser contains a customized implementation of the JavaHelp system and an appropriate version of the JRE (Java Runtime Environment). This JRE might have been delivered with the browser, or it might have been downloaded by the client into the `CLASSPATH`. The implementation can use the JavaHelp system content pane, or it can use the HTML viewer that is part of the web browser.



1. The HTML page that contains the applet tag is loaded into the browser.
2. The applet is downloaded from the server and executed.
3. The user requests help.
4. The applet forwards the request to the JavaHelp system.
5. Help data is downloaded from the server and displayed in the JavaHelp system viewer (or browser window).

3.5.6.2 Applet(2)

In the second scenario, the JavaHelp system classes are downloaded along with the applet. Because the JavaHelp system is an optional package of the Java release, it is possible that a fully compliant Java^{TM} 2 SDK browser might not have the the JavaHelp system classes in its `CLASSPATH`. In this case the JavaHelp system classes must be downloaded from the server. Since the JavaHelp system is quite small, this approach is often practical. Browsers might provide additional means for installing extensions downloaded through this mechanism.



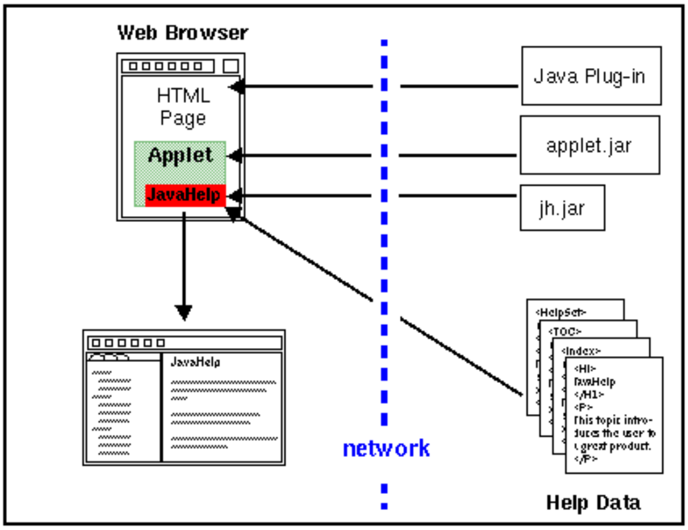
1. The HTML page that contains the applet tag is loaded into the browser.
2. The applet is downloaded from the server and executed.
3. JavaHelp system classes are downloaded from the server.
4. The user requests help.
5. The applet forwards the request to the JavaHelp system.

6. Help data is downloaded from the server and displayed in the JavaHelp system viewer (or browser window).

3.5.6.3 Applet(3)

The third scenario describes the case in which the applet is downloaded to a browser environment that has neither the appropriate JRE nor the Javahelp system installed.

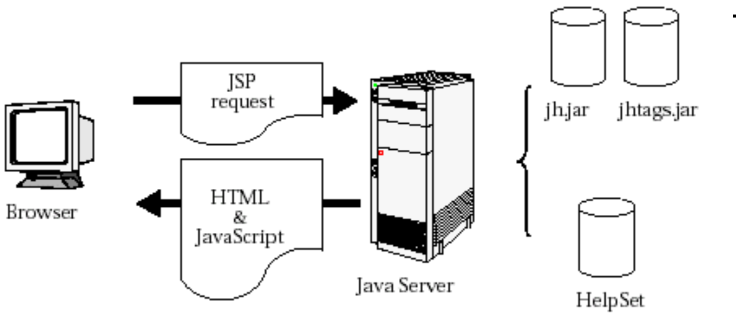
In this case, the Java^{TM} Plug−in can be used to download the required JRE and the JavaHelp system standard extension classes. The Java Plug−in allows developers to specify a specific JRE on the HTML page that is required to run their applet. If the correct JRE is not present on the user's system, the Java Plug−in software downloads the correct JRE and installs it on the user's system. The new JRE is subsequently available to any applet that requires it. Because the JavaHelp system is a standard extension to the Java platform, the JavaHelp system classes can be downloaded along with the JRE.



1. The HTML page that contains the applet tag is loaded into the browser
2. The Java Plug−in is downloaded. It prompts user to download appropriate JRE and JavaHelp system classes.
3. JRE and JavaHelp system classes are downloaded from the server.
4. Java Plug−in starts the JRE.
5. The applet is downloaded from the server and executed.
6. The user requests help.
7. The applet forwards the request to the JavaHelp system.
8. Help data is downloaded from the server and displayed in the JavaHelp system viewer (or browser window).

3.6 Server−based JavaHelp Helpsets

By combining the JavaHelp software API with new JavaHelp JSP tag libraries, web developers are now able to provide help for server−based applications that provide HTML pages to a browser. The diagram below illustrates the architecture.



A browser initiates a JSP request. Examples of a JSP request are displaying the help content in the helpset, the navigators, or the data for a given navigator. Typically, the JSP request contains JavaBeans^{TM} components as well as JSP tag extensions. The Java^{TM} server turns the request into a Java Servlet. The servlet access the appropriate information from the helpset by using the classes in the JavaHelp library (`jh.jar`) and the JavaHelp tag library (`jhtags.jar`) and returns HTML and possibly JavaScript or dynamic HTML (DHTML) to the browser.

Next Overview Topic: Full−text Search

3.7 Full−text Search

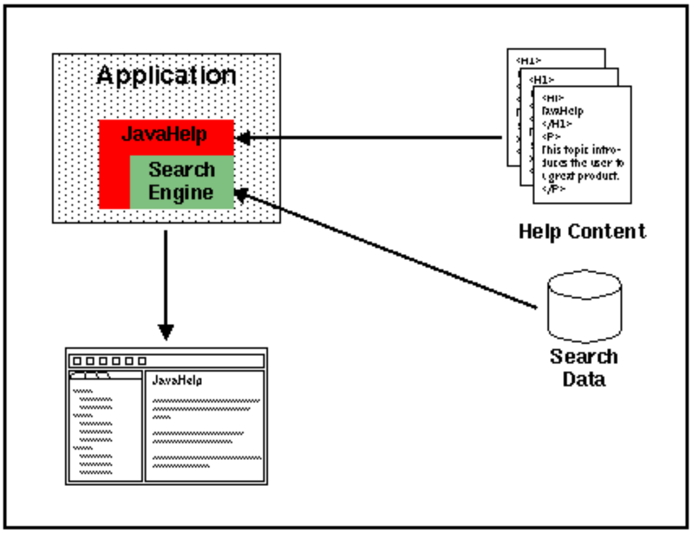
The JavaHelp system includes a full−text search facility that is fully−featured, compact, fast, and extremely flexible. The JavaHelp system is shipped with a search database indexer. Help authors use the search database indexer to create a compact database that is distributed with the application's help data. When a user initiates a search, the search engine searches the database to determine matches. Alternative search engines can be substituted for the standard JavaHelp system search engine.

The following scenarios illustrate some of the different ways that the full−text search can be used. Three scenarios are presented:

- Standalone
- Client−side
- Server−side

3.7.1 Standalone

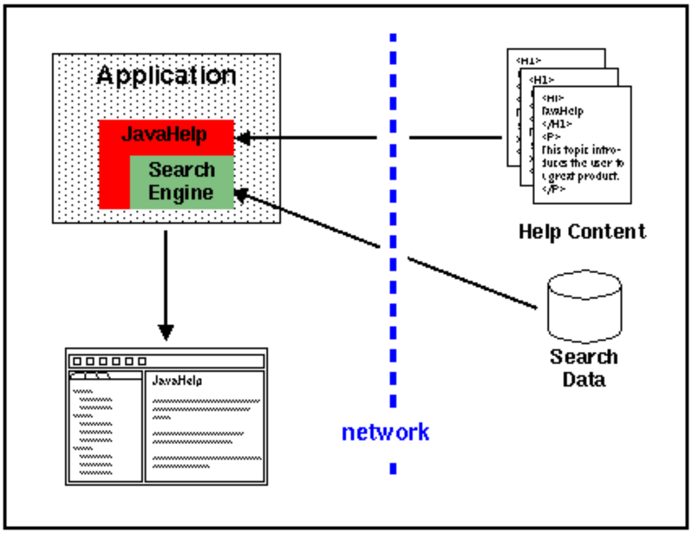
In a standalone search, all of the components (search engine, search database, and help topics) are local to the application.



1. Search is initiated
2. Search engine loads database
3. Search engine searches database and delivers “hits” to application
4. User (or application) chooses a “hit”
5. Content is loaded and displayed

3.7.2 Client−Side

From an implementation point−of−view, the client−side search is identical to the previously described standalone search except that the components are downloaded from a server. This arrangement is common with browser−based applications (applets), where the help data usually resides on the same server as the applet code. When a search is initiated, the search data is downloaded from the server, read into the browser's memory, and searched. The topic files are downloaded only when they are presented.

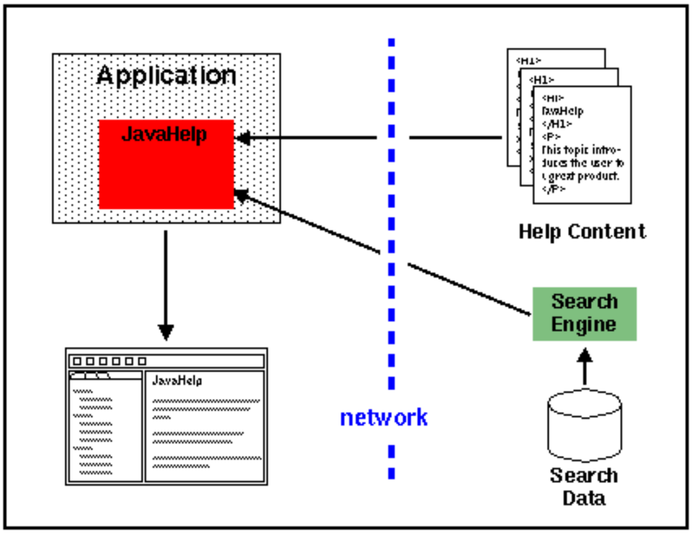


1. Search is initiated
2. Search engine loads database
3. Search engine searches database and delivers “hits” to application
4. User (or application) chooses a “hit”
5. Content is loaded and displayed

During the initial search, time is required to download the search database. Once downloaded, the data can be kept in memory or in a temporary file on the client machine and the searches are quite fast.

3.7.3 Server−Side

In a server−side search, the search data, topic files, *and the search engine* are all located on the server side − only the results of the search are downloaded to the client.



1. Search is initiated
2. JavaHelp requests search from server
3. Server-side search engine searches database and delivers "hits" to application
4. User (or application) chooses a "hit"
5. Content is loaded and displayed

This option also works well for applets. It permits developers to use alternate search engines (for example, AlltheWeb, Google, or Lycos) and can be quicker to start because the search database is not downloaded. (It is especially fast if the search engine is already running on the server). Note that this approach works very well with Java servlets.

Next Overview Topic: JavaHelp System Lightweight Components

3.8 JavaHelp System Lightweight Components

Lightweight components can add functionality to help topics. They are similar to Java applets, but load and execute more quickly.

A help author can use a lightweight component that is already implemented in the JavaHelp system. This component implements popup windows and secondary windows. To use this lightweight component in an HTML topic file, you use the HTML `<object>` tag as described in Calling Popup and Secondary Windows From an HTML Topic.

In addition, it is possible to call popup and secondary windows from Java programs and from the TOC and index views of the JavaHelp viewer. For more information on this technique, see Opening Popup and Secondary Windows with the Presentation Manager.

A Java developer can create new lightweight components as well. For example, a such a component might add functionality like animation and multimedia to help topics. The Java^{TM} 2 Platform includes high quality audio as well as a video viewer that supports the common formats. For more information see Creating Lightweight Components.

4 Authoring Help Information

The topics in this section of the *JavaHelp System User's Guide* describe the aspects of the JavaHelp software that are of primary interest to online help authors. These topics assume that the author is responsible for creating the metadata files that the JavaHelp software uses to present information, as well as the topics that inform the application's users. Together the metadata and topic files are referred to as a *helpset*.

If you use a help authoring tool to create your help system, some of the details described in this chapter are managed for you by the tool.

The following list summarizes the tasks required to create a helpset. All these tasks are described in this section of the *JavaHelp System User's Guide*:

- Create HTML topics
- Create a helpset file
- Create a map file
- Create a table of contents file
- Create an index file
- Create a full−text search database
- Compress and encapsulate the help files into a JAR file for delivery to customers

Consider the following strategy as a way to get started:

1. Use the demonstration programs included with the release to acquaint yourself with JavaHelp system features. The most useful demo for this purpose is `idehelp`. This program demonstrates a fully functional help system.
2. Read the remainder of this chapter of the *JavaHelp System User's Guide*, and explore the helpset files created for the `idehelp` demo (extract the files in `demos\hsjar\idehelp.jar`).
3. Create your own helpset as follows:
    1. Create HTML topics using an editor of your choosing.
    2. Copy the metadata files you extracted from `demos\hsjar\idehelp.jar` (`IdeHelp.hs`, `IdeHelpIndex.xml`, `IdeHelpTOC.xml`, and `Map.jhm`) to the folder that contains your topic files and rename them appropriately.
    3. Edit the metadata files to match your help information.
    4. Optionally create a full−text search database.
    5. Use the `hsviewer` command to display your helpset.

4.1 Viewing Helpsets

A helpset viewer is provided with the release to enable you to view your helpsets. If the path to the `java` executable file is in your PATH variable, you can enter the following command in your operating system's command−line shell to run the helpset viewer (where *JavaHelp_home* is the JavaHelp system installation directory):

```
java −jar JavaHelp_home/demos/bin/hsviewer.jar
```

On a Windows system, you can configure a shortcut to run the viewer. For example, if your J2SE installation is in `C:\j2sdk1.4.1` and your JavaHelp system installation is in `C:\JavaHelp`, you can configure the shortcut as follows:

1. If necessary, create a shortcut on your desktop.
    1. Open the file Explorer and navigate to a folder containing an executable file (for example, `c:\j2sdk1.4.1\bin\java.exe`).
    2. Right−click the file and choose Create Shortcut.

    3. Drag the newly created shortcut to your desktop.
2. Right−click the shortcut and choose Properties.
3. In the Shortcut dialog, type the following command in the Target field:

   ```
   c:\j2sdk1.4.1\bin\java −jar c://JavaHelp//demos//bin//hsviewer.jar
   ```
4. Save the shortcut.
5. You can then double−click the shortcut to open the viewer in the Java virtual machine.

All helpsets are displayed in the JavaHelp system help viewer—the same viewer used to provide help in applications.

- If you want the viewer to find the default helpset, add its path to the `CLASSPATH` system variable.
- For the purpose of running the examples, the JRE you use to run `hsviewer` does not have to be the same version as the JRE on which your application is deployed. For example, you can use J2SE 1.4.1 to run the demo programs (including `hsviewer`) even if your application is deployed on J2SE 1.3.1.
- For a list of limitations, bugs, and "idiosyncrasies" that pertain to the JavaHelp system HTML viewer, see Limitations and Bugs.
- The viewer toolbar does not include a reload button. The easiest way to reload a file after you change it is to click the viewer's "previous" and "next" buttons.

4.1.1 Displaying a Helpset with `hsviewer.jar`

To display a specific helpset, start the helpset viewer `hsviewer.jar` as described above. When the viewer opens, either click the Browse button to navigate to a helpset or, in the URL field, type the full path to the helpset file. When the helpset has loaded, click Display to view the helpset in the viewer.

Alternatively, you can specify the helpset by using a command−line switch with `hsviewer.jar`. You can do this on the command line itself, in a batch file, in a script file, in a JAR file, or in a shortcut. The command−line syntax of the `hsviewer.jar` command−line interface is:

```
java −jar hsviewer.jar [−helpset hs_name]
```

`−helpset` Specifies the helpset name:

hs_name The full path to a helpset file. For example (on a Windows system):

```
C:\> java −jar c:\JavaHelp\demos\bin\hsviewer.jar
         −helpset c:\JavaHelp\demos\hs\newmerge\MergeHelp.hs
```

4.1.2 Displaying a Helpset in Windows by Clicking the `.hs` File

In Windows, you can open a file in a program by double−clicking the file in the Explorer. For example, if you double−click an HTML file, it opens in your default web browser. This technique works because Windows can associate a file extension (like `.html`) with a program that displays the file. You can use this technique to open a helpset file in the helpset viewer (`hsviewer.jar`). Here is how to do it.

1. Create a batch file that runs `hsviewer.jar` and accepts a command−line parameter.

   For example, if the JavaHelp system is installed in `c:\JavaHelp` and `java.exe` is in your PATH, you would put the following line in the file:

```
java −jar c://JavaHelp//demos//bin//hsviewer.jar −helpset %1
```

2. Save the file as `hsviewer.bat`.

3. Open Windows Explorer, navigate to a directory containing a helpset (`.hs`) file, and double−click the file.

   Windows displays the Open With dialog, which enables you to associate the helpset file with a program that opens the file.

4. Click the Other button, navigate to the directory where you saved the `hsviewer.bat` file, and choose that file as the one that will open `.hs` files.

5. Click OK in the Open With dialog.

6. The helpset opens in the helpset viewer.

   In the future, when you double−click a helpset file in the Explorer, it will open in the helpset viewer.

4.1.3 Displaying a Helpset by Using an Executable JAR File

You can display a specific helpset in a standalone environment by using the `sunw.demo.jhdemo.Runner` class available in `hsviewer.jar`, specifying that class and some other information in a manifest file, and creating an executable JAR file from the manifest file.

To create a JAR file that can display a single helpset:

1. Create a manifest file for the helpset.

   For example, shown below is the manifest file that displays the *JavaHelp System User's Guide*. (See the `UserGuide.jar` file in the *JavaHelp−Home*`/demos/bin` directory.)

   ```
   Main−Class: sunw.demo.jhdemo.Runner
   Run−Class: sunw.demo.jhdemo.JHLauncher
   Class−Path: ../../javahelp/lib/jh.jar hsviewer.jar ../../doc/jhug/
   Arguments: −helpset jhug.hs
   ```

   The syntax of the manifest file is as follows:

   `Main−Class: sunw.demo.jhdemo.Runner`
   `Run−Class: sunw.demo.jhdemo.JHLauncher`
   `Class−Path:` *jar−file* **|** *directory*
   `Arguments: −javahelp` *helpset−filename*

`Main−Class:` The main class to execute when running this JAR file. This class is a standard argument for executable JAR files, and in this case it must always be `sunw.demo.jhdemo.Runner`. For example:

```
Main−Class: sunw.demo.jhdemo.Runner
```

`Run−Class:` The class that the `sunw.demo.jhdemo.Runner` executes. This class is usually `sunw.demo.jhdemo.JHLauncher`, but it could be another class that launches a JavaHelp viewer. The following code is the typical `Run−Class` entry:

```
Run−Class: sunw.demo.jhdemo.JHLauncher
```

`Class−Path:`

Files to use with the `Run−Class`. Specify them as a series of space−separated JAR files or directories to be added to existing `CLASSPATH` environment variable.

Note: The files must be relative to the location of the executable JAR file. For example, if the JAR file is in the `demos/bin` directory of the JavaHelp installation directory, this entry would be::

```
  Class−Path: ../../javahelp/lib/jh.jar hsviewer.jar
../../doc/jhug/
```

`Arguments:` Arguments to be passed to the `Run−Class` when `Run−Class` is executed. These arguments are specific to the `Run−Class`. For example:

```
−helpset jhug.hs
```

2. Create an executable JAR file by using the `jar` command. The JAR file contains only one file, the manifest file. Other files are specified in the `Class−Path` argument and must be relative to the JAR file. The command to create the manifest file is:

   ```
    jar cmf manifest_file jar_file
   ```

   For example:

   ```
   jar cmf manifest.mf UsersGuide.jar
   ```

3. Run the JAR file by opening it in `java.exe` or executing it from the Windows Explorer (if you have set up `java.exe` as the application that executes JAR files). For example:
   - In the command shell, change to the *JavaHelp−home*`/demos/bin` directory and enter the following command:

     ```
     java −jar UserGuide.jar
     ```
   - In Windows Explorer, navigate to the directory containing the JAR file and double−click it to display the help file.

     Note: If double−clicking the file opens it in WinZip or some other zip utility, right−click the file, choose Open, and then navigate to the `java.exe` program in your J2SE installation's bin directory.

4.2 Accessibility and JavaHelp Viewer Shortcut Keys

The JavaHelp^{TM} system tri−pane viewer is designed to be accessible. Most features in the help viewer are accessible from the keyboard.

The first time you open the help viewer, the focus should be on the default entry in the TOC navigator. Sometimes you cannot tell where the focus is because of a bug that can occur when the Back button is unselected. If you cannot tell where the focus is, use the following procedure to establish focus again:

1. Press Ctrl−F1 to display alternate text for a toolbar button. This action works only if the current focus is on a toolbar button.
2. If no alternate text displays, press F6 to see if the focus moves to an item in the current navigator.
3. If the focus does not move to the navigator, press F6 once more (because focus might have moved to the Content pane). Focus should now be in the Navigation pane.
4. With focus in the Navigation pane, if you want to be absolutely sure where you are, you can repeatedly press Shift−Tab followed by Ctrl−F1 until alternate text displays for a toolbar button.

4.2.1 Traversing the Viewer

If the focus is in the toolbar buttons and you press the Tab key to traverse through the viewer, the traversal order is as follows:

1. The toolbar buttons from left to right
2. The currently open tab in the Navigation (left) pane
3. The contents of that navigator

   Behavior in the navigator varies depending on whether or not the navigator has a Search field. It will take up to four more presses of the Tab key to move to the Content pane. Pressing F6 instead will move focus directly to the Content (right) pane.
4. The topic in the Content (right) pane

When focus is in the Content pane, pressing the Tab key has no effect. The Tab key does not wrap from the Content pane back to the toolbar. You can traverse backwards from the Content pane by pressing F6 to move to the Navigation pane, and then pressing Shift−Tab repeatedly.

4.2.2 Traversing and Using the Toolbar Buttons

You can move the focus between the toolbar buttons by pressing either the Left Arrow and Right Arrow keys or the Tab and Shift−Tab keys. If you want to stay in the toolbar and move from button to button, the Arrow keys are the best choice. If you want to be able to move focus out of the toolbar, use the Tab and Shift−Tab keys.

When a button has focus, you can perform two actions:

- Pressing Spacebar activates the button (for example, activating the Print button prints the currently selected topic or topics).
- Pressing Ctrl−F1 displays the button's alternate text.

4.2.3 Traversing and Using the Navigators

The Navigation pane has a tabbed interface. If focus is on the last toolbar button, pressing the Tab key takes you to the currently selected Navigator tab. At that point, you can either press the Tab key to enter the navigator or press Right Arrow or Left Arrow to select a different navigator. The behavior of the focus depends on the type of navigator, as follows:

4.2.3.1 TOC, Favorites, and Glossary Navigators

- If you use an Arrow key to move to the navigator from another Navigator tab, the focus moves automatically to an entry in the navigator.
- If you use a Tab key to move to the navigator (for example, from a toolbar button), the focus remains on the Navigator tab. If you press the Tab key, the focus moves to an entry in the navigator.
- In the navigator list, if you want to move to the Content pane, using the F6 key produces the most predictable results. The Tab key has varying results depending on whether or not there are scrollbars. Pressing the Tab key once changes focus to the vertical scrollbar if there is one. Pressing the Tab key again changes focus to the horizontal scrollbar if there is one. Pressing Tab again changes focus to the Content (right) pane. If you want to move to the Content pane, pressing the F6 key produces the most predictable results.
- In any Navigator list the following keys move focus from item to item and change the corresponding topic in the Content (right) pane:
    - ♦ The Up Arrow and Down Arrow keys move the focus from item to item in the navigator.
    - ♦ The Page Up and Page Down keys move the focus through the navigator a page at a time.
    - ♦ The Home and End keys move the focus to the beginning and end of the list.
    - ♦ The Right Arrow key opens a folder and the Left Arrow key closes a folder.

    - ♦ Ctrl−Right Arrow scrolls left and Ctrl−Left Arrow scrolls right.
    - ♦ F8 selects the splitter bar between the Navigator pane and the Content pane. With the splitter bar selected, you can move it right and left with the Right Arrow and Left Arrow keys.
- The F6 key changes focus to the Content pane.

4.2.3.2 Index and Search Navigators

These navigators have a text field for the search entry at the top of the navigator content pane, so they behave differently from the other navigators.

- If you use an Arrow key to move to the navigator from another Navigator tab, the focus moves immediately to the Search field.
- If you use a Tab key to move to the navigator (for example, from a toolbar button), the focus remains on the Navigator tab. If you press the Tab key again, the focus moves to the Search field.
- In the Search field, if you press the Tab key, the focus moves to the navigator list.
- In the navigator list, if you want to move the focus to the Content pane, using the F6 key produces the most predictable results. The Tab key has varying results depending on whether or not there are scrollbars. Pressing the Tab key once changes the focus to the vertical scrollbar if there is one. Pressing the Tab key again changes the focus to the horizontal scrollbar if there is one. Pressing Tab again changes the focus to the Content (right) pane.
- In any Navigator list the following keys move the focus from item to item and change the corresponding topic in the Content (right) pane:
    - ♦ The Up Arrow and Down Arrow keys move the focus from item to item in the navigator.
    - ♦ The Page Up and Page Down keys move the focus through the navigator a page at a time.
    - ♦ The Home and End keys move the focus to the beginning and end of the list.
    - ♦ The Right Arrow key opens a folder and the Left Arrow key closes a folder.
    - ♦ Ctrl−Right Arrow scrolls left and Ctrl−Left Arrow scrolls right.
    - ♦ F8 selects the splitter bar between the Navigator pane and the Content pane. With the splitter bar selected, you can move it right and left with the Right Arrow and Left Arrow keys.
- The F6 key changes the focus to the Content pane.

4.2.4 Traversing and Using the Content Pane

The Content pane, on the right side of the viewer, displays the current help topic. You can get to this pane by navigating to the Navigation (left) pane and pressing F6. When you are in the Content pane, you can use the following keys to move around and perform actions:

- Right Arrow and Left Arrow move the focus left or right one letter at a time. You cannot see the focus change.
- Ctrl−Right Arrow and Ctrl−Left Arrow move the focus left or right one word at a time. You cannot see the focus change.
- Up Arrow and Down Arrow move the focus up and down one line at a time. You cannot see the focus change.
- Page Up and Page Down move the the topic up and down a page at a time.
- Ctrl−Home and Ctrl−End move focus to the beginning and end of the topic.
- Ctrl−T selects the next link. Ctrl−Shift−T selects the previous link.
- Shift−Spacebar activates the selected link and goes to the linked topic.
- F8 selects the splitter bar between the Navigator pane and the Content pane. With the splitter bar selected, you can move it right and left with the Right Arrow and Left Arrow keys.
- F6 changes the focus to the Navigation (left) pane.

4.2.5 Shortcut Key Table

The following table lists the keystrokes you can use to navigate through the help viewer.

Keys	**Action**
Tab	Shifts focus to the next component. Does not work if the current focus is the Content pane.
Shift−Tab	Shifts focus to the previous component. Does not work if the current focus is the Content pane.
Up Arrow	Selects the previous item in a Navigator list. Moves the focus to the previous line in a topic in the Content pane. Moves the splitter bar left.
Down Arrow	Selects the next item in a Navigator list. Moves the focus to the next line in a topic in the Content pane. Moves the splitter bar right.
Left Arrow	Shifts focus to the previous tab or the previous button. Closes a folder in a Navigator list. Moves one character to the left in a topic in the Content pane. Moves the splitter bar left.
Right Arrow	Shifts focus to the next tab or the next button. Opens a folder in a Navigator list. Moves one character to the right in a topic in the Content pane. Moves the splitter bar right.
Ctrl−Left Arrow	Scrolls to the left in a Navigator list or in the Content pane. Moves one word to the left in a topic in the Content pane.
Ctrl−Right Arrow	Scrolls to the right in a Navigator list or in the Content pane. Moves one word to the right in a topic in the Content pane.
Home	Selects the first item in a Navigator list.
End	Selects the last item in a Navigator list.
Ctrl−Home	Selects the first line in a topic in the Content pane.
Ctrl−End	Selects the last line in a topic in the Content pane
Page Up	Scrolls up one page.
Page Down	Scrolls down one page.
Ctrl−T	Shifts focus to the next link in a topic in the Content pane.
Ctrl−Shift−T	Shifts focus to the previous link in a topic in the Content pane.
Ctrl−Spacebar	Follows a link in a topic in the Content pane.
Spacebar	Activates the toolbar button that has focus.

Ctrl−F1	Displays alternate text for a toolbar button.
F6	Shifts focus between the Navigation (left) pane and the Content (right) pane.
F8	Selects the splitter bar if focus is in the Content pane or the Navigation pane.

4.3 Setting Up Your JavaHelp Project

There are two primary things to consider when you set up your help projects:

- How best to organize help information files to organize them logically and conveniently for *authoring*
- How to organize help information to best *package* it for delivery to your users.

4.3.1 Authoring

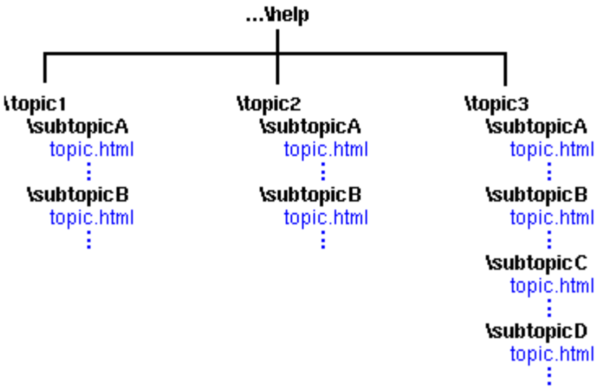
The JavaHelp system is *file−based* − topics are contained in files that are displayed in the help viewer one file at a time. It is a good idea to group related topics together to keep them organized and to make it as easy as possible to link the topics together.

The JavaHelp system uses URLs. A URL can resolve to the contents of a file in a file system, a file on the web, or a portion of a JAR file, or it can even be generated dynamically by a server.

You might also consider organizing topics to make it easy to package them into a compressed JAR file for delivery to your users.

For both these reasons, it is usually best to organize your topics in a folder hierarchy that you can "tear off" and place in the JAR file.

The following diagram shows such a hierarchy:



4.3.1.1 Links

The destination of a link to another topic in the helpst should be specified *relative* to the file that contains the link. The following is an example of such a relative link:

```
<A HREF="../subtopicB/topic.html">new topic</A>
```

Do not specify links to other topics by using an absolute link. For example, the following link uses a full path name to the destination of the link:

```
<A HREF="C:/product/help/subtopicB/topic.html">new topic</A>
```

Only relative links remain valid when the topic hierarchy is packaged into a JAR file and then installed on the user's computer.

4.3.1.2 File Separators ("/" vs. "\")

All files in a JavaHelp system are specified as URLs, which use the forward slash ("/") as the separator between elements (files) in a hierarchy. Although in some cases a backslash ("\") works on Windows platforms, if files that contain such references are installed on a different platform, the references no longer work.

4.3.2 Packaging

In addition to the topic files, the help information includes *metadata* files that contain information about the help system. Where you locate these metadata files can affect how you package, deliver, and update the help information for your users.

In JavaHelp systems there are two kinds of metadata:

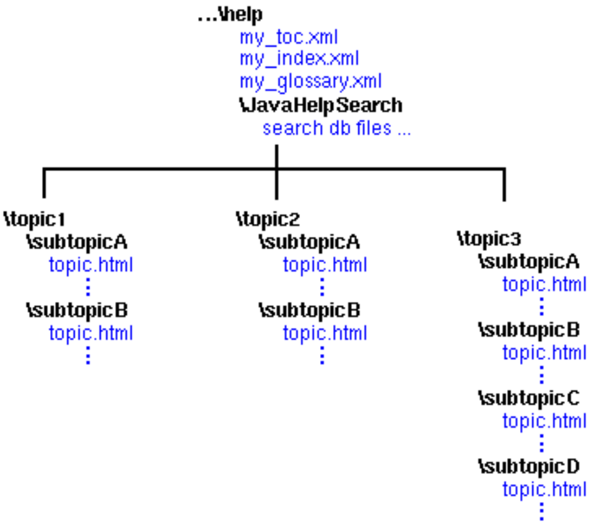
- Navigational data
- Helpset data

4.3.2.1 Navigational Data

Navigational data files contain information that is used by the JavaHelp system navigators. The standard JavaHelp system navigators are:

- Table of contents
- Index
- Full−text search
- Glossary

Each of these navigators has a metadata file associated with it that contains navigational data. These metadata files should be located in close proximity to the topic files to make it easier to package them into JAR files with the topic files for delivery to customers. The following diagram displays an example.



4.3.2.2 Helpset Data

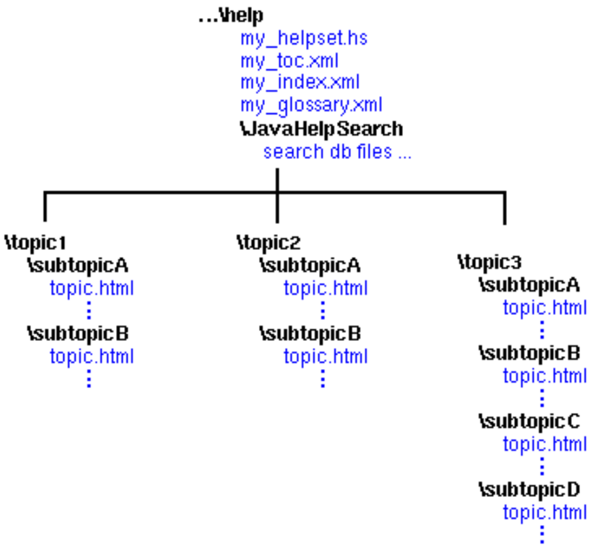
Helpset data is information that the JavaHelp system needs to run your help system. It is contained in two files:

- Helpset file
- Map file

When the JavaHelp system is activated by your application, the first thing it does is read the helpset file. The helpset file contains all the information needed to run the help system. As you can imagine, your application must be able to find the helpset file after the product is installed on your user's system.

The helpset file contains the location of the map file and in most cases, the map file is read when the helpset is initialized. The map file is used to associate topic IDs with URLs (paths to HTML topic files).

The following diagram shows how a help hierarchy might be set up to include the helpset file and map file.



4.3.3 Packaging a Helpset into a JAR File

You can package your help information into JAR files for delivery to your users. Usually, you package the helpset file and map file in the JAR file along with the topic files and navigational files. For more information on packaging helpsets, see JAR Files.

See also:

The Helpset File
Map File
JAR Files
Table of Contents File
Index File
Glossary File
Creating the Full−Text Search Database

4.4 Helpset File

When the JavaHelp system is activated by your application, the first thing it does is read the helpset file specified by the application. The helpset file defines the helpset for that application. A helpset is the set of data that comprises your help system. The helpset file includes the following information:

Map file	The map file is used to associate topic IDs with the URL or path name of HTML topic files.

View information
: Information that describes the navigators being used in the helpset. The standard navigators are: table of contents, index, and full−text search. Additional navigators are the glossary and the favorites navigators. Information about custom navigators is included here as well.

Helpset title
: The `<title>` defined in the helpset (`.hs`) file. This title appears at the top of the main window and any secondary windows defined in your helpset file.

Home ID
: The name of the (default) ID that is displayed when the help viewer is called without specifying an ID.

Presentation
: The windows in which to display the help topics. This is a new feature of the JavaHelp 2 software that is described in more detail below under `<presentation>`.

Sub−helpsets
: This optional section statically includes other helpsets by using the `<subhelpset>` tag. The helpsets indicated by this tag are merged automatically into the helpset that contains the tag. More details about merging can be found in Merging Helpsets.

Implementation
: This optional section creates a registry that provides key data mapping to define the HelpBroker class to use in the `HelpSet.createHelpBroker` method. The registry also determines the content viewer to user for a given MIME type. See `<impl>` below.

After your product is installed on your user's system, your Java program must be able to find the helpset file. The application specifies the path to the helpset file when it starts the JavaHelp system. By convention, the name of the helpset file ends with the `.hs` extension.

4.4.1 Helpset File Format

The format of the helpset file is based on the World Wide Web Consortium Extended Markup Language (XML 1.0) proposed recommendation:

```
http://www.w3.org/TR/PR−xml−971208
```

The following is an example of a helpset file (description follows):

```
<?xml version='1.0' encoding='ISO−8859−1' ?>
<!DOCTYPE helpset
  PUBLIC "−//Sun Microsystems Inc.//DTD JavaHelp HelpSet Version 2.0//EN"
         "http://java.sun.com/products/javahelp/helpset_2_0.dtd">
<helpset version="2.0">
   <!−− title −−>
   <title>Java Development Environment − Help</title>

   <!−− maps −−>
   <maps>
     <homeID>top </homeID>
     <mapref location="Map.jhm" />
   </maps>

   <!−− views −−>
   <view xml:lang="en" mergetype="javax.help.UniteAppendMerge">
      <name>TOC</name>
      <label>Table Of Contents</label>
      <type>javax.help.TOCView</type>
      <data>IdeHelpTOC.xml</data>
   </view>

   <view xml:lang="en" mergetype="javax.help.SortMerge">
      <name>Index</name>
      <label>Index</label>
      <type>javax.help.IndexView</type>
      <data>IdeHelpIndex.xml</data>
   </view>
   <view xml:lang="en">
      <name>Search</name>
      <label>Search</label>
      <type>javax.help.SearchView</type>
         <data engine="com.sun.java.help.search.DefaultSearchEngine">
         JavaHelpSearch
         </data>
   </view>

   <!-- A glossary navigator -->
   <view  mergetype="javax.help.SortMerge">
      <name>glossary</name>
      <label>Glossary</label>
      <type>javax.help.GlossaryView</type>
      <data>glossary.xml</data>
   </view>

   <!-- A favorites navigator -->
   <view>
      <name>favorites</name>
      <label>Favorites</label>
      <type>javax.help.FavoritesView</type>
   </view>

   <!-- presentation windows -->

   <!-- This window is the default one for the helpset.
     *  Its title bar says "Project X Help". It
     *  is a tri-paned window because displayviews, not
     *  defined, defaults to true and because a toolbar is defined.
     *  The toolbar has a back arrow, a forward arrow, and
     *  a home button that has a user-defined image.
   -->
   <presentation default=true>
       <name>main window</name>
       <size width="400" height="400" />
       <location x="200" y="200" />
       <title>Project X Help</title>
       <toolbar>
           <helpaction>javax.help.BackAction</helpaction>
           <helpaction>javax.help.ForwardAction</helpaction>
           <helpaction image="homeicon">javax.help.HomeAction</helpaction>
       </toolbar>
   </presentation>

   <!-- This window is simpler than the main window.
     *  It's intended to be used a secondary window.
     *  It has no navigation pane or toolbar.
   -->
   <presentation displayviews=false>
       <name>secondary window</name>
       <size width="200" height="200" />
       <location x="200" y="200" />
   </presentation>

   <!-- subhelpsets -->
   <subhelpset location="file:/c:/Foobar/HelpSet2.hs" />

   <!-- implementation section -->
   <impl>
      <helpsetregistry helpbrokerclass="javax.help.DefaultHelpBroker" />
      <viewerregistry viewertype="text/html"
         viewerclass="com.sun.java.help.impl.CustomKit />
      <viewerregistry viewertype="text/xml"
         viewerclass="com.sun.java.help.impl.CustomXMLKit />
   </impl>
</helpset>
```

4.4.1.1 The Helpset Tags

The following table describes the helpset tags:

<helpset>		Defines the helpset. It can contain all the following tags.
•	<title>	Names the helpset. This string can be accessed by the application and used in the presentation (for example, in the viewer header stripe).
•	<maps>	Specifies the default topic and URL of the map file used in the helpset. **<maps> Tags** • <homeID> Specifies the name of the (default) ID that is displayed when the help viewer is called if an ID is not explicitly specified. • <mapref> Specifies the URL of the map file relative to the helpset.
•	<view>	Defines the navigators used in the helpset. Each navigator is defined by its own <view>. **<view> Attributes** • xml:lang="*lang*" Language for the view. Use the standard locale−country−variant format. Some examples: *xml:lang="de"* *xml:lang="en"* *xml:lang="en−US"* • mergetype="*class*" Specifies the path to the merge class. The merge classes are: *javax.help.UniteAppendMerge* *javax.help.SortMerge* *javax.help.AppendMerge* *javax.help.NoMerge* For more information, see Merging Helpsets. **<view> Tags** • <name> Names the view. • <label>

		Specifies a label associated with the view. This string is displayed in the navigator's tab if the presentation's `displayviewimages` attribute is `"false"`. • `<type>` Specifies the path to the navigator class. • `<data>` Specifies the path to the data used by the navigator. When used with the search navigator, this tag contains the following attribute: `engine` A `String` indicating the path to the search engine class. • `<image>` Specifies the image displayed for the navigator in the tab bar at the top of the navigator pane. The argument to this attribute is an ID defined in the map file. The ID must be associated with a GIF or JPEG file. If this attribute is not specified, the default navigator image is displayed.
•	`<presentation>`	Defines the windows used in the helpset. Each window is defined by its own `<presentation>` tag. See the helpset example above for sample code for this tag and its attributes. **`<presentation>` Attributes** • `xml:lang="`*lang*`"` Language for the view. Use the standard locale−country−variant format. Some examples: *`xml:lang="de"`* *`xml:lang="en"`* *`xml:lang="en-US"`* • `default="true|false"` This presentation is the default one for this helpset. The default value of this attribute is `"false"`. (In other words, if you do not specify this attribute, the presentation is not the default one.) If more than one presentation is specified as `default`, the last one specified is the default presentation. • `displayviews="true|false"` Show the navigational views of this helpset in a pane on the left side of the window (like the tri−paned viewer). The default value is `"true"`.

- `displayviewimages="true|false"`

  The default value "`true`" displays the image for each navigator in the navigator's tab. If set to "`false`", the text defined in each view's `<label>` tag is displayed instead.

`<presentation>` Tags

- `<name>`

  Names the window, allowing it to be called by name from the Java program or from the TOC. If you define a presentation, be sure to tell the Java programmer its name and when you expect it to be called from the program.

- `<size>`

  Specifies the size of the window with the following attributes:

`width="nnn"`	Width in pixels.
`height="nnn"`	Height in pixels.

- `<location>`

  Specifies the position of the window with the following attributes:

`x="nnn"`	Horizontal (x) coordinate.
`y="nnn"`	Vertical (y) coordinate.

- `<title>`

  Specifies the text that appears in the title bar at the top of the window.

- `<image>`

  Specifies the image displayed on the left side of the help window's title bar. The argument to this attribute is an ID defined in the map file. The ID must be associated with a GIF or JPEG file. If this attribute is not specified, the default JavaHelp image is displayed.

- `<toolbar>`

  Indicates that the window is to have a toolbar.

  You define buttons on the toolbar by using a `<helpaction>` tag for each button and a Java class name for the action.

  ```
  <helpaction>
  javax.help.HelpAction
  </helpaction>
  ```

		HelpAction is any of the following default class names, each of which defines an action and a button image:
		`BackAction`
		Goes to previous topic.
		`FavoritesAction`
		Adds current map ID (currently displayed topic) to Favorites navigator.
		`ForwardAction`
		Goes to next topic.
		`HomeAction`
		Loads the home topic in the content pane
		`PrintAction`
		Prints the topics selected in the navigator.
		`PrintSetupAction`
		Runs print setup before printing.
		`ReloadAction`
		Reloads the current topic page in the content viewer.
		`SeparatorAction`
		Creates a separator between action buttons.
		Each action button has a default button image. If you want to use a different image, you must define a map ID for the image file in the map file and reference the map ID inside the `<helpaction>` tag by using the `image` attribute.
		`<helpaction>` Attribute
		`image="`*mapID*`"`
		For example:
		`<helpaction`
		`image="images_backbutton">`
		`javax.help.BackAction`
		`</helpaction>`

•	`<subhelpset>`	This optional tag can be used to specify helpsets you want merged with the helpset that contains the tag. See Merging Helpsets for more information. Contains the following attribute: `location` URL of the helpset file to be merged.
•	`<impl>`	The implementation section creates a registry that provides key data mapping to define the HelpBroker class to use in the `HelpSet.createHelpBroker` method. The registry also determines the content viewer to user for a given MIME type. **`<impl>` Tags** • `<helpsetregistry>` Registers the default `HelpBroker` class. Uses the following attribute: `helpbrokerclass="class"` – Required. Name of a class that implements `HelpBroker`. • `<viewregistry>`

		Registers a viewer class for a given MIME type. Uses the following attributes: `viewertype="mine/type"` − Required. MIME type. `viewerclass="class"` − Required. Class name.

See also:

Map File
Table of Contents File
Index File
Glossary File
Favorites File
Creating the Full−Text Search Database

4.5 The Map File

When the JavaHelp system is activated by your application, the first thing it does is read the application's helpset file. The next thing it does is read the map file listed in the helpset file. The map file is used to associate topic IDs with URLs (paths to HTML topic files) and to define the window that can display help topics. By convention, map file names use the `.jhm` suffix.

The format of the map file is based on the World Wide Web Consortium Extended Markup Language (XML).

The following code listing is an example of a short map file:

```
<?xml version='1.0' encoding='ISO−8859−1' ?>
<!DOCTYPE map
    PUBLIC "−//Sun Microsystems Inc.//DTD JavaHelp Map Version 2.0//EN"
    "http://java.sun.com/products/javahelp/map_2_0.dtd">
<map version="2.0">
<mapID target="toplevelfolder" url="images/toplevel.gif" >
      <mapID target="hol_intro" url="hol/hol.html" />
      <mapID target="halloween" url="hol/hall.html"/>
      <mapID target="jackolantern" url="hol/jacko.html" />
      <mapID target="jacko_carving" url="hol/jacko.html#carving" />
      <mapID target="mluther" url="hol/luther.html" />
      <mapID target="reformation" url="hol/inforefo.html" />
      <mapID target="fawkes" url="hol/guy.html" />
      <mapID target="thanksgiving" url="hol/thanks.html" />
      <mapID target="thanksgiving_turkey" url="hol/thanks.html#turkey" />
      <mapID target="1thanksproc" url="hol/thanks2.html" />
      <mapID target="gwthanksproc" url="hol/thanks3.html" />
      <mapID target="althanksproc" url="hol/thanks4.html" />
      <mapID target="valentine" url="hol/val.html" />
      <mapID target="onlove" url="hol/love.html" />
</mapID>
</map>
```

Note that images referred to as IDs (for example, in the TOC) can also be associated with an ID. In this example, `toplevelfolder` is associated with the GIF image `toplevel.gif`.

4.5.0.1 The Map Tags

The following table describes the tags that can be used in the map file:

`<map>` Defines the map. It contains `<mapID>` tags and the following optional attributes.

- `xml:lang="lang"` Language for the map file. Use the standard locale−country−variant format.

  Some examples:

  ```
  xml:lang="de"
  xml:lang="en"
  xml:lang="en−US"
  ```

- `version="1.0"|"2.0"` Version of JavaHelp software.

`<mapID>` Defines a map entry. Uses the following attributes:

- `xml:lang="lang"` Language for the map ID. Use the standard locale−country−variant format.

  Some examples:

  ```
  xml:lang="de"
  xml:lang="en"
  xml:lang="en−US"
  ```

- `target` Specifies the ID to associate with the URL specified in the `url` attribute.
- `url` Specifies the URL to associate with the ID specified in the `target` attribute.

See also:

The Helpset File
JAR Files
Table of Contents File
Index File
Creating the Full−Text Search Database

4.6 JAR Files

This topic describes how JAR files are used in the JavaHelp system.

- Using JAR files
- The `jar` command
- Creating JAR files
- Listing JAR files
- Extracting files from JAR files
- The JAR protocol

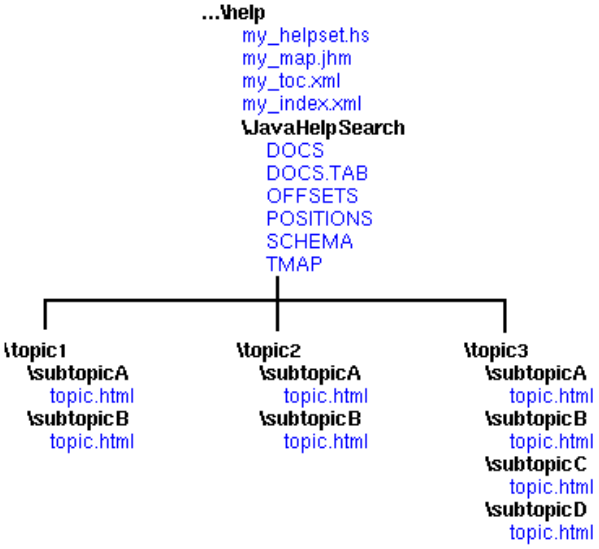
4.6.1 Using JAR Files

After you create your help information, you will usually encapsulate it into a single file and compress it for delivery to your users. The JavaHelp system uses the JAR (Java ARchive) format for encapsulation and compression. The JAR file format is based on the popular ZIP file format. The JavaHelp system automatically extracts information from the JAR file when it is required.

Until support is available from GUI−base help authoring tools, the `jar` command (located in the J2SE bin folder) must be used from a command−line prompt to create, read, and extract data from JAR files.

4.6.2 Sample Help Hierarchy

The following sections refer to this sample help hierarchy:



4.6.3 The `jar` Command

The `jar` command syntax is:

```
jar [ctxvfm] [jar−file] [manifest−file] files ...
Option flags are:
    c  create new archive
    t  list table of contents for archive
    x  extract named (or all) files from archive
    v  generate verbose output on standard error
    f  specify JAR file name
    m  include manifest information from specified
       manifest file
```

For more detailed information about the `jar` command or format, please refer to `http://java.sun.com/beans/jar.html`.

The `jar` command is located in the `bin` directory of the J2SE.

4.6.4 Creating JAR Files

To create a JAR file from your help files, make the top level help folder the current folder. The `jar` command descends recursively through the different directories and copies all of the files to the JAR file.

Use the following steps to create a JAR file named `my_help.jar` from the hierarchy example above:

1. `C:\>` **`cd ...\help`** (where "..." is the path above the `\help` folder)
2. `C:...\help>` **`jar −cvf my_help.jar *`**

The `jar −cvf` command copies all the files in the `\help` folder and in all folders hierarchically beneath it into a JAR file named `my_help.jar`. As the command creates the JAR file, it reports its progress with output like the following:

```
    adding: my_helpset.hs (in=5757) (out=2216) (deflated 61%)
```

This indicates that the file `my_helpset.hs` was added to the JAR file and compressed 61% (from 5272 bytes to 2150 bytes).

When you create a JAR file, the `jar` command automatically creates a manifest file for you. The manifest file consists of a list of files present within the archive itself.

4.6.5 Listing JAR Files

Use the `t` option to list the files included in a JAR file:

```
C:\> jar -tvf my_help.jar
  5272 Fri Apr 03 14:48:04 PST 1998 META-INF/MANIFEST.MF
  5757 Fri Apr 03 12:21:04 PST 1998 my_helpset.hs
  1345 Wed Feb 18 14:40:16 PST 1998 my_map.jhm
  1478 Wed Feb 18 14:40:16 PST 1998 my_toc.xml
  4678 Thu Mar 12 07:28:54 PST 1998 my_index.xml
  2345 Thu Mar 12 07:28:32 PST 1998 JavaHelpSearch/DOCS
  3456 Thu Mar 19 11:26:56 PST 1998 JavaHelpSearch/DOCS.TAB
  1457 Fri Mar 13 13:30:06 PST 1998 JavaHelpSearch/OFFSETS
  1465 Thu Mar 19 11:26:56 PST 1998 JavaHelpSearch/POSITIONS
  1234 Thu Mar 19 11:26:56 PST 1998 JavaHelpSearch/SCHEMA
  3214 Thu Mar 19 11:26:56 PST 1998 JavaHelpSearch/TMAP
  3113 Thu Mar 12 07:28:36 PST 1998 topics/topic1/subtopicA/topic.html
   230 Thu Mar 19 11:26:56 PST 1998 topics/topic1/subtopicB/topic.html
  1661 Wed Feb 18 14:40:46 PST 1998 topics/topic2/subtopicA/topic.html
  3181 Wed Feb 18 14:40:46 PST 1998 topics/topic2/subtopicB/topic.html
  1667 Thu Mar 19 11:26:56 PST 1998 topics/topic3/subtopicA/topic.html
  9072 Thu Mar 12 07:28:36 PST 1998 topics/topic3/subtopicB/topic.html
  3673 Thu Mar 19 11:26:56 PST 1998 topics/topic3/subtopicC/topic.html
   551 Fri Mar 13 13:30:12 PST 1998 topics/topic3/subtopicD/topic.html
```

4.6.6 Extracting Files from JAR Files

Use the x option to extract files from the JAR file:

```
C:\> jar -xvf my_help.jar
 extracted: META-INF/MANIFEST.MF
 extracted: my_helpset.hs
 extracted: my_map.jhm
 extracted: my_toc.xml
 extracted: my_index.xml
 extracted: JavaHelpSearch/DOCS
 extracted: JavaHelpSearch/DOCS.TAB
 extracted: JavaHelpSearch/OFFSETS
 extracted: JavaHelpSearch/POSITIONS
 extracted: JavaHelpSearch/SCHEMA
 extracted: JavaHelpSearch/TMAP
 extracted: topics/topic1/subtopicA/topic.html
 extracted: topics/topic1/subtopicB/topic.html
 extracted: topics/topic2/subtopicA/topic.html
 extracted: topics/topic2/subtopicB/topic.html
 extracted: topics/topic3/subtopicA/topic.html
 extracted: topics/topic3/subtopicB/topic.html
 extracted: topics/topic3/subtopicC/topic.html
 extracted: topics/topic3/subtopicD/topic.html
```

Note that it is not necessary to extract files from the JAR file to use them with the JavaHelp system. The JavaHelp system reads files directly from the JAR file as they are required.

4.6.7 The `JAR:` Protocol

The Java^{TM} 2 SDK implements a protocol for referring explicitly to files within JAR files. The syntax of the `jar:` protocol is:

```
    jar:<url>!/{entry}
```

The `jar:` protocol can be used to refer to entries within JAR files, the entire JAR file, or a directory as base URLs (JAR directory).

Examples:

An entry within a JAR file:

```
jar:http://www.foo.com/bar/baz.jar!/COM/foo/Quux.class
```

A JAR file:

```
jar:file://www.foo.com/bar/baz.jar!/
```

A JAR directory:

```
jar:file://www.foo.com/bar/baz.jar!/COM/foo/
```

"`!/`" is called the *separator*.

For more information, refer to the Java^{TM} 2 SDK documentation.

See also:

- The Helpset File
- Map File
- Table of Contents File
- Index File
- Creating the Full−Text Search Database

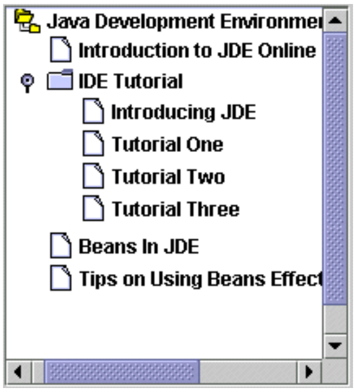
4.7 Table of Contents File

The table of contents (TOC) file describes for the TOC navigator the content and layout of the TOC. The format of the TOC file is based on the World Wide Web Consortium (W3C) Extended Markup Language (XML). Following is a very small example of a TOC file:

```
<?xml version='1.0' encoding='ISO−8859−1' ?>
<!DOCTYPE toc
  PUBLIC
  "−//Sun Microsystems Inc.//DTD
   JavaHelp TOC Version 2.0//EN"
  "http://java.sun.com/products/javahelp/toc_2_0.dtd">
<toc version="2.0">
  <tocitem image="toplevelfolder"
           text="Java Development Environment">
     <tocitem target="jde.intro">Introduction to JDE Online Help />
     <tocitem text="IDE Tutorial" target="tut.starttoc">
        <tocitem text="Introducing JDE" target="tut.intro" />
        <tocitem text="Tutorial One" target="tut.quickstart" / >
        <tocitem text="Tutorial Two" target="tut.edit" />
        <tocitem text="Tutorial Three" target="tut.errors" />
     </tocitem>
     <tocitem text="Beans in JDE" target="bean.jbeanstory" />
     <tocitem text="Tips on Using Beans Effectively"
```

```
              target="bean.beantips"
              mergetype="javax.help.SortMerge"
              presentationtype="javax.help.SecondaryWindow
              presentationname="mainsw" />
  </tocitem>
</toc>
```

This example produces the following TOC display:



Java Development Environmei
Introduction to JDE Online
IDE Tutorial
Introducing JDE
Tutorial One
Tutorial Two
Tutorial Three
Beans In JDE
Tips on Using Beans Effect

4.7.0.1 The TOC Tags

The following table describes the TOC tags:

<table>
<tr>
<td><code>&lt;toc&gt;</code></td>
<td>Defines the TOC. This tag contains <code>&lt;tocitem&gt;</code> tags and the following optional attributes.
<table>
<tr><td><code>xml:lang="lang"</code></td><td>Language for the TOC. Use the standard locale−country−variant format.

<i>Some examples:</i>
<code>xml:lang="de"</code>
<code>xml:lang="en"</code>
<code>xml:lang="en−US"</code></td></tr>
<tr><td><code>version="1.0"|"2.0"</code></td><td>Version of JavaHelp software.</td></tr>
<tr><td><code>categoryclosedimage</code></td><td>(<i>optional</i>) Specifies the image displayed to the left of a closed TOC folder. A TOC folder is any entry in the TOC that has subnodes. The argument to this attribute is an ID defined (associated with a GIF or JPEG image) in the map file. If this attribute is not specified, the default folder image is displayed.

You can override this setting for any single TOC folder by setting the folder's <code>image</code> attribute (see <code>&lt;tocitem&gt;</code> below).</td></tr>
<tr><td><code>categoryopenimage</code></td><td>(<i>optional</i>) Specifies the image displayed to the left of an open TOC folder. A TOC folder is any entry in the TOC that has subnodes. The argument to this attribute is an ID defined (associated with a GIF or JPEG image) in the map file. If this attribute is not specified, the default folder image is displayed.</td></tr>
</table>
</td>
</tr>
</table>

		You can override this setting for any single TOC folder by setting the folder's `image` attribute (see `<tocitem>` below).
	`topicimage`	(*optional*) Specifies the image displayed to the left of a TOC topic. A TOC topic is an entry in the TOC that has no subnodes and links to a help topic. The argument to this attribute is an ID defined (associated with a GIF or JPEG image) in the map file. If this attribute is not specified, the default topic image is displayed. You can override this setting for any single TOC topic by setting the topic's `image` attribute (see `<tocitem>` below).
`<tocitem>`	Defines a TOC entry. Nesting *entry1* in *entry2* defines *entry2* to be hierarchically contained within *entry1*. Uses the following attributes:	

	`xml:lang="lang"`	Language for the TOC item. Use the standard locale−country−variant format.
		Some examples:
		`xml:lang="de"`
		`xml:lang="en"`
		`xml:lang="en−US"`

	`text`	Specifies the text that displays in the TOC.
	`target`	(*optional*) Specifies the ID to display when the entry is chosen by the user. IDs are defined (associated with a URL) in the map file. If this attribute is not used, the entry does not link to a topic
	`image`	(*optional*) Specifies the image displayed to the left of a TOC item. The argument to this attribute is an ID defined (associated with a GIF or JPEG image) in the map file. If this attribute is not specified, the default open folder, closed folder, or topic image is displayed. This image overrides the global image, if any, specified for the `<toc>` tag's `categoryclosedimage`, `categoryopenimage`, or `topicimage` attributes (see `<toc>` above).

	`mergetype="`*class*`"`	(*optional*) Path to a valid merge class for the view. The default merge class for a TOC is `javax.help.AppendMerge`.
		The merge classes are:
		`javax.help.UniteAppendMerge`
		`javax.help.SortMerge`
		`javax.help.AppendMerge`
		`javax.help.NoMerge`
		For more information, see Merging Helpsets.

	`expand="true|false"`	(*optional*) Specifies whether to expand the TOC item and its subitems when the TOC initially opens. The default setting is `"none"`, a setting that expands only the top level items.
	`presentationtype`	(*optional*) Specifies the type of window in which the topic will be displayed (defined in the `<presentation>` section of the `.hs` file). For more information, see the presentation feature in Helpset File.
	`presentationname`	(*optional*) Specifies the name of the window in which the topic will be displayed.

See also:

The Helpset File
JAR Files
Map File
Index File
Glossary File
Favorites File
Creating the Full−Text Search Database

4.8 Index File

The index file describes to the index navigator the content and layout of the index. The format of the index file is based on the World Wide Web Consortium Extended Markup Language (XML). The following is a very small example of an index file:

```
<?xml version='1.0' encoding='ISO−8859−1' ?>
   <!DOCTYPE index
     PUBLIC
     "−//Sun Microsystems Inc.//DTD
      JavaHelp Index Version 2.0//EN"
     "http://java.sun.com/products/javahelp/index_2_0.dtd">

   <index version="2.0">
      <indexitem text=".prof extension (profile data)"
                 target="prof.profile" />
      <indexitem text="accelerators (keyboard), see 'keyboard commands'" />
      <indexitem text="adding existing portfolio" target="proj.import" />
      <indexitem text="adding existing project">
         <indexitem text="naming project" target="proj.importdirectory" />
         <indexitem text="naming storage directory"
                    target="proj.importdirectory" />
         <indexitem text="procedures for" target="proj.importproject2" />
      </indexitem>
      <indexitem text="analyzing program performance, see 'profiler'" />
      <indexitem text="Java Applets">
         <indexitem text="overview" target="applet_over"
                    presentationtype="javax.help.SecondaryWindow
                    presentationname="mainsw">
         <indexitem text="editing in content page"
                    target="applet_editing">
         <indexitem text="inserting in content page"
                    target="applet_insert">
      </indexitem>
   </index>
```

4.8.0.1 The Index Tags

The following table describes the index tags:

`<index>`	Defines the index. It can contains `<indexitem>` tags and the following optional attributes.	

	`xml:lang="lang"`	Language for the index. Use the standard locale−country−variant format.
		Some examples:
		`xml:lang="de"`
		`xml:lang="en"`
		`xml:lang="en−US"`

	`version="1.0"|"2.0"`	Version of JavaHelp software.
	.	
`<indexitem>`	Defines an index entry. Nesting *entry1* within *entry2* defines *entry2* to be hierarchically contained within *entry1*. Uses the following attributes:	

	`xml:lang="lang"`	Language for the index item. Use the standard locale−country−variant format.
		Some examples:
		`xml:lang="de"`
		`xml:lang="en"`
		`xml:lang="en−US"`

	`text="`*string*`"`	Specifies the text that displays in the index.
	`target="`*string*`"`	(*optional*) Specifies the map ID of the topic that displays when the entry is chosen by the user. IDs are defined (associated with a URL) in the map file. If this attribute is not used, the index entry does not link to a topic (probably because it's being used to group sub−entries).

	`mergetype="`*class"*	(*optional*) The path to a valid merge class for the view. The default merge type for an index is `javax.help.AppendMerge`.
		The merge classes are:
		`javax.help.UniteAppendMerge`
		`javax.help.SortMerge`
		`javax.help.AppendMerge`
		`javax.help.NoMerge`
		For more information, see Merging Helpsets.

	`expand="true|false"`	(*optional*) Specifies whether to expand the index item and its subitems when the index initially opens. The default setting is `"none"`, a setting that expands only the top level items.
	`presentationtype`	(*optional*) Specifies the type of window in which the topic will be displayed (defined in the `<presentation>` section of the `.hs` file). For more information, see the presentation

	feature in Helpset File.
`presentationname`	(*optional*) Specifies the name of the window in which the topic will be displayed.

See also:

The Helpset File
Map File
JAR Files
Table of Contents File
Glossary File
Favorites File
Creating the Full−Text Search Database

4.9 Glossary Navigator and File

The glossary file describes for the glossary navigator the content and layout of the glossary. The format of a glossary file is the same as that of an index file. The glossary file format, as with the index file format, is based on the World Wide Web Consortium Extended Markup Language (XML).

The following XML code shows a sample definition of a glossary navigator in the helpset file:

```
   <view xml:lang="en" mergetype="javax.help.SortMerge">
      <name>glossary</name>
      <label>Glossary</label>
      <type>javax.help.GlossaryView</type>
      <data>glossary.xml</data>
  </view>
```

The following XML code shows a small example of a glossary file:

```
<?xml version='1.0' encoding='ISO−8859−1'  ?>
   <!DOCTYPE index
     PUBLIC
     "−//Sun Microsystems Inc.//DTD
      JavaHelp Index Version 2.0//EN"
     "http://java.sun.com/products/javahelp/index_2_0.dtd">

   <index version="2.0">
      <indexitem text="applet" target="applet_def"/>
      <indexitem text="application" target="application_def"/>
      <indexitem text="application server"
                 target="appServer_def"/>
      <indexitem text="AWT" target="awt_def"/>
      <indexitem text="beans" target="bean_def"/>
   </index>
```

4.9.0.1 The Glossary Tags

The following table describes the glossary tags:

`<index>`	Defines the glossary. It can contains `<indexitem>` tags and the following optional attributes.	

	`xml:lang="lang"`	Language for the glossary. Use the standard locale−country−variant format.
		Some examples:
		`xml:lang="de"`
		`xml:lang="en"`
		`xml:lang="en-US"`

	`version="1.0"|"2.0"`	Version of JavaHelp software.
	.	
`<indexitem>`	Defines a glossary entry. Nesting *entry1* within *entry2* defines *entry2* to be hierarchically contained within *entry1*. Uses the following attributes:	

	`xml:lang="lang"`	Language for the glossary item. Use the standard locale−country−variant format.
		Some examples:
		`xml:lang="de"`
		`xml:lang="en"`
		`xml:lang="en-US"`

	`text="`*string*`"`	Specifies the text that displays in the glossary.
	`target="`*string*`"`	Specifies the map ID of the topic that displays when the entry is chosen by the user. IDs are defined (associated with a URL) in the map file.

	`mergetype="`*class"*	(*optional*) Specifies the path to a valid merge class for the glossary. The default merge class for a glossary is `javax.help.AppendMerge`.
		The merge classes are:
		`javax.help.UniteAppendMerge`
		`javax.help.SortMerge`
		`javax.help.AppendMerge`
		`javax.help.NoMerge`
		For more information, see Merging Helpsets.

	`expand="true|false"`	(*optional*) Specifies whether to expand the glossary item and its subitems when the glossary initially opens. The default setting is `"none"`, a setting that expands only the top level items.
	`presentationtype`	(*optional*) Specifies the type of window in which the topic will be displayed (defined in the `<presentation>` section of the `.hs` file). For more information, see the presentation feature in Helpset File.
	`presentationname`	(*optional*) Specifies the name of the window in which the topic will be displayed.

See also:

The Helpset File
Map File

JAR Files
Table of Contents File
Index File
Favorites File
Creating the Full−Text Search Database

4.10 Favorites Navigator and File

Favorites are links to helpset topics that the user wants to save and reuse. The favorites file describes the content and layout of these links in the favorites navigator. Unlike other navigational views that store the view's metadata in the helpset directories, favorites are stored in the user's directory in the file *userdir*/`.JavaHelp/Favorites.xml`.

The favorites file format is based on the World Wide Web Consortium Extended Markup Language (XML). The DTD for this syntax is `dtd/favorites_2_0.dtd`. The top level tag is `<favorites>`. A version attribute is optional. When present, its value must be "2.0".

The JavaHelp system specifies one favorites navigator view: `javax.help.FavoritesView`. Favorites do not require a data definition as part of the navigator view definition. Also, the `mergetype` tag is ignored.

The following XML code shows a sample definition of a favorites navigator in the view section of a helpset file:

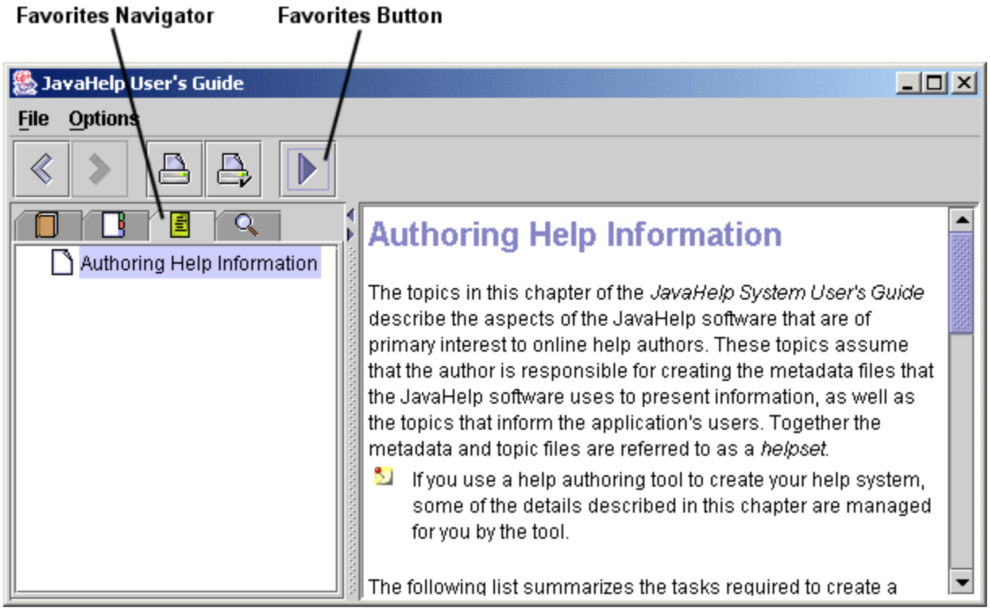
```
<view>
   <name>Favorites</name>
   <label>Favorites</label>
   <type>javax.help.FavoritesView</type>
</view>
```

The following figure shows how the Favorites navigator looks in the helpset for the JavaHelp System Users Guide. As the figure shows, when you add a Favorites navigator to a helpset, the JavaHelp system also adds a Favorites button to the default toolbar. Clicking the Favorites button enables the user to add the currently displayed help topic to the list of favorites in the navigator.

If you have defined your own windows (presentations) in the helpset file, the favorites button is not added automatically. To display a favorites button on the presentation's toolbar, you must add the following help action to the `<presentation>` tag's `<toolbar>` tag:

```
<helpaction>javax.help.FavoritesAction</helpaction>
```

For more information, see the toolbar tag description in the Helpset File section.



Authoring Help Information

The topics in this chapter of the *JavaHelp System User's Guide* describe the aspects of the JavaHelp software that are of primary interest to online help authors. These topics assume that the author is responsible for creating the metadata files that the JavaHelp software uses to present information, as well as the topics that inform the application's users. Together the metadata and topic files are referred to as a *helpset*.

If you use a help authoring tool to create your help system, some of the details described in this chapter are managed for you by the tool.

The following list summarizes the tasks required to create a

The following XML code is an example of a favorites file, which is created for you by the JavaHelp system:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE favorites
    PUBLIC "-//Sun Microsystems Inc.//DTD JavaHelp Favorites Version 2.0//EN"
    "http://java.sun.com/products/javahelp/favorites_2_0.dtd">
<favorites version="2.0">
  <favoriteitem text="Love Holidays" >
     <favoriteitem text="On Love" target="onlove" hstitle="History of the Holidays"/>
     <favoriteitem text="Valentines" target="valentine" hstitle="History of the Holidays"/>
  </favoriteitem>
  <favoriteitem text="Numbers" >
      <favoriteitem text="Zero" target="0" hstitle="Master"/>
      <favoriteitem text="Zero - note "
                 url="file:/usr/test/hs/Zeronote.html"
                 hstitle="Master"/>
  </favoriteitem>
```

4.10.0.1 The Favorites Tags

The following table describes the favorites tags:

`<favorites>`	Defines the user favorites list. It contains `<favoriteitem>` tags and the following optional attributes.
	`xml:lang="lang"` Language for the favorites list. Use the standard locale−country−variant format.
	Some examples:
	`xml:lang="de"`
	`xml:lang="en"`
	`xml:lang="en-US"`
	`version="2.0"` Version of JavaHelp software.
	.

`<favoriteitem>`	Defines a favorites link. You can nest favorites links to create hierarchies of favorites. Uses the following attributes:	

	`xml:lang="lang"`	*(optional)* Language for the favorites link. Use the standard locale−country−variant format.
		Some examples:
		`xml:lang="de"`
		`xml:lang="en"`
		`xml:lang="en−US"`

	`text="`*string*`"`	Specifies the text that displays in the list.
	`target="`*string*`"`	(*optional*) Specifies the map ID of the topic that displays when the link is chosen by the user. IDs are defined (associated with a URL) in the map file.
	`url="`*string*`"`	(*optional*) Specifies the URL of the topic that displays when the link is chosen by the user. IDs are defined (associated with a URL) in the map file.
	`hstitle="`*string*`"`	Specifies the title of the helpset.
	`image="`*string*`"`	(*optional*) Specifies the image displayed before a favorites link. The argument to this attribute is a mapID defined (associated with a `GIF` or `JPEG` image) in the map file. If this attribute is not specified, the default images are displayed.
	`presentationtype`	(*optional*) Specifies the type of window in which the favorite topic will be displayed (defined in the `<presentation>` section of the `Hs` file). For more information, see the presentation feature in Helpset File.
	`presentationname`	(*optional*) Specifies the name of the window in which the favorites topic will be displayed.

See also:

The Helpset File
Map File
JAR Files
Table of Contents File
Index File
Glossary File
Creating the Full−Text Search Database

4.11 Context−Sensitive Help

The JavaHelp system provides a number of features that enable you to provide context−sensitive help to your users. Context−sensitive help is information provided to users based on the context of the task in which they are involved.

Implementing context−sensitive help involves associating help topics with objects in the application's graphical user interface (GUI) such as menu items, buttons, text boxes, and windows. Help authors generally work with developers to determine which topics are assigned to each object. The developer assigns map IDs to the application's GUI objects, then the help author associates those IDs with topic URLs in the map file. The fact that hard URL addresses are not embedded in the application code allows the author to change topics without requiring the developer to change the application.

For details about how context−sensitive help is implemented for an application, see Implementing Context−Sensitive Help.

4.11.1 Types of Context−Sensitive Help

The JavaHelp system provides mechanisms for two types of context−sensitive help: *user−initiated help* and *system−initiated help*.

4.11.2 User−Initiated Help

User initiated help delivers information to users when they explicitly ask for it. The JavaHelp system includes the following user−initiated mechanisms:

- Window−Level Help
- Field−Level Help
- Help Menu
- Help Button

4.11.2.1 Window−Level Help

Users can obtain help about container objects such as application windows and dialog boxes that have focus by pressing the F1 function key (on systems with a Help key, the Help key also works). An object is considered to have focus when it is in a state that allows the user to interact with it. By default, help information is displayed in the help viewer.

4.11.2.2 Field−Level Help

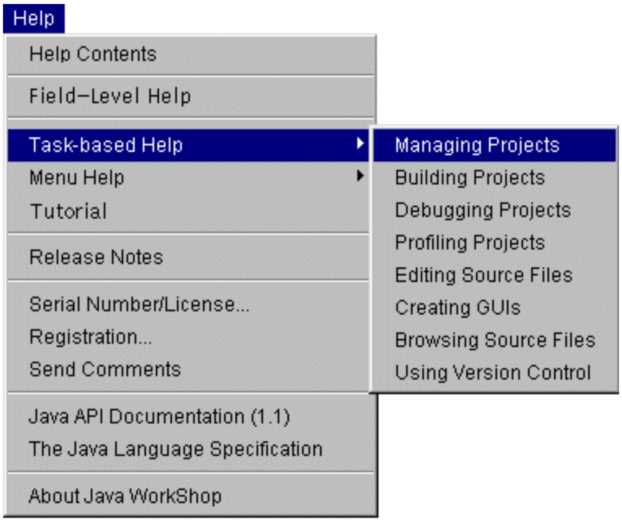
Users can use field−level help to obtain help about any GUI object. To use field−level help, the user:

1. Clicks the field−level help button ↖? or chooses the Help > Field−Level Help menu item to change the cursor to the special field−level help cursor (↖?)
2. Selects a GUI object

By default, help information is displayed in the help viewer and the cursor returns to its original state.

4.11.2.3 Help Menu

The Help menu can be used to provide help to users about specific tasks or objects. The following Help menu contains a submenu of items that provide help about completing various tasks.



4.11.2.4 Help Button

It is common for dialog boxes to contain a Help button that provides help information about how to use the dialog box. Clicking the Help button is usually equivalent to pressing the F1 key while the dialog box has focus.

4.11.3 System−Initiated Help

Most applications provide information automatically when the user performs a particular action. Most commonly, this information consists of status, warning, or error messages. It is also possible for the application to use the help viewer to provide more detailed help based on user actions.

See also:

Implementing Context−Sensitive Help

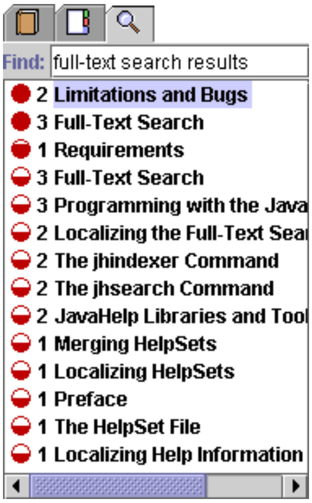
4.12 Full−Text Search

The JavaHelp system full−text search engine uses a natural language search technology that not only retrieves documents, but locates specific passages within those documents where answers to queries are likely to be found. The technology involves a conceptual indexing engine that analyzes documents to produce an index of their content and a query engine that uses this index to find relevant passages in the material.

As the help author, you create the search database that is searched by the JavaHelp system full−text search engine. The process of creating the search database is described in Creating the Full−Text Search Database.

4.12.1 How Searching Works

To initiate a search the user enters a natural language query in the search navigator Find text box. The results are reported back to the user in the following display:



Find: full-text search results

● 2 Limitations and Bugs
● 3 Full-Text Search
◕ 1 Requirements
◒ 3 Full-Text Search
◒ 3 Programming with the Java
◒ 2 Localizing the Full-Text Sear
◒ 2 The jhindexer Command
◒ 2 The jhsearch Command
◒ 2 JavaHelp Libraries and Tool
◒ 1 Merging HelpSets
◒ 1 Localizing HelpSets
◒ 1 Preface
◒ 1 The HelpSet File
◒ 1 Localizing Help Information

- The circle in the first column indicates the ranking of the matches for that topic. The more filled−in the circle is, the higher the ranking. There are five possible rankings (from highest to lowest): ● ◕ ◒ ◔ ○
- The number in the second column indicates the number of times the query was matched in the listed topic.
- The text specifies the name of the topic (as specified in the topic's `<TITLE>` tag) in which matches are found.

To avoid confusion, ensure that the `<TITLE>` tag corresponds to the title used in the table of contents.

4.12.1.1 Relaxation Ranking

The search engine uses a technique called *relaxation ranking* to identify and score specific passages of text that are likely to answer the user's query. The relaxation ranking algorithm compares the user's query terms with occurrences of the same or related terms in the help topics. The search engine attempts to find passages in the help topics in which as many as possible of the query terms occur in the same form and the same order. The search engine automatically relaxes these constraints to identify passages in which:

- Not all of the terms occur
- The terms occur in different forms
- The terms occur in a different order
- The terms occur with intervening words

The search engine assigns appropriate penalties (that lower the ranking) to the passages for these deviations from the specified query.

The ranking process improves as queries become more complex and include more information.

4.12.1.2 Morphing

The JavaHelp search engine uses "morphing" technology to find words with common roots. For example, when the term "build" is included in a search string, matches that contain "built", "builder", "building", and "builds" are returned.

See also:

Creating the Full−Text Search Database
The `jhindexer` Command

4.13 Creating the Full−Text Search Database

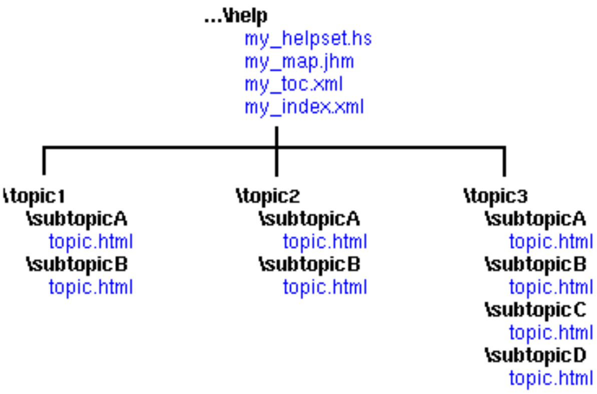
When a user initiates a full−text search, the JavaHelp system full−text search engine searches a special search database to find matches quickly. You use the `jhindexer` command to create the search database for your help topics.

The search database created by the `jhindexer` command consists of six files located in a folder named `JavaHelpSearch`. As described in Setting Up a JavaHelp System, the search database folder is usually located in the same folder as the rest of the help metadata files.

4.13.1 Example

The following example describes how to use the `jhindexer` command in its simplest form. For details about other features of the `jhindexer` command, see The `jhindexer` Command.

The example assumes that your help information is arranged in the following hierarchy:



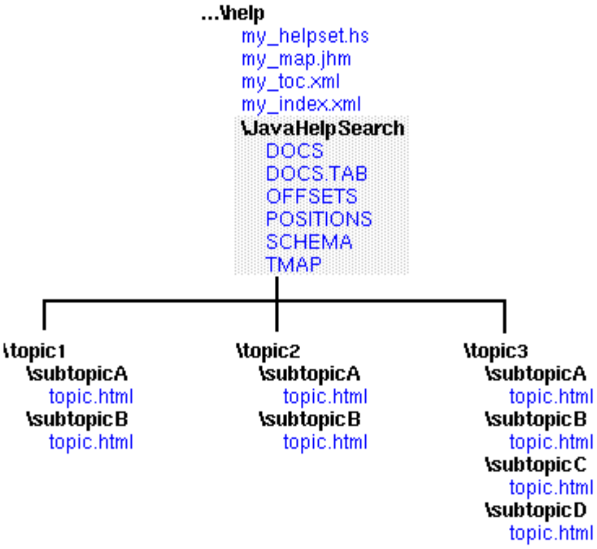
To create a full−text search database:

1. Make the `...\help` folder the current folder
2. Specify the top−level folders as arguments to the `jhindexer` command, as follows:

```
jhindexer topic1 topic2 topic3
```

The `jhindexer` command is located in the `javahelp\bin` folder of the JavaHelp system release.

The `jhindexer` command recursively descends the help hierarchy, indexing all the files it encounters. When finished, `jhindexer` places the search database files in a folder named `JavaHelpSearch` in the current folder:



▶ **To verify the validity of a full−text search database:**

1. Make the `...\help` folder the current folder
2. Specify the `JavaHelpSearch` folder as the argument to the `jhsearch` command:

```
jhsearch JavaHelpSearch
```

See also:

The `jhindexer` Command
The `jhsearch` Command
Localizing the Full−Text Search Database
Full−Text Search

4.14 The `jhindexer` Command

The `jhindexer` creates a full−text search database used by the JavaHelp system full−text search engine to locate matches. You can use the `jhsearch` command to verify the validity of the database.

To build a full−text search database use the following commands:

Win32

`jhindexer` `[`*`options`*`] [`*`file`*` | `*`folder`*`]*`

Solaris/SPARC

`jhindexer` `[`*`options`*`] [`*`file`*` | `*`folder`*`]*`

If the argument is a folder, the folder is searched recursively for JavaHelp system content files.

The following options are available:

−c *file*	A configuration file name. See Config File below.
−db *dir*	The name of the database output folder. By default the output folder is named `JavaHelpSearch` and is created in the current folder.

`-locale` *lang_country_variant*
The name of the locale as described in `java.util.Locale`. For example: `en_US` (English, United States) or `en_US_WIN` (English, United States, Windows variant).

`-logfile` *file*
Captures `jhindexer` messages in a specified file. You can use this option to preserve `jhindexer` output on Win32 machines where the console window is dismissed after execution terminates.

`-nostop words`
Causes stop words to be indexed in the full−text search database.

`-verbose`
Displays verbose messages while processing.

4.14.1 Stop Words

You can direct the JavaHelp system's full−text search indexer to exclude certain words from the database index. These words are called *stop words*. By default, the indexer ignores (does not index) the following stop words when it encounters them in your help topics:

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

You can override the indexer's default stop word behavior in two ways:

- Use the `−nostopwords` option with the `jhindexer` command to force the indexer to ignore stop words and to index every word in your help topics.
- Use the `config` file to specify your own list of stop words.

4.14.2 `Config` File

You can use the `config` file to:

- Change the path names of the files as they are stored in the search database. Use this option when you create the search database with paths to topic files that are different from the paths the help system will later use to find them.
- Explicitly specify the names of the topic files you want indexed.
- Specify your own list of stop words.

Each of these options is described below.

4.14.2.1 Changing Path Names

You can remove and prepend portions of the topic file names as they are stored in the search database. This is useful when the path to the topic files you use during development is different from the path the help system will later use to find the topic files during searches.

To remove a portion of the path name from all of the indexed files:

Add the following line to the `config` file:

```
IndexRemove path
```

where *path* is the portion of the path you want removed.

For example, to change:

```
/public_html/JavaHelp/demo/docs/file.html
```

to:

```
docs/file.html
```

add the following line to the `config` file:

```
IndexRemove /public_html/JavaHelp/demo/
```

▶ **To prepend a different path to the indexed files:**

Add the following line to the `config` file:

```
IndexPrepend path
```

For example, to change:

```
docs/file.html
```

to:

```
my_product/install/docs/file.html
```

add this line to the `config` file:

```
IndexPrepend my_product/install/
```

4.14.2.2 Specifying Files for Indexing

You can explicitly specify the names of the files you want indexed. In the `config` file, specify the names in a list in the following format:

```
File filename
File filename
File filename
      .
      .
      .
```

Be sure to use "/" as the file separator when specifying files for indexing.

4.14.2.3 Specifying Stop Words

You can specify your own list of stop words in the `config` file. When you specify your own list, the indexer does not use the default stop word list. You can specify a list of stop words in two ways:

- Add the list of words directly to the `config` file. Use the following format:

  ```
  StopWords word, word, word...
  ```
- In the `config` file, specify the name of a file that contains a list of stop words:

```
    StopWordsFile filename
```

The stop words file must list each stop word on its own line.

See also:

Creating the Full−Text Search Database
The `jhsearch` Command
Localizing the Full−Text Search Database

4.15 The `jhsearch` Command

The `javahelp\bin\jhsearch` command is a command−line program that you can use to query the JavaHelp system full−text search database created with the `jhindexer` command. The `jhsearch` command can be used to verify the validity of a search database without invoking the help viewer.

To use `jhsearch`:

Win32

`jhsearch` *`database_folder`*

Solaris/SPARC

`jhsearch` *`database_directory`*

By default, the `jhindexer` command creates the database files in a database folder named `JavaHelpSearch`.

See also:

Creating the Full−Text Search Database
The `jhindexer` Command

4.16 Opening Popup and Secondary Windows From an HTML Topic

In an HTML topic, you can link to any help topic in your help system and display it in a number of different windows.

- You can link to the help topic by using the standard `<a href=`*URL*`>` syntax. The help topic opens in the current help viewer as another topic.
- You can use the `<object>` tag to open a popup window.
- You can use the `<object>` tag to open a secondary window.

Secondary windows and popups are implemented by a the lightweight component class `JHSecondaryViewer`. You use this class in an HTML topic by defining an `<object>` tag with the appropriate parameters. This topic describes how these windows look and how to use this tag to display popup and secondary windows.

Popups and secondary windows are not used in this *JavaHelp System User's Guide* because they cannot be included in the PDF version. To see actual examples of how these windows can be used, experiment with the `object` demo located in the following directory:

```
demos\bin
```

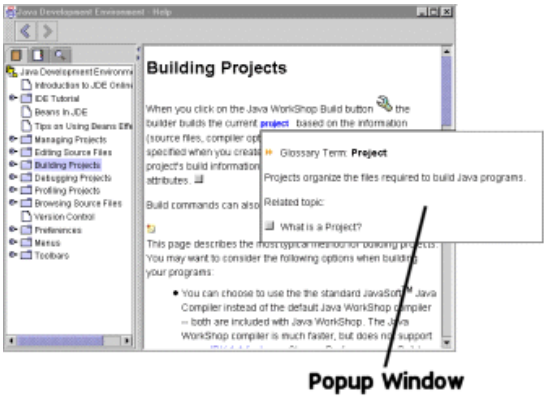
You can also launch the object demo program by using shortcuts (program groups, desktop icons, links) that you might have created during the installation of the JavaHelp system.

4.16.1 Differences Between Popups and Secondary Windows

Popups and secondary windows are functionally similar. They are both fully capable HTML windows that can include graphics, links, and lightweight components. Most of the features described in this topic apply to both types of windows. The following lists describe their differences:

Popups:

- Are always displayed directly adjacent to the object the user clicks to activate the popup.
- Cannot be resized or moved by the user.
- Are dismissed whenever focus is changed to another part of the help viewer.
- Have only a content viewer.



Building Projects

When you click on the Java WorkShop Build button the builder builds the current project based on the information (source files, compiler opt specified when you creat project's build information attributes.

Build commands can also

This page describes the most typical method for building projects. You may want to consider the following options when building your programs:

- You can choose to use the the standard JavaSoft™ Java Compiler instead of the default Java WorkShop compiler -- both are included with Java WorkShop. The Java WorkShop compiler is much faster, but does not support

Glossary Term: **Project**

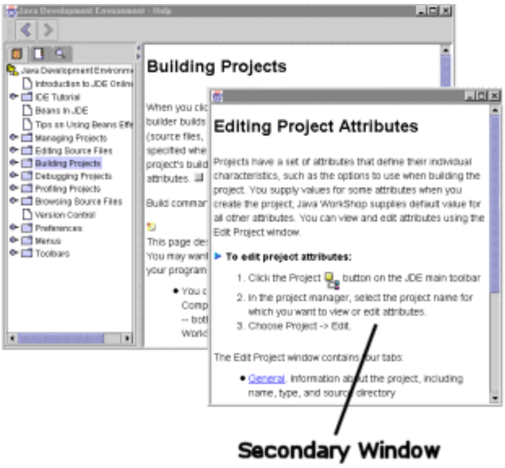
Projects organize the files required to build Java programs.

Related topic:

What is a Project?

Secondary windows:

- Can be displayed anywhere on the screen.
- Can be iconified, resized, and moved by the user.
- Persist until they are dismissed or the help viewer is dismissed.
- Can contain a navigation pane and a toolbar, but by default have a content viewer only.



Building Projects

Editing Project Attributes

Projects have a set of attributes that define their individual characteristics, such as the options to use when building the project. You supply values for some attributes when you create the project; Java WorkShop supplies default value for all other attributes. You can view and edit attributes using the Edit Project window.

To edit project attributes:

1. Click the Project button on the JDE main toolbar
2. In the project manager, select the project name for which you want to view or edit attributes.
3. Choose Project -> Edit.

The Edit Project window contains four tabs:

- General. Information about the project, including name, type, and source directory

4.16.2 Working with Popups and Secondary Windows

You define the characteristics of a popup or secondary window by using a set of parameters with the `<object>` tag. For example, the following `<object>` tag definition creates a secondary window that uses the presention named "secondary window" defined in the helpset file. The secondary window is activated by clicking the text object "Click here".

```
<object classid="java:com.sun.java.help.impl.JHSecondaryViewer">
     <param name="content" value="../topicB/task_topic.html">
     <param name="viewerActivator" value="javax.help.LinkLabel">
     <param name="viewerStyle" value="javax.help.SecondaryWindow">
     <param name="viewerSize" value="300,400">
     <param name="viewerName" value="secondary window">
     <param name="text" value="Click here">
     <param name="textFontFamily" value="SansSerif">
     <param name="textFontSize" value="x-large">
     <param name="textFontWeight" value="plain">
     <param name="textFontStyle" value="italic">
     <param name="textColor" value="red">
</object>
```

This popup object has the following characteristics:

- The content of the window is the file at the location `../topicB/glossary_def.html`.
- The object that the user clicks (`viewerActivator`) is a link.
- The type of window (`viewerStyle`) is a popup.
- The size of the window (`viewerSize`) is 300 pixels wide by 400 pixels high.
- The text that the user sees in the link is "Click here".
- The remaining `param` values define the font, font size, weight, style, and color of the text that the user sees.

The `<param>` element specifies parameters to the JHSecondaryViewer component. The `<param>` element takes two attributes: `name`, and `value`. Parameters can be specified in any order. If parameters conflict, the one specified last is used. Valid parameter names are:

- `viewerStyle`
- `content`
- `viewerActivator`
- `viewerSize`
- `viewerLocation`
- `viewerName`
- `iconByName`
- `iconByID`
- `text`
- `textFontFamily`
- `textFontSize`
- `textFontWeight`
- `textFontStyle`
- `textColor`

4.16.3

The following sections describe each element of the object tag definition and provide examples of a parameter that performs a particular function.

4.16.4 Window Type (`viewerStyle`)

The type of window, popup or secondary, is defined by the following parameter:

```
<param name="viewerStyle"
       value="javax.help.Popup"|"javax.help.SecondaryWindow">
```

If you omit this parameter, the window defaults to a secondary window.

For example, the following parameter specifies a popup:

```
<param name="viewerStyle" value="javax.help.Popup">
```

4.16.5 Content or ID

The content of the object is defined by one of the following parameters:

```
<param name="content" value="URL >
<param name="id" value="MapID >
```

The content of popups and secondary windows is rendered by the same HTML engine used in the main help viewer. Anything that is rendered in the main help viewer can be used in a popup or secondary window, including links, graphics, and lightweight components (for example, popup/secondary windows). You can specify the topics displayed in a window by using a URL or a JavaHelp system map ID.

If you use a link in a popup or secondary window, whatever you link to will also be displayed in that same window. Therefore, links are not often used in these kinds of windows because it is preferable to keep users in the main viewer where they have access to the TOC, index, and other navigational tools.

4.16.6 Activation (`viewerActivator`)

You indicate the object that the user clicks to activate the window with the following parameter:

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

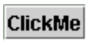
Users activate popup/secondary windows by clicking one of the following objects:

4.16.6.1 Button ( )

This object is a standard button provided as part of the popup or secondary window component. You can use the button as pictured to the left or you can specify a string of text or an image to replace the ">" character on the button.

The following sample code defines a button with the text "ClickMe":

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

Here's what this button looks like: 

The following sample code defines a button that has a GIF image on it:

```
<param name="viewerActivator" value="javax.help.LinkButton">
<param name="text" value="../images/popup_icon.gif">
```

Here's what this button looks like:

4.16.6.2 Text object

This object is a specified string of text inserted inline with the text of the topic. You can control font characteristics of the text to make it stand out.

For example, following parameters define the text object "Click here" to be SanSerif, medium in size, bold, italic, and blue:

```
     <param name="viewerActivator" value="javax.help.LinkLabel">
     <param name="text" value="Click here">
     <param name="textFontFamily" value="SansSerif">
     <param name="textFontSize" value="medium">
     <param name="textFontWeight" value="bold">
     <param name="textFontStyle" value="italic">
     <param name="textColor" value="blue">
```

Here's what this text looks like: ***Click here***

4.16.6.3 Graphic object

This object is a GIF or JPG image.

The following sample code defines a GIF image from the file `rel_topic_button.gif`:

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

Here's what this button looks like:

4.16.7 Window Size (`viewerSize`>, Location (`viewerLocation`) , and Name (`viewerName`)

You can specify the height and width of a popup or secondary window. When content exceeds the size of the window, scroll bars are automatically added to the window. For example, to specify a width of 300 pixels and a height of 400 pixels, use the following parameter:

```
    <param name="viewerSize" value="300,400">
```

For secondary windows, you can also specify the window's position and name.

Window Location

You can specify the position of secondary windows on the screen. The parameter specifies the x,y position (in pixels) of the upper left corner of the secondary window on the screen, with 0,0 being the upper left corner of the screen. Popups ignore this parameter. For example, the following parameter specifies a window whose top left corner is 300 pixels from the left side of the screen and 400 pixels from the top:

```
    <param name="viewerLocation" value="300,400">
```

Named Windows (secondary windows only)

You can name secondary windows. Doing so enables you to reuse an already active window. Popups ignore this parameter.

For example, the following parameter defines a window with the name glossary_window:

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

4.16.8 Text

When you add text to a button or use a text object as an activator, you can control the following font characteristics:

Font family

Parameter name: `textFontFamily`

You can set the font family to:

Serif
SansSerif
Monospaced
Dialog
DialogInput
Symbol
For example,

```
<param name="textFontFamily" value="SansSerif">
```

Font size

Parameter name: `textFontSize`

You can set the size of the font to:

xx−small
x−small
small
medium
large
x−large
xx−large
bigger (Increases the current base font size by 1)
smaller (Decreases the current base font size by 1)
*nn*pt (Sets the font size to a specific point value of *nn*)
+*n* (Increases the current base font size by a value of *n*)
−*n* (Decreases the current base font size by a value of *n*)
n (Sets the font size to the point size associated with the index *n*)
For example,

```
<param name="textFontSize" value="x−large">
```

Font weight

Parameter name: `textFontWeight`

You can set the weight of the font to:

plain
bold
For example,

```
<param name="textFontWeight" value="plain">
```

Font style

Parameter name: `textFontStyle`

You can set the style of the font to:

plain
italic
For example,

```
<param name="textFontStyle" value="italic">
```

Font color	You can set the color of the font to:
Parameter name: `textColor`	black blue cyan darkGray gray green lightGray magenta orange pink red white yellow For example, `<param name="textColor" value="red">`

See also:

Creating Lightweight Java Components

4.17 Merging Helpsets

The JavaHelp system provides a mechanism for merging multiple helpsets into a single helpset. You use the merge functionality to merge a helpset's views (TOC, index, glossary, favorites, and full text search) into those of an existing helpset, known as the *master* helpset.

An example of this functionality is a suite of applications, each with its own helpset. Each time the customer installs a new application that is part of the suite, its help information is merged with the help information in the currently installed applications in the suite.

Merging of helpsets can be done statically, by specifying helpsets in the XML code of a master helpset's helpset (`.hs`) file, or dynamically, by writing code in a Java program that uses the JavaHelp software API. The type of merge your software performs depends on the structure of your applications. For example:

- If you know which helpsets could be available, as you would in a suite of applications, you might prefer to perform a static merge, which merges specific, named helpsets if they are installed.
- If you do not know which helpsets might be available, as might be the case with a developer platform that could have any number of modules or plugins, you might prefer to do a dynamic merge.

When a helpset is merged, there are four types of merges that can be performed for each view: SortMerge, UniteAppendMerge, AppendMerge, and NoMerge. Each type of view has its own default merge type. The help author can override a view's default merge type in the helpset (`.hs`) file by setting a view's `mergetype` property. (For example, see the TOC and Index views defined in the example under Helpset File Format.) The help writer can also set a specific merge type for an element of a TOC, an index, or a glossary. (For example, see the sample TOC in Table of Contents File.)

For more information on dynamic merging of helpsets, see the Developer topic Merging Helpsets Dynamically.

The rest of this section covers various aspects of setting up helpsets for merging, which can be done by a help author without the need for Java programming.

4.17.1 The Master Helpset

When merging helpsets, there must be an initial helpset into which all other helpsets are merged. This initial helpset is called the *master* helpset. This helpset can simply be the first one installed, or it can be a specially designed dataless master helpset. All other helpsets are merged into the master helpset.

The master helpset controls which views can be merged. For example, if the master helpset does not have a Glossary view, but a helpset being merged does have such a view, the Glossary view is not displayed in the merged helpset.

The master helpset can be an actual, functioning helpset or it can be empty (*dataless*). A dataless master helpset defines a set of views that do not contain data (there is no `<data>` tag specified for the views). The dataless master helpset serves as a container into which subhelpsets can be merged. You might use a dataless master to ensure that a set of views is shown in the merged helpset.

For example, the following code defines a dataless master helpset for a suite of applications. The helpset does a static merge: it declares a series of subhelpsets and the JavaHelp system merges those helpsets if they are installed on the user's system.

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
   <!DOCTYPE helpset
     PUBLIC
     "-//Sun Microsystems Inc.//DTD
      JavaHelp HelpSet Version 2.0//EN"
      "http://java.sun.com/products/javahelp/helpset_2_0.dtd">

   <helpset version="2.0">
      <!-- title -->
      <title>JavaHelp System User's Guide</title>

      <!-- views -->
      <view>
         <name>TOC</name>
         <label>Table Of Contents</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>Search</label>
         <type>javax.help.SearchView</type>
      </view>

      <subhelpset location="app1.hs" />
      <subhelpset location="app2.hs" />
      <subhelpset location="app3.hs" />
      <subhelpset location="app4.hs" />

   </helpset>
```

4.17.2 Understanding Merge Types

The JavaHelp system provides four merge types that control how helpsets are merged: `javax.help.UniteAppendMerge`, `javax.help.SortMerge`, `javax.help.AppendMerge`, and `javax.help.NoMerge`. Each view of a helpset (TOC, index, search, glossary, and favorites) has a default merge type that can be overridden by the help author in the helpset's Helpset (`.hs`) file by setting the `mergetype` attribute for that view, as described later in Using Merge Types. (As also described in that section, it is possible to set a merge type for an element of a view, such as an index entry.)

When a helpset is merged into a master helpset, each view is merged according to the merge type that has been set for it.

4.17.2.1 `javax.help.UniteAppendMerge`

This merge type preserves the hierarchy of the master view by merging matching elements at the same level into one element, and then merging and sorting any sub−elements of the matching elements. Any remaining elements are appended to the end of the view. This type of merging works well for TOC views.

Merged elements can be united into a single element only if they are identical. For example, if there is a folder in the master TOC that has a target (a link to a help topic) and a folder in a helpset being merged that has the same name but a different target, these two folders will appear separately in the merged TOC, with the helpset in parentheses after the folder name. If you want the folders to merge into a single element, they must match exactly (have the same targets or no targets at all).

- UniteAppendMerge is slower than SortMerge and is much slower than AppendMerge. If you do UniteAppend merges with the TOCs of multiple helpsets, you might want to test the merge speed of helpsets that have large TOCs to ensure that the speed is adequate for your users.
- UniteAppendMerge cannot detect if a navigator (for example, a TOC) that is being merged has entries in it that duplicate each other. If you do not want duplicate entries in the merged TOC, you must ensure that the TOC for each helpset is constructed properly and has no entries that duplicate each other. (An example of duplicated entries is a TOC that lists the same entry in more than one place for organizational purposes).

In the following example from the New Merge sample helpset, there is a master TOC with place holders for specific topics in the TOCs of the helpsets that the help writer expected to be merged. The master helpset is dataless. Its purpose is to ensure that the views of the subhelpsets appear in a particular order. The unmerged TOCs look like this:

Master TOC	Vertebrates TOC	Invertebrates TOC

Animal Categories
   Vertebrates
 	
Invertebrates
	Animal Categories
	   Vertebrates
	      Fish
	      Amphibians
	      Reptiles
	      Birds
	      Mammals
	         Marsupials
	         Primates
	         Rodents
	         Cetaceans
	         Animals Like Seals
	   Pictures *`[Sort merge]`*
	      Bat
	      Bears
	         Black Bear
	         Grizzly
	         Koala
	      Bird
	      Crocodile
	      Dolphin
	      Elephant
		Animal Categories
		   Invertebrates
		      Protozoa
		      Echnioderms
		      Annilids
		      Mollusks
		      Arthropods
		         Crustaceans
		         Arachnids
		         Insects
		   Pictures *`[Sort merge]`*
		      Butterfly
		      Clam
		      Crab
		      Dragon
		      Sea Star
		      Spider
		      Sponge
		      Worms

	Fish Frog Giraffe Kangaroo Leopard Lizard Monkey Orca Seal Shark Snake Wolves Arctic Wolf Timber Wolf	

By applying the UniteAppendMerge rules to the the TOCs being merged, the JavaHelp system produces a merged helpset with the following characteristics:

- The Vertebrates and Invertebrates TOC items appear in the same order as in the master TOC.
- There is a new Pictures TOC item that is appended after the Invertebrates TOC item.
- The invertebrate and vertebrate sub−elements of the Pictures TOC element are sorted canonically. (The Pictures subnode is defined to use Sort and not UniteAppendMerge, which is why all the picture subitems are sorted in the merged TOC below.)

The following table shows the entire merged helpset:

Merged TOC

- Animal Categories
  - Vertebrates
    - Fish
    - Amphibians
    - Reptiles
    - Birds
    - Mammals
      - Marsupials
      - Primates
      - Rodents
      - Cetaceans
      - Animals Like Seals
  - Invertebrates
    - Protozoa
    - Echnioderms
    - Annilids
    - Mollusks
    - Arthropods
      - Crustaceans
      - Arachnids
      - Insects
- Pictures
  - Bat
  - Bears
    - Black Bear
    - Grizzly
    - Koala
  - Bird

- Butterfly
- Clam
- Crab
- Crocodile
- Dragon
- Dolphin
- Elephant
- Fish
- Frog
- Giraffe
- Kangaroo
- Leopard
- Lizard
- Monkey
- Orca
- Seal
- Sea Star
- Shark
- Snake
- Spider
- Sponge
- Wolves
  - Arctic Wolf
  - Timber Wolf
- Worms

4.17.2.2 `javax.help.SortMerge`

View data is collated at each level of the view according to the helpset's locale collation rules. (The view is sorted canonically.) If there is an entry in the current master or merged helpset that has the same name and ID as an entry being merged, the merged entry is ignored (the two entries become one entry in the merged view). If the names are the same, the helpset title is added in parentheses to the end of the entry.

This merge type is the default type for the Search view. This type of merge is useful when you have information that is collated, such the elements of an Index or Glossary view. It is not useful when you have information that is in hierarchical form, such as a TOC. To use this merge type for the Index and Glossary views, you must override the default type of those views. The previous example does a SortMerge for the Pictures subitems.

SortMerge is slower than AppendMerge, but faster than UniteAppendMerge. If you do sort merges with the indexes of multiple helpsets, you might want to test the merge speed of helpsets that have large indexes to ensure that the speed is adequate for your users.

SortMerge cannot detect if a navigator (for example, an index) that is being merged has entries in it that duplicate each other. If you do not want duplicate entries in the merged index, you must ensure that the index for each helpset is constructed properly and has no entries that duplicate each other. (An example of duplicate entries is an index with multiple "space" entries added for vertical spacing.)

In the example below, the Edit, File, and Help Menu entries have the same text, but point to different IDs. When the JavaHelp system merges the two, it distinguishes them by adding by the helpset titles `(Java Workshop)`and `(Java Studio)`.

Java Workshop Index	**Java Studio Index**	**Merged Index**

Menus Build Menu Debug Menu Edit Menu File Menu Help Menu Toolbars Edit/Debug Toolbar Main Toolbar	Developer Resources Examples List of Additional Examples Step−by−step Example Menus Edit Menu File Menu Help Menu View Menu Toolbars Composition Toolbar Main Toolbar	Developer Resources Examples List of Additional Examples Step−by−step Example Menus Build Menu Debug Menu Edit Menu (Java Workshop) Edit Menu (Java Studio) File Menu (Java Workshop) File Menu (Java Studio) Help Menu (Java Workshop) Help Menu (Java Studio) View Menu Toolbars Composition Toolbar Edit/Debug Toolbar Main Toolbar (Java Workshop) Main Toolbar (Java Studio)

4.17.2.3 `javax.help.AppendMerge`

This merge type appends the new view data (data in the view being merged) to the end of the existing view data. No attempt is made to merge identical entries or sort the results.

This type of merge was the only one available in version 1 of the JavaHelp software for the TOC and Index views. To maintain compatibility with this earlier version of JavaHelp software, this merge type is the default for these two views. It is also the default merge type for the Glossary view. If you want the resulting merged Index and Glossary views to be sorted canonically, you must set the `mergetype` attribute for these views to `javax.help.SortMerge` in the Helpset file.

4.17.2.4 `javax.help.NoMerge`

No merging is performed: the view does not appear in the merged helpset. This merge type applies to an entire view, not to elements of a view. It is the default merge type for the Favorites view because that view is user−dependent and is stored in a single file in the user's directory.

4.17.3 Using Merge Types

As described in the previous section, the JavaHelp system provides four merge types that control how helpsets are merged: UniteAppendMerge, SortMerge, AppendMerge, and NoMerge. Each view of a helpset (TOC, index, search, glossary, and favorites) has a default merge type.

The default merge types for each type of view are:

- **TOC**. `javax.help.AppendMerge`
- **Index**. `javax.help.AppendMerge`
- **Search**. `javax.help.SortMerge`
- **Glossary**. `javax.help.AppendMerge`
- **Favorites**. `javax.help.NoMerge`

You can override these merge types for an entire view in the Helpset file, or you can specify a merge type for an element of a view, such as an index entry.

4.17.3.1 Specifying Merge Types in the Helpset File

You can override the default merge type for each view in a helpset. For example, you can merge the TOC views by using UniteAppendMerge and the Index views by using SortMerge. To override a view's merge type, you set the `mergetype` attribute for that view in the view's Helpset (Hs) file.

When a helpset is merged into a master helpset, each view is merged according to its default merge type or the merge type that you have set for it.

For example, a merge type that works well with indexes is SortMerge. The following code shows how to specify this merge type for an index:

```
<view mergetype="javax.help.SortMerge">
      <name>Index</name>
      <label>Index</label>
      <type>javax.help.IndexView</type>
      <data>AnimalsIndex.xml</data>
   </view>
```

4.17.3.2 Specifying Merge Types in the View Files

You can override the view's merge type for each element of a TOC or Index view. Unless overridden again, the merge type definition applies to any subitems of the entry.

- In a TOC, the `<tocitem>` tag supports the `mergetype` attribute.

  In the following example, the `Release Information` entry and its three subitems all have the `javax.help.SortMerge` merge type:

  ```
  <tocitem text="Release Information" mergetype="javax.help.SortMerge">
      image="chapter" target="rel.release">
    <tocitem text="Contents of the Release"
        image="topic" target="rel.contents"/>
    <tocitem text="Requirements"
        image="topic" target="rel.requirements"/>
    <tocitem text="Changes Since the 1.0 Release"
        image="topic" target="rel.changes"/>
  </tocitem>
  ```

- In an index, the `<indexitem>` tag supports the `mergetype` attribute.

  In the following example, the `context sensitive help` entry and its three subitems all have the `javax.help.UniteAppendMerge` merge type:

  ```
  <indexitem text="context sensitive help" <br>
             mergetype="javax.help.UniteAppendMerge">
    <indexitem text="window-level help"
       target="auth.csh.window-level" />
    <indexitem text="field-level help"
       target="auth.csh.field-level" />
    <indexitem text="menu help"
       target="auth.csh.menu" />
  </indexitem>
  ```

4.17.4 Merging Helpsets Statically

You can specify which help sets to merge in two ways:

- **Dynamically**. You can write Java code that searches for helpsets and uses the JavaHelp software APIs to merge any that finds. Since this is a programming task and not something a help author is expected to do, this technique is described separately in the Developer topic Merging Helpsets Dynamically.
- **Statically**. You can add a `<subhelpset>` tag to the master helpset file to explicitly add helpsets. If the IDE does not find a specified subhelpset, it ignores the tag.

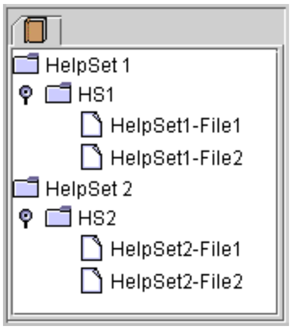
To merge helpsets statically, add `<subhelpset>` tags to a master helpset file to specify other helpsets that you want to merge with the original helpset. The merge operation is performed whenever the containing helpset is instantiated.

In the following simple example, `HelpSet2` is merged with `HelpSet1` to produce the unified TOC display shown below the code sample:

```
-HelpSet1.hs-
   <?xml version='1.0' encoding='ISO-8859-1' ?>
   <!DOCTYPE helpset
     PUBLIC
     "-//Sun Microsystems Inc.//DTD
      JavaHelp HelpSet Version 2.0//EN"
     "http://java.sun.com/products/javahelp/helpset_2_0.dtd">
   <helpset version="2.0">
      <title>HelpSet 1</title>
      <maps>
        <homeID>hs1_file</homeID>
        <mapref location="hs1.jhm" />
      </maps>
      <view mergetype="javax.help.UniteAppendMerge">
         <name>TOC</name>
         <label>Table Of Contents</label>
         <type>javax.help.TOCView</type>
         <data>hs1TOC.xml</data>
      </view>
     <subhelpset location="HelpSet2.hs" />
   </helpset>

-HelpSet2.hs-
   <?xml version='1.0' encoding='ISO-8859-1' ?>
   <!DOCTYPE helpset
     PUBLIC
     "-//Sun Microsystems Inc.//DTD
      JavaHelp HelpSet Version 2.0//EN"
     "http://java.sun.com/products/javahelp/helpset_2_0.dtd">
   <helpset version="2.0">
      <title>HelpSet 2</title>
      <maps>
        <homeID>hs2_file</homeID>
        <mapref location="hs2.jhm" />
      </maps>
      <view mergetype="javax.help.UniteAppendMerge">
         <name>TOC</name>
         <label>Table Of Contents</label>
         <type>javax.help.TOCView</type>
         <data>hs2TOC.xml</data>
      </view>
   </helpset>
```

Unified TOC Display



A static merge has the following special features:

- The helpset that contains the `<subhelpset>` tag is considered to be the *master* helpset. All helpsets are merged with the master helpset.
- When helpsets are merged, only views with the same name (`<name>` tag) as a view in the master helpset file are merged. Note that in the example above, both views are named "TOC". Any views in the subhelpsets that do not match the views in the master helpset are not displayed.
- Multiple `<subhelpset>` tags can be included in a helpset file. Helpsets are appended in the order in which they occur in the helpset file. If a helpset specified in a `<subhelpset>` tag is not found, it is ignored and no error is issued.
- The `<subhelpset> location` attribute takes a URL as its argument.

5 Programming with the JavaHelp System

The topics in this chapter of the *JavaHelp System User's Guide* describe the aspects of the JavaHelp system of primary interest to application developers.

The JavaHelp system classes are distributed in the following JAR files. The following redistributable JAR files are located in the `javahelp\lib` folder:

`jh.jar`	The standard library that includes everything needed to use the help viewer and the standard navigator types (TOC, index, full−text search).
`jhbasic.jar`	A subset of `jh.jar` that does not include support for the full−text search engine. This subset might be useful for simple help systems that do not require a full−text search database or for help systems whose size is important.
`jhall.jar`	Includes all the JavaHelp system classes, including the tools required to create a search database.
`jsearch.jar`	The default full−text search engine used in the JavaHelp system.

5.1 Supplemental Information

You will probably find the following supplemental documentation and source code useful:

API	The `javadoc` generated documentation of the JavaHelp software API is included with this release. You can view the API by using the JavaHelp software API viewer (`demos\bin\apiviewer`) or by using a web browser (`doc\api\index.html`).
Specification	The specification for version 2.0 of the JavaHelp system is included in this release and can be found at: `doc\spec\JavaHelp_V2_0_Specification.pdf`
Sample Source Files	This release includes demo programs that demonstrate JavaHelp system functionality. Sources for the demo programs are included in the release at: `demos\src`

See also:

Adding the JavaHelp System to Applications
Accessing a URL Through a Firewall
Implementing Context−Sensitive Help
Merging Helpsets Dynamically
Embedding JavaHelp Components
Creating Lightweight Java Components
Server Based JavaHelp

5.2 Adding the JavaHelp System to Applications

The following code sample adds a JavaHelp system to an application. It is followed by a series of steps explaining more about what is happening in the code:

```
import javax.help.*;
// Find the HelpSet file and create the HelpSet object:
   String helpHS = "myHelpSet.hs";
   ClassLoader cl = ApiDemo.class.getClassLoader();
   try {
      URL hsURL = HelpSet.findHelpSet(cl, helpHS);
      hs = new HelpSet(null, hsURL);
   } catch (Exception ee) {
      // Say what the exception really is
      System.out.println( "HelpSet " + ee.getMessage());
      System.out.println("HelpSet "+ helpHS +" not found")
      return;
   }
// Create a HelpBroker object:
   hb = hs.createHelpBroker();
// Create a "Help" menu item to trigger the help viewer:
   JMenu help = new JMenu("Help");
   menuBar.add(help);
   menu_help = new JMenuItem("Launch Help");
   menu_help.addActionListener(new CSH.DisplayHelpFromSource( hb ));
```

The folllowing steps explain more about the preceding code sample:

1. Import the JavaHelp system classes:

   ```
   import javax.help.*;
   ```

   Be sure to add one of the JavaHelp system libraries (for example, `jh.jar`) to your application's `CLASSPATH`.

2. Find the helpset file and create the helpset object:

   The ApiDemo class is in one of the demo programs. Substitute your own class for this one in your code.

   ```
   String helpHS = "myHelpSet.hs";
   ClassLoader cl = ApiDemo.class.getClassLoader();
   try {
       URL hsURL = HelpSet.findHelpSet(cl, helpHS);
       hs = new HelpSet(null, hsURL);
   } catch (Exception ee) {
       System.out.println( "HelpSet " + ee.getMessage());
       System.out.println("HelpSet "+ helpHS +" not found")
       return;
   }
   ```

   In this code sample, `findHelpSet()` takes a `ClassLoader` object as its first parameter. If you add your helpset's directory to the `CLASSPATH`, `findHelpSet()` will find it because it calls `ClassLoader.getResource()`, which searches the directories in the `CLASSPATH` for the helpset file. If `getResource()`finds a `.jar` file under a directory, it opens the `.jar` file and searches in it for the helpset file.

   For more information on setting the class path, see http://java.sun.com/j2se/1.4.2/docs/tooldocs/solaris/classpath.html.

3. Create a HelpBroker object:

   ```
   hb = hs.createHelpBroker();
   ```

4. Create a "Help" menu item to trigger the help viewer:

```
JMenu help = new JMenu("Help");
menuBar.add(help);
menu_help = new JMenuItem(("Launch Help");
menu_help.addActionListener(new CSH.DisplayHelpFromSource( hb ));
```

5.2.1 Helpset

The first thing your application does is read the helpset file specified by the application. The helpset file defines the *helpset* for that application. A helpset is the set of data that constitutes your help system. The helpset file includes the following information:

Map file
: The map file is used to associate topic IDs with the URL or path name of HTML topic files.

View information
: Information that describes the navigators being used in the helpset. The standard navigators are: table of contents, index, and full−text search. Information about custom navigators is included here as well.

Helpset title
: The title of the helpset as defined in the helpset file's `<title>` tag.

Home ID
: The name of the (default) ID that is displayed when the help viewer is called without specifying an ID.

Sub−helpsets
: This optional section can be used to statically include other helpsets by using the tag. The helpsets indicated by this tag are merged automatically into the helpset that contains the tag. More details about merging can be found in Merging Helpsets.

For more information about the helpset file, see Helpset File.

5.2.2 HelpBroker

The HelpBroker is an agent that negotiates and manages the display of help content for your application. The HelpBroker also provides "convenience" methods that you can use to implement context−sensitive help. See Implementing Context−Sensitive Help for details.

You can implement a help system without using the HelpBroker. However, without the HelpBroker you have to write code to directly manage the HelpViewer and JHelp objects, navigators, and context−sensitive help functionality (F1 key on dialogs, help button activation, and on item help button/menu activation).

For a list and description of the HelpBroker methods, see the API at: doc\api\javax\help\HelpBroker.html.

See also:

Programming with the JavaHelp System
sAccessing a URL Through a Firewall
Implementing Context−Sensitive Help
Merging Helpsets Dynamically
Embedding JavaHelp Components
Creating Lightweight Java Components
Server Based JavaHelp

5.3 Implementing Context−Sensitive Help

The JavaHelp system provides classes and methods that help you implement context−sensitive help in your applications. The following sections:

- Summarize the context−sensitive help system
- Describe the basic elements of the system
- Describe how to implement context−sensitive help
- Describe the APIs that support dynamic ID assignment

5.3.1 Summary

The following table summarizes the context−sensitive help system.

CSH Type	***Activation Mechanism***	***Object for Which Help Is Provided***	***Implementation Steps***
Window−Level	Press F1 (or Help) key.	Window with focus	• Set helpIDs for components • Capture F1 key press • Get window that has focus • Get helpID for window • Display help (optionally in a specific *presentation*)
Field−Level	1. Activate field−level help. 2. Navigate with mouse or keyboard. 3. Select object.	Selected object	• Set helpIDs for components • Activate field−level help (button or menu item) • Track context−sensitive events • Get helpID for selected object • Display help (optionally in a specific *presentation*)
Help Button Menu Item	Click button or choose menu item.	Topic associated with button or menu item	• Create button or menu object • Set helpID on object • Get helpID on object • Display help
System Initiated	Internal, varies.	Internal, varies	• Display help (optionally in a specific *presentation*)

5.3.2 Basic Elements

This section describes the low−level elements used in implementing context−sensitive help.

If you use the "convenience" methods in the `HelpBroker` object to implement context−sensitive help, these low−level elements are managed for you.

The basic steps for implementing context−sensitive help are:

1. Set and get `Component` help properties for GUI objects
2. Track context−sensitive events

5.3.2.1 Setting and Getting Component Help Properties

To provide context−sensitive help for GUI Components and menu items, you must associate a help ID with that `Component` or menu item. To make that association, you set the `helpID` property and (if you use multiple helpsets) the `HelpSet` for the `Component` or `MenuItem` The JavaHelp system `CSH` class provides the following convenient methods for setting and getting the `helpID` for `Components` and `MenuItems`:

5.3.2.2 setHelpIDString Component

```
public static void setHelpIDString(Component comp, String helpID);
```

Sets the `helpID` for a component.

5.3.2.3 getHelpIDString Component

```
public static String getHelpIDString(Component comp);
```

Returns the `helpID` for a component.

5.3.2.4 setHelpSet Component

```
public static void setHelpSet(Component comp, HelpSet hs);
```

Sets the `HelpSet` for a component.

5.3.2.5 getHelpSet Component

```
public static HelpSet getHelpSet(Component comp);
```

Returns the `HelpSet` of a component.

5.3.2.6 setHelpIDString MenuItem

```
public static void setHelpIDString(MenuItem comp,
                                   String helpID);
```

Sets the `helpID` for a menu item.

5.3.2.7 getHelpIDString MenuItem

```
public static String getHelpIDString(MenuItem comp);
```

Returns the `helpID` for a menu item.

5.3.2.8 setHelpSet MenuItem

```
public static void setHelpSet(MenuItem comp, HelpSet hs);
```

Sets the helpset for a menu item.

5.3.2.9 getHelpSet MenuItem

```
public static HelpSet getHelpSet(MenuItem comp);
```

Returns the helpset of a menu item.

5.3.2.10 Tracking Context−Sensitive Events

The context−sensitive help class provides the `CSH.trackCSEvents` method, which you can use to easily track context−sensitive events. This method traps all non−navigational events until an object is selected. It returns the selected object. Following is the declaration for the method:

```
public static Component CSH.trackCSEvents()
```

5.3.3 Implementing Context−Sensitive Help

The sections that follow describe how to use the JavaHelp system to implement various forms of context−sensitive help.

5.3.3.1 The `HelpBroker`

The JavaHelp system defines a `HelpBroker` interface that provides convenience methods that greatly simplify the job of implementing context−sensitive help. `HelpBroker` methods hide much of the underlying implementation details. In exchange, using the `HelpBroker` limits the flexibility of your implementation. For example, if you use the `DefaultHelpBroker`, you must display help information in the standard help viewer.

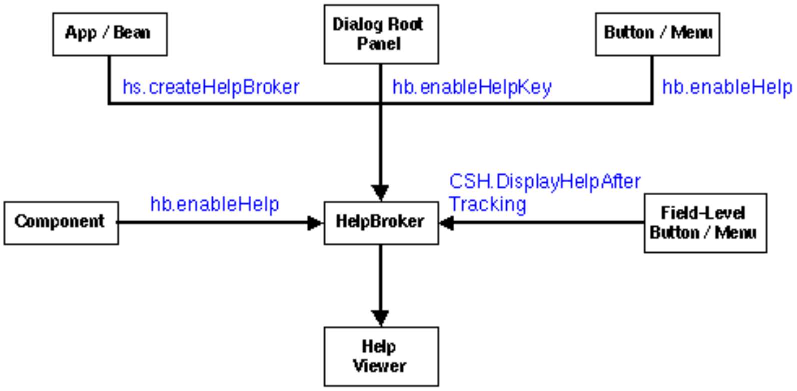
You can implement context−sensitive help without using the `HelpBroker`, or you can use the `HelpBroker` for some tasks and not for others. For example, if your implementation requires something not provided in the `HelpBroker`, such as displaying context−sensitive help in a different viewer, you must use the basic classes (`CSH`, `JHelp`) directly. For information about those classes, use the JavaHelp system `apiviewer` command.

With some of these methods, you can specify the *presentation type*, the type of window in which the help topics is displayed. The examples show generic names for popup windows and secondary windows that will always work. However, it is possible that specific presentation definitions have been provided in the helpset file by the help author. Since the help author can define attributes of presentations, including their size, their position, and the number and type of panes, if presentations are defined in the helpset file, you might want to use their names in help calls.

A `HelpBroker`'s convenience methods enable:

- Window−level help for a dialog box
- Help buttons for dialog boxes
- Buttons and menu items that activate field−level help

The following illustration shows how the HelpBroker and its context−sensitive methods (`hb.*`) are used with other JavaHelp system components:



5.3.3.2 `HelpBroker` Context−Sensitive Methods

A `HelpBroker` provides the following context−sensitive methods:

5.3.3.3 setHelpSet

```
 public void setHelpSet(HelpSet hs);
```

Sets the `HelpSet` for the current `HelpBroker` (there can be only one `HelpSet` per `HelpBroker`). If you use this method to change helpsets, the displays in the corresponding `JHelp` component and `JHelpNavigator` are changed.

5.3.3.4 getHelpSet

```
 public HelpSet getHelpSet();
```

Gets the current `HelpSet` for the `HelpBroker`.

5.3.3.5 setCurrentID

```
 public void setCurrentID(Map.ID id) throws BadIDException;
```

Sets the current ID that is to be displayed in the help viewer. If `hs` is null, the `HelpBroker`'s current `HelpSet` is used. If `hs` is different from the current `HelpSet` (and not contained in the current `HelpSet`), the `setHelpSet` method is executed.

5.3.3.6 setCurrentURL

```
 public void setCurrentURL(URL url, HelpSet hs) throws BadIDException;
```

Displays the specified URL in the help viewer. If `hs` is null, the `HelpBroker`'s current `HelpSet` is used. If `hs` is different from the current `HelpSet` (and not contained in the current `HelpSet`), the `setHelpSet` method is executed.

5.3.3.7 enableHelpKey

```
public void enableHelpKey(Component comp, String id, HelpSet hs,
                          String presentationType, String presentationName);
```

Enables the Help key on a Component (the F1 key on Windows machines). This method works best when the component is the `rootPane` of a `JFrame` in Swing−based applications, or a `java.awt.Window` (or subclass thereof) in AWT−based applications. This method sets the default `helpID` and `HelpSet` for the `Component` and registers keyboard actions to trap the "Help" keypress. If the object with the current focus has a `helpID`, the `helpID` is displayed when the Help key is pressed; otherwise, the default `helpID` is displayed. You can optionally specify the type of help window in which a help topic is displayed.

For example, the following code specifies that the help presentation is a secondary window named `mainSW`:

```
JTextArea newText = new JTextArea();

hb.enableHelp(newText, "debug.overview", hs);
. . .
rootpane = frame.getRootPane();
mainHelpBroker.enableHelpKey(rootpane,
                             "top",
                             null,
                             "javax.help.SecondaryWindow",
                             "mainSW");
```

5.3.3.8 enableHelp Component

```
public void enableHelp(Component comp, String id, HelpSet hs);
```

Enables help activation for a help component (for example, a Help button). This method:

- Registers the `helpID` and `HelpSet` on `comp`
- Sets the `HelpBroker`'s `HelpActionListener` on `comp`

5.3.3.9 enableHelp MenuItem

```
public void enableHelp(MenuItem comp, String id, HelpSet hs)
```

Enables help activation for a `MenuItem`. This method:

- Registers the `helpID` and `HelpSet` on `comp`
- Sets the `HelpBroker`'s `HelpActionListener` on `comp`

5.3.3.10 enableHelpOnButton Component

```
  public void enableHelpOnButton(
                Component comp, String id, HelpSet hs,
                String presentationType,
                String presentationName);
```

Enables help for a `Component`. This method sets the `helpID` and `HelpSet` for the `Component` and adds an `actionListener`. When an action is performed it displays the `Component`'s `helpID` and `HelpSet` in the default viewer. If the `Component` is not a `javax.swing.AbstractButton` or a `java.awt.Button`, an `IllegalArgumentException` is thrown. You can optionally specify the type of help window in which a help topic is displayed.

For example, the following code specifies that the help presentation is a secondary window named `mainSW`:

```
JButton helpButton = new JButton("Help",
                                 "javax.help.SecondaryWindow",
                                 "mainSW");

mainHelpBroker.enableHelpOnButton(helpButton,
                                  "browse.strings",
                                  null,
                                  "javax.help.SecondaryWindow",
                                             "mainSW");
```

5.3.3.11 enableHelpOnButton MenuItem

```
public void enableHelpOnButton(MenuItem comp, String id, HelpSet hs,
                               String presentationType,
                               String presentationName);
```

Enables help for a `MenuItem`. This method sets the `helpID` and `HelpSet` for the `Component` and adds an `actionListener`. When an action is performed it displays the `helpID` and `HelpSet` in the default viewer. You can optionally specify the type of help window in which a help topic is displayed.

5.3.3.12 `CSH` Inner Classes

The `CSH` class contains three inner classes that provide support for context−sensitive help.

5.3.3.13 `CSH.DisplayHelpAfterTracking`

```
CSH.DisplayHelpAfterTracking(HelpSet hs,
                            String presentationType,
                            String presentationName)
```

This class defines an `ActionListener` that displays help for a selected object after tracking context−sensitive events. Its constructor takes a `HelpBroker` object. You can optionally specify the type of help window in which the help topic is displayed. For example, you could display help for a toolbar button in a popup window as follows:

```
JToolBar toolbar=new JToolBar();
. . .
helpbutton= addButton(toolbar, "images/help.gif", "help");
helpbutton.addActionListener(
  new CSH.DisplayHelpAfterTracking (mainHS,
                                    "javax.help.Popup",
                                    null));
```

5.3.3.14 `CSH.DisplayHelpFromFocus`

```
CSH.DisplayHelpFromFocus(HelpSet hs,
                         String presentationType,
                         String presentationName)
```

An `ActionListener` that displays the help of the object that currently has focus. This method is used to enable a `HelpKey` action listening for components other than the `RootPane` or window. This listener determines if the object with the current focus has a `helpID`, and if it does the helpID is displayed. If the object does not have a `helpID`, the `helpID` on the action's source is displayed (if one exists). You can optionally specify the type of help window in which the help topic is displayed.

5.3.3.15 `CSH.DisplayHelpFromSource`

```
CSH.DisplayHelpFromSource(HelpSet hs,
                          String presentationType,
                          String presentationName)
```

An actionListener that gets the `helpID` for the action source and displays the helpID in the help viewer. Its constructor takes a HelpBroker object. You can optionally specify the type of help window in which the help topic is displayed.

5.3.3.16 Window−Level Help

Start your window−level help implementation by setting the helpID and (if you use multiple helpsets) the `HelpSet` for each component for which you want help. If you do not set help for a given component, `CSH.getHelpID()` recursively searches through the component's ancestors until it finds the first ancestor with a helpID, or until it reaches the last ancestor. For example:

```
:
        JTextArea newText = new JTextArea();
        hb.enableHelp(newText, "debug.overview", hs);
                   :
```

After you set the helpID and helpset for all components, use the HelpBroker `enableHelpKey()` method to enable the F1 key for the frame's RootPane. The `hb.getHelpKeyActionListener()` method enables the F1 key on `ActionListener` objects other than root panes. For example, the following code displays the help in the default viewer:

```
:
    rootpane = frame.getRootPane();
    mainHelpBroker.enableHelpKey(rootpane, "top", null);
                   :
```

If you want to display help in a popup window, substitute the following line of code:

```
    mainHelpBroker.enableHelpKey(rootpane, "top", null,
                                 "javax.help.Popup", null);
```

If you want to display help in a secondary window named `mainSW`, substitute the following line of code:

```
    mainHelpBroker.enableHelpKey(rootpane, "top", null,
                                 "javax.help.SecondaryWindow",
                                 "mainSW");
```

5.3.3.17 Field−Level Help

Start your field−level help implementation by setting the helpID and (if you use multiple helpsets) helpset for each component for which you want help. If you do not set help for a given component, `CSH.getHelpID()` recursively searches through the component's ancestors until it finds the first ancestor with a helpID, or until it reaches the last ancestor.

After you set the helpID and helpset for all components, create a field−level help button or menu item. Set an ActionListener on the button or menu item with a HelpBroker object using `getOnItemActionActionListener`. For example:

```
JToolBar toolbar=new JToolBar();
CSH.setHelpID(toolbar,"toolbar.main");
               :
helpbutton= addButton(toolbar, "images/help.gif", "help");
helpbutton.addActionListener(
        new CSH.DisplayHelpAfterTracking(mainHelpBroker));
```

The following invocation would display the field−level help in a popup window:

```
JToolBar toolbar=new JToolBar();
CSH.setHelpID(toolbar,"toolbar.main");
               :
```

```
helpbutton= addButton(toolbar, "images/help.gif", "help");
helpbutton.addActionListener(
        new CSH.DisplayHelpAfterTracking(mainHelpBroker,
                                         "javax.help.Popup",
                                         null));
```

The following invocation would display the field−level help in a secondary window:

```
JToolBar toolbar=new JToolBar();
CSH.setHelpID(toolbar,"toolbar.main");
                :
helpbutton= addButton(toolbar, "images/help.gif", "help");
helpbutton.addActionListener(
        new CSH.DisplayHelpAfterTracking(mainHelpBroker,
                                         "javax.help.SecondaryWindow",
                                         "mainSW"));
```

5.3.3.18 Help Menu and Help Button Help

To implement Help menu or Help button help:

1. Create a menu item or button.
2. Set the `helpID` and (if you use multiple helpsets) the helpset on the object.
3. Enable help on the object with the `HelpBroker`.

The `CSH` class provides the `CSH.DisplayHelpFromSource` class to enable help on objects with types other than `AbstractButton`, `Button`, or `MenuItem`. For example:

```
JButton helpButton = new JButton("Help");
mainHelpBroker.enableHelpOnButton(helpButton, "browse.strings", null);
```

`HelpBroker.enableHelpOnButton` uses `CSH.DisplayHelpFromSource` and also sets the appropriate ID on the `Button` and the `ActionListener` on the `Button`. If this example used `CSH.DisplayHelpFromSource` instead, it would have to set the ID and ActionListener explicitly. Using `HelpBroker` in this example simplifies the code.

5.3.3.19 System Initiated Help

All the other help activations discussed in this section result from the user's clicking a button, pressing a key, or selecting an item in the navigator or content viewer. With system initiated help, the action is not initiated by the user, but rather by the application, which recognizes that the user is need of help and automatically calls the help system. For example, the user might have repeatedly tried an operation that failed every time or canceled a task midway through an operation.

Although system initiated help is rarely implemented with the help viewer, it is simple to do so. When help is presented internally within an application, pass a valid helpID to a HelpBroker object. For example:

```
                            :
        try {
            mainHelpBroker.setCurrentID(helpID);
        } catch (Exception ee) {
            System.err.println("trouble with visiting id; "+ee);
        }
                    :
```

If you wanted the help to display in a popup window, you could use the following code instead:

```
:
        try {
             Popup popup = (Popup)Popup.getPresentation(mainHS,null);
             popup.setInvoker (component);
             popup.setCurrentID (helpID);
             popup.setDisplayed(true);
        } catch (Exception ee) {
            System.err.println("trouble with visiting id; "+ee);
        }
                   :
```

If you wanted the help to display in a secondary window, you could use the following code:

```
:
        try {
           mainHelpBroker.showID(helpID,
                                 "javax.help.SecondaryWindow",
                                 "main");
        } catch (Exception ee) {
          System.err.println("trouble with visiting id; "+ee); }
        }
                       :
```

5.3.3.20 Sample Code

The following example shows the code required for the different types of context−sensitive help using a default helpset:

```
:
   try {
            ClassLoader cl = ApiDemo.class.getClassLoader();
            URL url = HelpSet.findHelpSet(cl, helpsetName);
            mainHS = new HelpSet(cl, url);
        } catch (Exception ee) {
            System.out.println ("Help Set "+helpsetName+" not found");
            return;
        } catch (ExceptionInInitializerError ex) {
            System.err.println("initialization error:");
            ex.getException().printStackTrace();
        }
        mainHB = mainHS.createHelpBroker();
                           :
        // Enable window−level help on RootPane
        rootpane = frame.getRootPane();
        mainHB.enableHelpKey(rootpane, "top", null);
                           :
        // Enable field−level help on various components
        JToolBar toolbar=new JToolBar();
        CSH.setHelpIDString(toolbar,"toolbar.main");
                           :
        //Enable Menu/Button help on Help menu item
        helpbutton= addButton(toolbar, "images/help.gif", "help");
        mainHelpBroker.enableHelpButton(helpbutton, "browser.strings", null);

        sourceIFrame = new JInternalFrame("Source", true, true, true, true);
        CSH.setHelpIDString(sourceIFrame, "edit.editsource");

        JTextArea newtext=new JTextArea();
        CSH.setHelpIDString(newtext, "build.build");

        newtext = new JTextArea();
        CSH.setHelpIDString(newtext, "debug.overview");

        newtext = new JTextArea();
        CSH.setHelpIDString(newtext, "browse.strings");
                           :
        // System Level help somewhere within the code
        try {
            mainHelpBroker.setCurrentID(helpID);
        } catch (Exception ee) {
            System.err.println("trouble with visiting id; "+ee);
        }
                            :
```

5.3.4 Dynamic Map ID Assignment

For certain objects, such as a `JTable`, having a single map ID per object is not sufficient. A technique is needed to determine programmatically the map ID based on cursor position, selection, or some other mechanism inherent to the object. For example a `Canvas` object might determine the map ID based on the object currently selected on the canvas or, alternatively, from the mouse cursor position.

The following APIs in the `CSH` class support dynamic ID assignment:

Name	Description
`addManager(CSH.Manager)`	Registers the specified manager to handle dynamic context−sensitive help.
`addManager(index,CSH.Manager)`	Registers the specified manager to handle dynamic context−sensitive help at the specified position in the list of managers.
`getManager(index)`	Returns the manager at the specified position in list of managers.
`getManagerCount()`	Returns the number of managers registered.
`getManagers()`	Returns array of managers registered.
`removeAllManagers()`	Remove all the dynamic context−sensitive help managers.
`removeManager(CSH.Manager)`	Remove the specified manager from the list of managers.
`removeManager(index)`	Remove the manager at the specified position in the list of managers.

Additionally the following interface has been defined in `CSH.Manager`:

Name	Description
`getHelpSet(Object, AWTEvent)`	Returns the `String` representing the `mapID` of the object based on the `AWTEvent`.
`getHelpIDString(Object, AWTEvent)`	Returns the `HelpSet` of the object based on the `AWTEvent`.

Instances of `CSH.Manager` work as filters. `CSH.getHelpIDString(comp)` and `CSH.getHelpSet(comp)` must call each registered `CSH.Manager`'s `getHelpIDString` or `getHelpSet` methods. If the `CSH.Manager` does not handle the component, it returns `null`. If no `CSH.Manager` provides a `HelpSet` or `HelpIDString` for the component, the `CSH` methods use the statically defined `HelpSet` and `HelpIDString` described in Using Statically Defined Help IDs. As with the statically defined `HelpSet` and `HelpIDString`, a failure in a request for a `HelpSet` and a `HelpIDString` is propagated to the component's parent.

5.3.4.1 Example: Dynamic Map ID Assignment

The following example shows how to use a component with a dynamically assigned `HelpSet` or a dynamically generated `HelpIDString`:

```
class MyCSHManager implements CSH.Manager {
   HelpSet hs;
   JEditorPane editor;
   MyCSHManager(JEditorPane editor, HelpSet hs) {
      this.editor = editor;
      this.hs = hs;
   }
   public HelpSet getHelpSet(Object comp) {
      if (comp == editor) {
         return hs;
      }
      return null;
   }
   public String getHelpIDString(Object comp) {
      if (comp == editor) {
         return getHelpIDFromCaretPostion(editor);
      }
      return null;
   }
}
```

You add the `CSH.Manager` to the `CSH` list of managers as follows:

```
CSH.AddCSHManager(new MyCSHManager(editor, hs));
```

5.3.5 Using Statically Defined Help IDs

Context−sensitive help in the JavaHelp system is organized around the notion of the ID−URL map referred by the <`map`> section of a helpset file. The key concept is that of the `Map.ID`, which is comprised of a `String−HelpSet` instance pair. The `String` is intended to be unique in the local map of the helpset. This is very important when considering helpset merging; otherwise, IDs would be required to be unique over all helpsets that might ever be merged.

There are three tasks involved in assigning context−sensitive help to an application:

1. Defining the appropriate `String` ID−URL map
2. Assigning an ID to each desired visual object
3. Enabling a user action to activate the help

5.3.5.1 Defining the ID−URL Map

The `Map` interface provides a means for associating IDs (`HelpSet.string`) with URLs. One such map is constructed from one or more map files that provide a simpler `String ID` to URL mapping, with the `HelpSet` being given either explicitly or implicitly.

The JavaHelp system has two classes that implement the Map interface: `FlatMap` and `TryMap`. A `FlatMap` does not support nesting of other maps into it, while a `TryMap` does. A `FlatMap` is a simple implementation while `TryMap` should support inverse lookups (for example, `getIDFromURL`) more efficiently. The implementation of `TryMap` in version 1.0 of the JavaHelp system is not particularly efficient.

Both `FlatMap` and `TryMap` have public constructors. The constructor for `FlatMap` takes two arguments:

- A URL to a property file providing a list of `String` and URL pairs
- A `HelpSet`

The HelpSet is used together with each String−URL pair to create the actual Map.ID objects that comprise the FlatMap. The constructor for TryMap has no arguments. Its Map is created empty, and other Maps are added or removed from it.

The Map interface can also be implemented by some custom class. One such class could, for example, be used to programmatically generate the map.

5.3.5.2 Assigning an ID to Each Visual Object

The next step is to assign to each desired GUI object an ID that will lead to the desired help topic. There are two mechanisms that can be involved:

- An explicit ID, either a plain String or a Map.ID, is assigned to the GUI object.

  The two basic methods used to assign IDs are setHelpIDString(Component, String) and setHelpSet(Component, String). If both are applied to a Component, then a Map.ID is assigned to that Component. If only setHelpIDString() is applied, then the HelpSet instance is obtained implicitly, as explained below in the next list item. A method overload is also provided for MenuItem objects.

  These methods take a Component as an argument. The implementation can vary depending on whether the argument is a JComponent or a plain AWT Component.

- A rule is used to infer the Map.ID for a GUI object based on the object's container hierarchy.

  The methods getHelpIDString(Component) and getHelpSet(Component) recursively traverse up the container hierarchy of the component trying to locate a Component that has been assigned a String ID. When found, the methods return the appropriate value. As before there is also an overloaded method for MenuItem.

5.3.5.3 Enabling a Help Action

The final step is to enable an action to trigger the presentation of the help data. CSH currently provides several ActionListener classes that can be used, described above under CSH Inner Classes. In addition, HelpBroker also provides some convenience methods that interact with these ActionListeners, as described above under HelpBroker Context−Sensitive Methods.

Since these methods are from a specific HelpBroker, if a HelpSet is not associated with the GUI object, the HelpSet of the HelpBroker is used automatically.

See also:

Programming with the JavaHelp System
Adding the JavaHelp System to Applications
Accessing a URL Through a Firewall
Merging Helpsets Dynamically
Embedding JavaHelp Components
Creating Lightweight Java Components
Server Based JavaHel

5.4 Merging Helpsets Dynamically

In addition to the static merging of helpsets described in Merging Helpsets, helpsets can be merged dynamically. To merge helpsets dynamically, you use the JavaHelp software API in your Java applications.

The basic API consists of the `HelpSet.add(HelpSet)` method and its corresponding `HelpSet.remove(HelpSet)`method. These methods fire `HelpSetEvent` events for the `HelpSetListeners` that have registered interest in them. The Component UIs for the TOC, index, search, and glossary views register for these events and react to changes.

The semantics of merging is implemented by individual `NavigatorView` objects and `JHelpNavigator` objects. There are three basic methods:

- `canMerge(NavigatorView)`
- `merge(NavigatorView)`
- `remove(NavigatorView)`

The `canMerge(NavigatorView)` method is present in both `NavigatorView` and `JHelpNavigator`. The `JHelpNavigator` method just calls into its corresponding `NavigatorView` method. The other two methods are present only in `JHelpNavigator`.

For more information about these classes and methods, refer to the JavaHelp software API documentation. API information can be viewed using a web browser (`doc/api/overview−summary.html`) or by using the sample API viewer in `demos\bin\apiviewer`.

Two demonstration programs, `demos\bin\merge` and `demos\bin\newmerge` are included with the JavaHelp system release. These programs demonstrate how helpsets can be merged and removed dynamically. The `newmerge` program demonstrates what happens with a UniteAppendMerge of TOCs and a SortMerge of indexes.

The sources for these programs are available under `demos\src\sunw\demo\`.

See also:

Programming with the JavaHelp System
Adding the JavaHelp System to Applications
Accessing a URL Through a Firewall
Implementing Context−Sensitive Help
Embedding JavaHelp Components
Creating Lightweight Java Components
Server Based JavaHelp

5.5 Embedding JavaHelp Components

JavaHelp system components can be embedded directly into the application. The following pared down steps are taken from the idedemo sample program included with the JavaHelp software release. A TOC navigator is added to the main application frame and the content pane is added to a tabbed display at the bottom of the main application frame. You can find the complete sources for the sample program at:

```
demos\src\sunw\demo\idedemo
```

1. Find the helpset file and create the `HelpSet` object:

   ```
   try {
          ClassLoader cl = ApiDemo.class.getClassLoader();
          URL url = HelpSet.findHelpSet(cl, "api");
          apiHS = new HelpSet(cl, url);
      } catch (Exception ee) {
          System.out.println ("API Help Set not found");
          return;
      }
   ```
2. Create the content pane:

```
JHelpContentViewer viewer1 = new JHelpContentViewer(apiHS);
```

3. Add the content pane to a container:

   ```
   messages.setComponentAt(miscTabIndex, viewer1);
   messages.setSelectedIndex(miscTabIndex);
   ```
4. Create a navigator (TOC):

   ```
   xnav = (JHelpNavigator) apiHS.getNavigatorView("TOC").createNavigator(viewer1.getModel());
   ```
5. Use the same model for both the content pane and navigator components (viewer1) to ensure that changes generated by one component are reflected in the other component. Note that the TOC can be created by using either the `HelpSet.getNavigatorView()` method or the `JavaHelpNavigator.getNavigatorView()` method. Both methods produce the same results; however, using `HelpSet` reduces the dependency on the GUI.

6. Add the navigator to the container:

   ```
   content.add(xnav, "Center");
   classViewerIFrame.setContentPane(content);
   ```

See also:

Programming with the JavaHelp System
Adding the JavaHelp System to Applications
Accessing a URL Through a Firewall
Implementing Context−Sensitive Help
Merging Helpsets Dynamically
Creating Lightweight Java Components
Server Based JavaHelp

5.6 Creating Lightweight Java Components

This topic describes how you can create lightweight Java components and add them to HTML topics using the HTML `<OBJECT>` tag. The last section in this topic contains references to supplemental information about lightweight components and the HTML `<OBJECT>` tag.

References to supplemental information are included at the end of this topic.

5.6.1 Lightweight Components for HTML Topics

Components intended for HTML topic pages are very similar to generic lightweight components. Components that do not require information about the View, or have parameters that can be set, can be used without modification.

5.6.1.1 View Information

Lightweight components that require information about the `View` must implement `javax/javahelp/impl/ViewAwareComponent`. These components implement the method `setViewData()`. The component can determine information from the `View` about the environment in which it is executing. For example, in the code snippet below the `Document` object is derived from the `View`:

```
    private View myView;
    static private URL base;

    public void setViewData(View v) {
```

```
    myView = v;
    Document d = myView.getDocument();
    // System.err.println("myDocument is: "+d);

    base = ((HTMLDocument) d).getBase();
    // System.err.println(" base is: "+base);

}
```

For more information about the `Document` interface see the following Swing API:

```
http://java.sun.com/j2se/1.4.1/docs/api/javax/swing/text/Document.html
```

Text formatting information can be derived from the `View` by querying its attribute set. Use the method `getAttributes` as shown below:

```
AttributeSet as = v.getAttributes();
```

Format attributes can be used by the component when the `AttributeSet` is passed as a parameter to a `StyleConstants` method. There are methods that can be used to determine a number of attributes, including the font family, font size, font weight, font style, underlining, background color, and foreground color. For example, to determine the default background color of an object, you can do the following:

```
Color color=StyleContants.getBackground(as)
```

For a full list of formatting attributes and corresponding methods see:

```
http://java.sun.com/j2se/1.4.1/docs/api/javax/swing/text/StyleConstants.html
```

5.6.1.2 Using Parameters

If your component takes parameters, you should follow these two additional steps:

1. Add accessor methods for each parameter that can be set.
2. Create a `BeanInfo` class that corresponds to the lightweight component class.

The component must accept parameter elements in any order.

5.6.1.3 Accessor Methods

Add accessor methods that enable the component to access the parameters through the Java reflection mechanism. In the following example, the `AButton` class has implemented accessor methods for the parameter "data" in the methods `getData` and `setData`:

```
private String data = "";

public void setData(String s) {
    data = s;
}

public String getData() {
    return data;
}
```

Even if the internal representation is not a `String`, both the returned value for the getter method and the parameter in the setter method must be a `String`.

5.6.1.4 BeanInfo Class

Create a `BeanInfo` class that provides explicit information about the lightweight component. The only method used by the `ContentViewer` from the `BeanInfo` classes is `getPropertyDescriptors`. In the complete example below, `JHSecondaryViewerBeanInfo` defines the property data accessible through the `getData()` and `setData()` methods in `JHSecondaryViewer`:

```
public class JHSecondaryViewerBeanInfo extends SimpleBeanInfo {

   public JHSecondaryViewerBeanInfo() {
   }

   public PropertyDescriptor[] getPropertyDescriptors() {
      PropertyDescriptor back[] = new PropertyDescriptor[15];
      try {
         back[0] = new PropertyDescriptor("content", JHSecondaryViewer.class);
         back[1] = new PropertyDescriptor("id", JHSecondaryViewer.class);
         back[2] = new PropertyDescriptor("viewerName", JHSecondaryViewer.class);
         back[3] = new PropertyDescriptor("viewerActivator", JHSecondaryViewer.class);
         back[4] = new PropertyDescriptor("viewerStyle", JHSecondaryViewer.class);
         back[5] = new PropertyDescriptor("viewerLocation", JHSecondaryViewer.class);
         back[6] = new PropertyDescriptor("viewerSize", JHSecondaryViewer.class);
         back[7] = new PropertyDescriptor("iconByName", JHSecondaryViewer.class);
         back[8] = new PropertyDescriptor("iconByID", JHSecondaryViewer.class);
         back[9] = new PropertyDescriptor("text", JHSecondaryViewer.class);
         back[10] = new PropertyDescriptor("textFontFamily", JHSecondaryViewer.class);
         back[11] = new PropertyDescriptor("textFontSize", JHSecondaryViewer.class);
         back[12] = new PropertyDescriptor("textFontWeight", JHSecondaryViewer.class);
         back[13] = new PropertyDescriptor("textFontStyle", JHSecondaryViewer.class);
         back[14] = new PropertyDescriptor("textColor", JHSecondaryViewer.class);
         return back;
      } catch (Exception ex) {
         return null;
      }
   }
   }
```

5.6.1.5 Parameter Names

When naming parameters, be sure to avoid names reserved in the HTML 4.0 specification for use as <`OBJECT`> tag attributes. For a complete list of <`OBJECT`> attributes see the HTML 4.0 specification:

```
http://w3c.org/TR/REC-html40/
```

5.6.2 Using the <OBJECT> Tag

You add lightweight components to JavaHelp topics by means of the <`OBJECT`> tag and its `classid` attribute. The help viewer only recognizes `classid` values prefixed with the "`java:`" tag. All other `classid` tags are ignored. The following example creates an `ALabel` within the HTML topic:

```
<OBJECT CLASSID="java:sunw.demo.object.ALabel"</OBJECT>
```

You can use standard <`OBJECT`> tag attributes (see the HTML 4.0 specification for more details), but to be recognized the lightweight component must have getter and setter methods for those attributes. A getter or setter method must operate on a `String`. For example, in the following example width and height for the `ALabel` are set if there are `getWidth`/`setWidth` and `getHeight`/`setHeight` methods in `ALabel`:

```
<OBJECT
    CLASSID="java:sunw.demo.object.ALabel"
    width="400" height="500">
</OBJECT>
```

Parameters are passed to lightweight components by using the `<param>` tag. A parameter is only recognized if the component has getter and setter methods for that parameter. A getter or setter method must operate on a `String`. In the example below, the help viewer passes a number of parameters and their values to a the `JHSecondaryViewer` component:

```
    <OBJECT classid="java:com.sun.java.help.impl.JHSecondaryViewer">
        <param name="content" value="../topicB/glossary_def.html">
        <param name="viewerActivator" value="javax.help.LinkLabel">
        <param name="viewerStyle" value="javax.help.Popup">
        <param name="viewerSize" value="300,400">
        <param name="text" value="Click here">
        <param name="textFontFamily" value="SansSerif">
        <param name="textFontSize" value="x-large">
        <param name="textFontWeight" value="plain">
        <param name="textFontStyle" value="italic">
        <param name="textColor" value="red">
    </OBJECT>
```

5.6.3 Supplemental Information

The following information supplements the information in this topic.

Lightweight Java Components

For general information about lightweight Java components see:

`http://java.sun.com/j2se/1.4.1/docs/guide/awt/demos/lightweight/index.html`

JavaHelp Components

As a reference, the sources to the lightweight components that implement JavaHelp system popups and secondary windows (`JHSecondaryViewer.java` and `JHSecondaryViewerBeanInfo.java`) can be found in `src.jar` at:

```
com\sun\java\javahelp\impl
```

For a description of how the `<OBJECT>` tag is used to implement popups and secondary windows, see Using Popup and Secondary Windows.

HTML 4.0 Specification

You can find a detailed description of the `<OBJECT>` tag as part of the HTML 4.0 specification at:

`http://w3c.org/TR/REC-html40/`

See also:

Programming with the JavaHelp System
Adding the JavaHelp System to Applications
Accessing a URL Through a Firewall
Implementing Context−Sensitive Help
Merging Helpsets Dynamically
Embedding JavaHelp Components
Server Based JavaHelp

5.7 Server−Based JavaHelp Helpsets

Server−based applications have the same need for online help as client based applications, but they require that the helpset runs in a web browser, as the applications do, and that it be accessed from a server. Version 1.0 of the JavaHelp software API provided a foundation for developing online help for server−based applications. However, the specification did not define standards for a JavaHelp bean or for a Java Server Pages™ (JSP) tag library to access helpset data. Version 2 of the JavaHelp software does define these standards and provides a tag library for server−based applications.

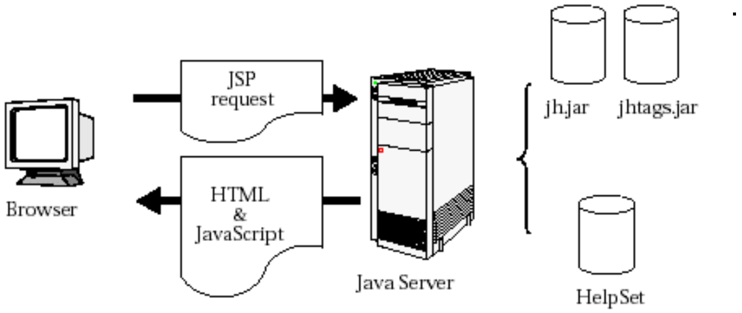
5.7.1 Java Server Pages

JSP enables web developers to develop dynamic web pages. JSP uses XML−like tags to encapsulate the logic that generates web content. JSP pages separate the page logic from its design and display, which prevents the overlapping of roles between web designers and programmers. Designers design the web pages and programmers add the logic and code to them.

For more information and tutorials on JavaServer Pages technology, see http://java.sun.com/products/jsp/docs.html.

5.7.2 Server−Based JavaHelp Architecture

By combining the JavaHelp software API with new JavaHelp JSP tag libraries, web developers are now able to provide help for server−based applications. The diagram below illustrates the architecture.



A browser initiates a JSP request. Examples of a JSP request are displaying the help content in the helpset, the navigators, or the data for a given navigator. Typically, the JSP request contains JavaBeans™ components as well as JSP tag extensions. The Java™ server turns the request into a Java Servlet. The servlet access the appropriate information from the helpset by using the classes in the JavaHelp library (`jh.jar`) and the JavaHelp tag library (`jhtags.jar`) and returns HTML and possibly JavaScript or dynamic HTML (DHTML) to the browser.

5.7.3 JavaHelp Server Components

Access to helpset data on a server is accomplished through a combination of JavaBeans components specific to the JavaHelp system and JSP tag extensions. This section defines the standard JavaHelp JavaBeans and JSP tag extensions and scripting variables.

5.7.3.1 JavaHelp Server Bean

`ServletHelpBroker` is the JavaBean component that stores help state information, such as the helpset in use, the current ID, the current navigator and other pieces of help information. While it implements the `javax.help.HelpBroker` interface, some of the methods are either not implemented or throw `UnsupportedOperationExceptions` if called. The `javax.help.HelpBroker` methods that are not implemented in this component are listed below:

Method	**Result**
`initPresentation()`	No Operation
`setDisplayed(boolean)`	Ignored
`boolean isDisplayed()`	Always returns true

`enableHelpKey(`
`Component,`
`String id, HelpSet)`
	No Operation

`enableHelp(`
`Component|MenuItem,`
`String id, HelpSet)`
	No Operation

`enableHelpOnButton(`
`Component| MenuItem,`
`String id, HelpSet)`
	No Operation

One new method is added to `ServletHelpBroker`:

Method	**Result**

`NavigatorView`
`getCurrentNavigatorView()`
	Returns the current navigator as a `NavigatorView`.

5.7.3.2 Using `ServletHelpBroker`

The `ServletHelpBroker` is used in the JSP request with a session scope. With this scope, the help broker remains in existence for the duration of a session. The following code defines the help broker:

```
<jsp:useBean id="helpBroker" class="ServletHelpBroker" scope="session" />
```

The `ServletHelpBroker` methods can be called in two ways:

- In tag libraries:

  ```
  <jh:validate helpBroker="<%= helpBroker %>" />
  ```

- Directly in the JSP:

  ```
  <FRAME SRC=
  "<jsp:getProperty name="helpBroker" property="currentURL" />"
  NAME="contentsFrame" SCROLLING="AUTO">
  ```

5.7.4 JavaHelp JSP Tag Extensions

While you could retrieve all the helpset information required for displaying online help or documentation by using JavaBeans components and JSP scriptlets, you can instead avoid the appearance of programming and use a standard set of tag extensions in the JavaHelp tag library to invoke application functionality. The JavaHelp tag library is a common set of building blocks that perform the following functions:

- Concealing the complexity of access to helpset data

- Introducing new scripting variables into a page
- Handling iterations without the need for scriptlets

The JavaHelp JSP tags are defined below:

Tag	**Tag Class** **TEI Class**	**Description**	**Attributes**
`validate`	`ValidateTag`	Validates a `HelpBroker` with various parameters. Enables easy setup of a help broker with a new helpset. Also enables merging of helpsets and setting the current ID.	*`helpbroker`* *required* `HelpBroker` object *`setInvalidURL`* *not required* String representing the URL for `InvalidHelpSet` message. *`helpSetName`* *not required* String representing the URL for `the helpset name.` *`currentID`* *not required* String id of desired currentID. *`merge`* *not required* Boolean value. If true then merge helpset into current helpset if one exists. Otherwise do not merge helpset.
`navigators`	NavigatorsTag NavigatorsTEI	Returns `NavigatorView` information for a given `HelpBroker`.	*`helpbroker`* *required* `HelpBroker` object *`currentNav`* *not required* String name of the current navigator`.`
`tocItem`	`TOCItemTag` `TOCItemTEI`	Provided with a `TOCView`, returns `TOCItem` information.	*`tocView`* *required* `TOCView` object *`helpbroker`* *required* `HelpBroker` object *`baseID`* *not required* Determined by expression. String text for the base identification of the `TOCItem`.
`indexItem`	`IndexItemTag` `IndexItemTEI`	Provided with an `IndexView`, returns `IndexItem` information.	*`IndexView`* *required* Determined by expression. `IndexView` object. *`helpbroker`* *required* `HelpBroker` object *`baseID`* *not required* Determined by expression. String text for the base identification of the `IndexItem`.
`searchItem`	`SearchItemTag` `SearchItemTEI`	Provided with a `SearchView`, returns `SearchItem` information.	*`SearchView`*

			required Determined by expression. `SearchView` object. *`helpbroker`* *required* `HelpBroker` object *`baseID`* *not required* Determined by expression. String text for the base identification of the `SearchItem`.

Unless otherwise specified, all attribute values are determined by expression. Also, with the exception of the `validate` tag, the body of all tags are JSP.

5.7.4.1 Using the `validate` Tag

The `validate` tag is designed to be used once in a JSP, as shown below:

```
<jh:validate helpBroker="<%= helpBroker %>" />
```

The preceding code verifies that a valid `HelpBroker` exists and then loads the helpset that has been defined either in the `validate` tag with the `helpSetName` attribute or as an `HTTP POST` request.

5.7.5 Navigator Scripting Variables

The `navigator`, `tocItem`, `indexItem`, and `searchItem` tag extensions introduce a predefined set of scripting variables into a page. These variables enable the calling JSP to control the presentation without having to perform processing to determine the content. Unless otherwise specified, each scripting variable creates a new variable, and the scope is set to `NESTED`. `NESTED` variables are available to the calling JSP only within the body of the defining tag.

5.7.5.1 Navigator Variables

The navigator variables are defined in the table below.

Variable	**Data Type**	**Description**
`classname`	`java.lang.String`	Name of the `NavigatorView` class.
`name`	`java.lang.String`	Name of the view as defined in the helpset.
`tip`	`java.lang.String`	Tooltip text for the view.
`iconURL`	`java.lang.String`	URL for the icon if set with the `imageID` attribute in the helpset.

5.7.5.2 Using the Navigator Variables

The `navigator` tag is used to return information about the current navigator. In the illustration below the `navigator` tag is used to determine the navigators that are used in the helpset and sets an HTML `<img>` tag based on the navigator name.

```
<jh:navigators helpBroker="<%= helpBroker %>" >
   <A HREF="navigator.jsp?nav=<%= name %>">
   <IMG src="<%= iconURL!=""?
        iconURL : "images/" + className +".gif" %>"
        Alt="<%= tip %>"
        BORDER=0></A>
</jh:navigators>
```

5.7.5.3 `tocItem` Variables

The tocItem variables are defined in the table below.

Variable	**Data Type**	**Description**
`name`	`java.lang.String`	`tocItem` text as defined in the `name` attribute.
`target`	`java.lang.String`	`tocItem` target as defined in the `target` attribute.
`parent`	`java.lang.String`	Hex value identifying the parent node.
`parentID`	`java.lang.String`	String identifying the parent node.
`node`	`java.lang.String`	Hex value identifying this node.
`nodeID`	`java.lang.String`	String identifying this node.
`iconURL`	`java.lang.String`	URL for the icon if set with the `imageID` attribute in the `tocItem`.
`contentURL`	`java.lang.String`	URL for the content represented by this item.
`isCurrentNav`	`java.lang.Boolean`	True if current navigator, false if not.

5.7.5.4 Using `tocItem`

The `tocItem` tag returns information about the TOC items defined in a `TOCView`. In the sample code below, the `TOCView` returns `tocItem` scripting variables that are added to the JavaScript tag `tocTree.addTreeNode`.

```
tocTree = new Tree("tocTree", 22, "ccccff", true, false);
<% TOCView curNav = (TOCView)helpBroker.getCurrentNavigatorView(); %>
<jh:tocItem helpBroker="<%= helpBroker %>" tocView="<%= curNav %>" >
   tocTree.addTreeNode("<%= parentID %>",
                       "<%= nodeID %>",
                       "<%= iconURL!=""?iconURL:"null" %>",
                       "<%= name %>","<%= helpID %>",
                       "<%= contentURL!=""?contentURL:"null" %>",
                       "<%= expansionType%>" );
</jh:tocItem>
tocTree.drawTree();
tocTree.refreshTree();
  <%
  ID id = helpBroker.getCurrentID();
  if (id != null) {
  %>
     tocTree.selectFromHelpID("<%= id.id%>");
  <%
     }
  %>
```

5.7.5.5 `indexItem` Variables

The `indexItem` variables are defined in the table below.

Variable	Data Type	Description
`name`	`java.lang.String`	`indexItem` text as defined in the name attribute.
`target`	`java.lang.String`	`indexItem` target as defined in the `target` attribute.
`parent`	`java.lang.String`	Hex value identifying the parent node.
`parentID`	`java.lang.String`	String identifying the parent node.
`node`	`java.lang.String`	Hex value identifying this node.
`nodeID`	`java.lang.String`	String identifying this node.
`iconURL`	`java.lang.String`	URL for the icon if set with the `imageID` attribute in the `indexItem`.
`contentURL`	`java.lang.String`	URL for the content represented by this item.

5.7.5.6 Using `indexItem`

The `indexItem` tag returns information about the index item defined in an `IndexView`. In the sample code below, the `IndexView` returns `indexItem` scripting variables that are added to the JavaScript tag `addNode`.

```
indexTree = new Tree("indexTree", 22, "ccccff", false, true);
<% IndexView curNav = (IndexView)helpBroker.getCurrentNavigatorView(); %>
<jh:indexItem indexView="<%= curNav %>" helpBroker="<%= helpBroker %>" >
    indexTree.addTreeNode("<%= parentID %>",
                          "<%= nodeID %>", "null",
                          "<%= name %>","<%= helpID %>",
                          "<%= contentURL!=""?contentURL:"null" %>",
                          "<%= expansionType%>");
</jh:indexItem>
indexTree.drawTree();
indexTree.refreshTree();
<%
  ID id = helpBroker.getCurrentID();
  if (id != null) {
  %>
     indexTree.selectFromHelpID("<%= id.id%>");
  <%
     }
  %>
```

5.7.5.7 `searchItem` Variables

The `SearchItem` variables are defined in the table below.

Variable	Data Type	Description
`name`	`java.lang.String`	Unique name of the `searchItem`.
`helpID`	`java.lang.String`	String ID associated with this `searchItem`.
`confidence`	`java.lang.String`	The quality of the hits as returned by the search engine.
`hits`	`java.lang.String`	Number of hits.
`contentURL`	`java.lang.String`	URL for the content represented by this item.

hitBoundries	java.lang.String	A list of boundaries. Returns in the format of {begin, end},...

5.7.5.8 Using searchItem

The searchItem tag returns information about the search items defined in a SearchView. In the sample code below, the SearchView returns searchItem scripting variables that are added to the JavaScript tag addNode.

```
searchList = new SearchList("searchList", 22, "ccccff");
<jh:searchTOCItem searchView="<%= curNav %>"
                  helpBroker="<%=    helpBroker %>"
                  query="<%= query %>" >
   searchList.addNode("<%= name %>",
                      "<%= confidence %>",
                      "<%= hits %>",
                      "<%= helpID %>",
                      "<%= contentURL %>" );
</jh:searchTOCItem>
searchList.drawList();
searchList.refreshList();
searchList.select(0);
```

See also:

Programming with the JavaHelp System
Adding the JavaHelp System to Applications
Accessing a URL Through a Firewall
Merging Helpsets Dynamically
Implementing Context−Sensitive Help
Embedding JavaHelp Components
Creating Lightweight Java Components

6 Localizing Help Information

This chapter contains information useful to localizers of JavaHelp systems. The following topics describe how to localize the various components of the JavaHelp system:

Localizing the Help Presentation

Describes how the presentation aspects of the JavaHelp system (primarily the help viewer) are localized.

Localizing Helpsets

Describes how to localize helpsets.

Localizing XML Data

Describes how to localize XML−based metadata files.

Localizing HTML Data

Describes how to localize HTML−base topic files.

Localization and Fonts

Describes the interaction of fonts with the localization of help information.

Localizing the Full−Text Search Database

Describes how to create localized full−text search databases.

6.1 Localizing the Help Presentation

The JavaHelp system viewer generally inherits the locale from the application. For information about how applications are internationalized, see the internationalization section of *The Java Tutorial* at:

`http://java.sun.com/docs/books/tutorial/i18n/index.html`

The culturally dependent data (for example, messages and labels) for the presentation components is contained in the property file named `javahelp.properties` in the `javax.help.resources` package.

If the locale of the help viewer is different from the locale of the application, the locale of the HelpBroker can be set using the `HelpBroker.setLocale()` method. Setting the locale of the HelpBroker sets the locale for all subordinate components. If you do not use the HelpBroker, set the locale of the JHelp* components directly using their `setLocal()` methods.

6.1.1 Data Input in the Viewer

There are two places in the JavaHelp system GUI where culturally dependent input is required: Index Find and Search Query.

In both cases, platform−specific input methods are used. Once text is entered in the Index Find or Search Query text boxes, additional locale−based processing is activated (usually by pressing the Enter key).

In the Index Find case, the input text is searched for within the index entries using locale−based comparisons. The locale used in Index Find is the locale of the Index navigator − usually the locale of the application, unless overridden using the `JHelpIndexNavigator.setLocale()` method.

In the Search Query case, the input text and the locale of the Search navigator are passed to a `HelpSearch` class. The `HelpSearch` class tokenizes the query text into words using the locale−specific tokenizer. The locale used in Search Query is the locale of the Search navigator − usually the locale of the application, unless overridden using the `HelpBroker.setLocale` or `JHelpSearchNavigator.setLocale` methods.

See also:

Localizing Help Information
Localizing Helpsets
Localizing XML Data
Localizing HTML Data
Localization and Fonts
Localizing the Full−Text Search Database

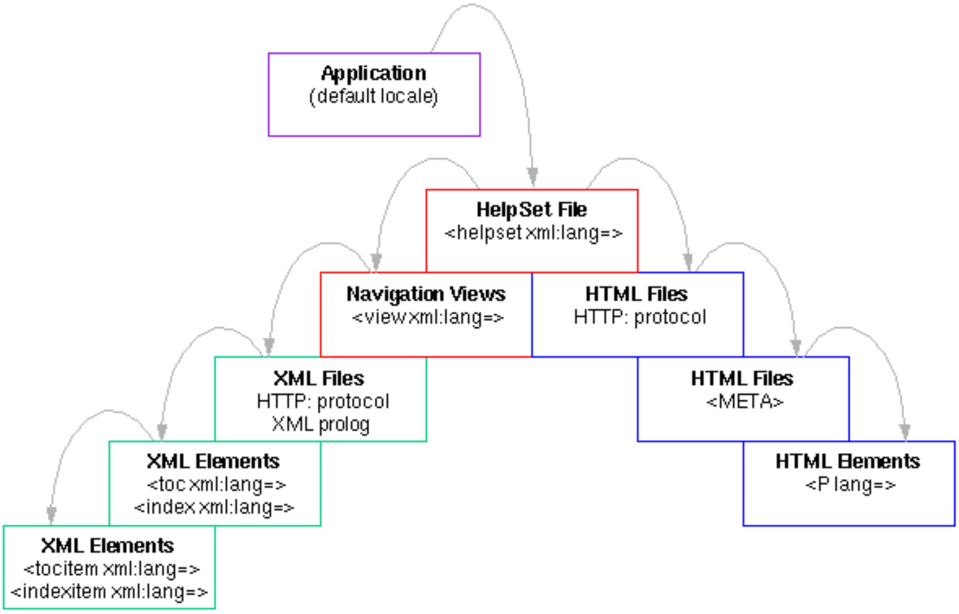
6.2 Localizing Helpsets

The portal to all JavaHelp system help information is the helpset file which defines the *helpset*. The helpset is the set of data that constitutes your help system and includes:

- Helpset file (XML)
- Map file (XML)
- TOC definition file (XML)
- Index definition file (XML)
- Topic files (HTML)
- Full−text search database

All helpset data can be localized, often in multiple ways. The process of localizing helpsets can be viewed as a *cascading* process, where each level of the cascade becomes more specific and takes precedence over the levels above it.

The following diagram shows the different levels in the JavaHelp system where the locale can be set and localization can occur, starting with the host application and moving into the helpset. Changes to locale are propagated down the hierarchy, with a change at each level overriding the locale set above it.



Legend:

🟪 Described in Localizing Help Presentation

🟥 Described in the following sections

🟩 Described in Localizing XML Data

🟦 Described in Localizing HTML Data

6.2.1 The Helpset File

The locale of a helpset is usually set through the helpset file. The locale for the entire helpset can be specified through the helpset file, although portions of it can be be selectively overridden in the data files.

6.2.1.1 Finding the Helpset File

When the application activates the JavaHelp system, the application uses the `HelpSet.findHelpSet` method to find the correct helpset file and return its location (URL). The full name of the helpset file is constructed based on the *name* of the helpset file specified as an argument to `HelpSet.findHelpSet`, and the *locale* based on either the system default locale or a locale specified as an optional argument.

The name of the locale−specific helpset file is constructed and then searched for in the following order (from most to least specific):

1. *name_language_country_variant*`.hs`
2. *name_language_country*`.hs`
3. *name_language*`.hs`
4. *name*`.hs`
5. *name_defaultlanguage_defaultcountry_defaultvariant*`.hs`
6. *name_defaultlanguage_defaultcountry*`.hs`
7. *name_defaultlanguage*`.hs`

The defaults are derived from the system with the `Locale.getDefault` method.

6.2.1.2 Setting Locales in the Helpset File

The helpset file can be used to control the locale of different aspects of the help system. The XML language controls used to set locale are discussed in more detail in Localizing XML Data.

The `xml:lang` attribute can be used within the `<helpset>` tag to specify the locale of the entire helpset (the other elements in the helpset file automatically inherit the locale). For example:

```
<helpset xml:lang="fr">
```

The locale specified for the helpset in this manner overrides any locale acquired from the system or the application. For this reason, it is the most reliable means for setting the helpset locale.

The locale of the `<title>` element is always the same as locale of the helpset. Any `xml:lang` attributes specified for the `<title>` element are ignored.

6.2.1.3 Navigation View Locale

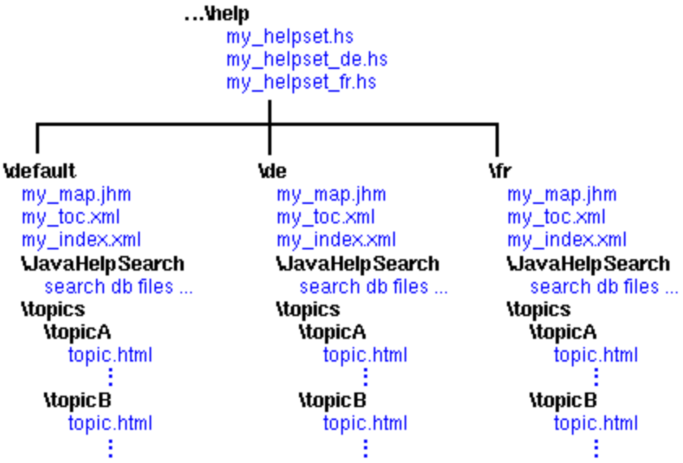
The `xml:lang` attribute can also be used to change the locale of the navigator views specified in the `<view>` elements (for example, the TOC and index). Note, however, that this locale is overridden by any locale settings specified by `xml:lang` attributes in the TOC and index XML definition files, as described in XML Data.

The locale of the `<label>` element is always the same as the locale of the containing view. Any `xml:lang` attributes specified for `<label>` elements are ignored.

6.2.1.4 Shipping Multiple Locales

The JavaHelp software makes it possible to simultaneously distribute multiple localized helpsets (for example, German, French, and English). As described above, the `HelpSet.findHelpSet` method determines the correct helpset file based on the system's locale or as set by the application using `HelpBroker.setLocale()`. You can include multiple, localized helpset files and locate the appropriate version using this naming convention.

If you ship multiple locales, you will probably organize your help information a little differently than is described in Setting Up a JavaHelp System. The following diagram shows one way you can organize the help information by locale:



Note that the paths specified in the `<data>` sections of the localized helpset files must point to the appropriate locations. For example:

```
<maps>
       <mapref> location="de/Map.jhm" />
    </maps>
    <view>
      <name>TOC</name>
      <label>Holidays</label>
      <type>javax.help.TOCView</type>
      <data>de/HolidayTOC.xml</data>
    </view>
```

6.2.1.5 Merging Localized Helpsets

JavaHelp system helpsets can be merged. The locale of a helpset is maintained in a merge operation. For instance, if the master helpset (locale `en_US`) is merged with another Helpset (locale `fr_FR`), the locale of both helpsets is maintained.

6.2.1.6 Map Data

Map data should not be localized. If IDs (`target` attribute) are localized they will no longer match the IDs used internally in the application.

See also:

Localizing Help Information
Localizing Help Presentation
Localizing XML Data
Localizing HTML Data
Localization and Fonts
Localizing the Full−Text Search Database
Helpset File
Table of Contents File
Index File

6.3 Localizing XML Data

The XML data that defines the TOC, index, and helpset files can be localized as specified in the XML 1.0 specification (`http://w3c.org/XML/`). Both the character encoding and language can be set for these files.

6.3.1 Character Encoding

Character encoding is an unambiguous mapping of the members of a character set (letters, ideographs, digits, symbols, or control functions) to specific numeric code values. The specified encoding applies to the entire file. Character encoding can be set for XML files using the following methods (listed in order of precedence):

- The HTTP protocol
- The XML prolog declaration

Only one encoding can be specified for any file.

6.3.1.1 HTTP Protocol

If the XML file is provided by a server via the HTTP protocol, the server can specify the character set using the `charset` parameter in the HTTP `Content−Type` field.

6.3.1.2 XML Prolog Declaration

Typically, the encoding attribute in the prolog to all of the XML metadata files is used to specify the encoding used for its character set. For example, the following prolog specifies the Latin−1 (ISO−8859−1) character set:

```
<?xml version='1.0' encoding='ISO−8859−1' standalone='yes' ?>
```

6.3.2 Setting the Language

The language can be set for the XML files using the following methods (listed in order of precedence):

- The `xml:lang` attribute, which can be set for any XML element (tag)
- By inheritance from the closest parent element (tag)
- The HTTP protocol `Content−Language` header
- The default locale of the helpset
- The default locale of the application

It is possible to mix languages in these files. A different language can be specified for each tag; however, only one character encoding can be specified for each file.

6.3.2.1 The `xml:lang` Attribute

The language for any element (tag) in XML files can be set using the `xml:lang` attribute. For example, the following code sets the language for that table of contents entry to German. Any elements (`<tocitem>` tags) nested in that tag automatically inherit that language:

```
<tocitem xml:lang="de" target="jde.intro">Homepage der JDE Online−Hilfe</tocitem>
```

Typically, the `xml:lang` attribute is set in the opening tag (for example, `<toc xml:lang="de">`), so all of the other elements in the TOC inherit the attribute. In this case the entire TOC is in German.

The syntax of the lang attribute is:

lang	= language−code
language−code	= primarycode ('−' subcode)
primarycode	= ISO639 | IonaCode | UserCode
ISO639	= 2 alpha characters
IonaCode	= (i | I) '−' (alpha characters)
UserCode	= (x | X) '−' (alpha characters)
subcode	= (alpha characters)

For more information about the `lang` attribute, please refer to the XML recommendation at the World Wide Web Consortium web site (`http://w3c.org/XML/`).

6.3.2.2 HTTP Protocol

If the XML file is provided by a server via the HTTP protocol, the server can specify the language for that file using the HTTP `Content−Language` header (for example, `Content−Language:en−US`).

See also:

Localizing Help Information
Localizing Help Presentation
Localizing Helpsets
Localizing HTML Data
Localization and Fonts
Localizing the Full−Text Search Database

6.4 Localizing HTML Data

The HTML data contained in topic files can be localized as specified in the HTML 4.0 specification (`http://w3c.org/TR/REC−html40/`). Both the character encoding and the language can be set.

6.4.1 Character Encoding

Character encoding is an unambiguous mapping of the members of a character set (letters, ideographs, digits, symbols, or control functions) to specific numeric code values. Character encoding can be set for HTML files in the following ways (listed in order of precedence):

- HTTP protocol
- HTML `<META>` declaration
- HTML `charset` attribute on an external source (not recognized by the JavaHelp system)

Only one encoding can be specified for any file.

6.4.1.1 HTTP Protocol

If the HTML file is provided by a server via the HTTP protocol, the server can specify the character set using the `charset` parameter in the HTTP `Content-Type` field.

6.4.1.2 `<META>` Declaration

The HTML `<META>` declaration can be used to specify the character encoding. Encoding is specified using the `charset` parameter, as follows:

```
<META HTTP-EQUIV="Content-Type" CONTENT="text/html;charset=x-euc-jp">
```

6.4.2 Specifying a Language

The language can be set in HTML files in the following ways (listed in order of precedence):

- The `lang` attribute
- Inheritance from the closest element (tag)
- The `<META>` declaration
- The HTTP protocol `Content-Language` header
- The default locale of the helpset
- The default locale of the application

6.4.2.1 The HTML `lang` Attribute

The `lang` attribute specifies the language of a specific element (tag>. It can be applied to every HTML element except the following: `<APPLET>`, `<BASE>`, `<BASEFONT>`, `<BR>`, `<FRAME>`, `<FRAMESET>`, `<HR>`, `<IFRAME>`, `<PARAM>`, and `<SCRIPT>`.

The following is an example of the `lang` attribute being used with the `<P>` tag:

```
    <P lang="en-US">
```

Any elements (tags) nested within a tag automatically inherit the parent tag's language.

The syntax of the lang attribute is:

lang	= language−code
language−code	= primarycode ('−' subcode)
primarycode	= ISO639 | IonaCode | UserCode
ISO639	= 2 alpha characters
IonaCode	= (i | I) '−' (alpha characters)
UserCode	= (x | X) '−' (alpha characters)
subcode	= (alpha characters)

For more information about the `lang` attribute, please refer to the HTML 4.0 specification at the World Wide Web Consortium web site (`http://w3c.org/TR/REC−html40/`).

6.4.2.2 <META> Declaration

The HTML <`META`> declaration can be used to specify the file's language. Language is specified using the `Content−Language` parameter:

```
<META http−equiv="Content−Language" content="en−US">
```

6.4.2.3 HTTP Protocol

If the HTML file is provided by a server via the HTTP protocol, the server can specify the language for that file using the HTTP `Content−Language` header (for example, `Content−Language:en−US`).

See also:

Localizing Help Information
Localizing Help Presentation
Localizing Helpsets
Localizing XML Data
Localization and Fonts
Localizing the Full−Text Search Database

6.5 Localization and Fonts

The JavaHelp system displays information using the host's default fonts. If a helpset contains information that cannot be presented using the default fonts, an alternate font glyph (usually a square) is displayed in its place.

The JavaHelp presentation font can be changed either by modifying the `font.properties` file in the JRE or by setting the font in the HelpBroker or JHelp* components.

The `HelpBroker.setFont(Font f)` method sets the font for a JavaHelp presentation and propagates the font to all of the presentation components. JHelp* components can set their fonts using the `setFont()` method.

For more information about Unicode font support and adding fonts to the JRE, please refer to the *Fonts* section in the following documents:

- (JDK 1.1)

  `http://java.sun.com/products/jdk/1.1/docs/guide/intl/`
- (Java 2 Platform)

  `http://java.sun.com/products/`
  `/1.2/docs/guide/internat/`

See also:

Localizing Help Information
Localizing Help Presentation
Localizing Helpsets
Localizing XML Data
Localizing HTML Data
Localizing the Full−Text Search Database

6.6 Localizing the Full−Text Search Database

The JavaHelp system full−text search database is constructed using the `jhindexer` command. The `jhindexer` command parses HTML topic files according to each file's character encoding. If the document format (HTML, GIF, text) supports a language attribute, the document text is tokenized according to the language attributes for its elements.

By default, `jhindexer` assumes the default locale − use the `locale` option to specify the locale directly to the `jhindexer` command. The syntax for the locale option is:

```
jhindexer −locale language_country_variant
```

The argument to the option is the name of the locale as described in `java.util.Locale`, for example, `en_US` (English, United States) or `en_US_WIN` (English, United States, Windows variant).

See also:

Localizing Help Information
Localizing Help Presentation
Localizing Helpsets
Localizing XML Data
Localizing HTML Data
Localization and Fonts
Creating the Full−text Search Database
The `jhindexer` Command

7 Index