JavaHelp™ System User's Guide
JavaHelp 1.1.3 - June 2002

This user's guide contains the following chapters:

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 [IMAGE]	A guide for localizers of JavaHelp systems.
Merging HelpSets	Information about how to statically and dynamically merge HelpSets.

[IMAGE]
[IMAGE] This guide is available in PDF format at:

```
doc\jhug.pdf
```

[IMAGE] The JavaHelp HelpSet for this guide can be found at:

```
doc\jhug
```

The JavaHelp 1.1.3 Release

This chapter describes the contents of the JavaHelp 1.1.3 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:

[IMAGE]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.
- Review the specification and the APIs.

[IMAGE]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.
- Examine the example HelpSets.
- Create a HelpSet and view it using the `hsviewer` command.

Contents of the Release

The JavaHelp Zip file contains the following:

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`)
API	`javadoc` generated documentation of the JavaHelp API. Can be viewed 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`)
Example 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 Definition) 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 JDK 1.2.2 viewer is included in `doc\css`.

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

Requirements

The following describes what is required to use the JavaHelp 1.1.3 release.

JDK

The JavaHelp 1.1.3 release is an optional package of both JDK1.1 and the Java^{TM} 2 SDK, Standard Edition. As a result, it works well with both versions.

[IMAGE]
Note that the demo programs in the demos\bin directory only work with the Java 2 platforms.

Java 2 SDK Features

Java 2 SDK contains some functionality not available on JDK1.1. As a result, the following JavaHelp system features are available only with the Java 2 SDK:

- You can run the JavaHelp demo programs
- The jar: protocol provides a general and consistent way to refer to files within a JAR file.
- The ability to create temporary files improves full-text search performance.
- A more flexible ClassLoader (including standard URLClassLoader) enables the easy creation of new ClassLoaders.
- An improved security model.
- Access to the SystemEventQueue from an applet (when permitted by the security manager) enables some context-sensitive help features in applets.
- Improved I18N support, including input methods.
- 2D graphics support.

Tested Java Platforms

The JavaHelp 1.1.3 release should work with any compliant version of JDK1.1 or the Java 2 SDK. We have tested this release against JDK1.1.8 and the Java 2 SDKs (JDK 1.2.2, 1.3.x, 1.4) on the following systems:

- Windows 95
- Windows NT4.0
- Windows 98
- Solaris 2.5.1/SPARC
- Solaris 2.6/SPARC
- Solaris 7/SPARC
- Solaris 8/SPARC

Changes Since the 1.1.2 Release

This page describes the most significant changes since the 1.1.2 release.

Installation Packages

The JavaHelp system is now released in a single Zip file installation package for all platforms. Prior releases should be uninstalled before installing V1.1.3.

New jhindexer and jhsearch scripts

New executable scripts for jhindexer and jhsearch are included in the Zip file. These scripts are intended to work on all supported platforms.

Demo programs

Demo programs are now in the form of executable JAR files. See Demonstration Programs for details.

Consolidation of Demo HelpSets

Demo helpsets are no longer duplicated in both `demos\hs` and `demos\hsjar`.

Fonts not Set Correctly in Index and Search Navigators

Fonts were not set properly in the Index and Search Navigators. This caused the Search Navigator to display titles incorrectly in certain locales.

JavaHelp System User's Guide **Updated**

The *JavaHelp System User's Guide* (this document) has been reformatted using a stylesheet:

```
doc/jhug/jhug.css.
```

Discussion Group (JAVAHELP-INTEREST)

Sun now 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, send mail to:

```
listserv@javasoft.com
```

In the body of the message type:

```
SUBSCRIBE JAVAHELP-INTEREST
```

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

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

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.

IDE demo	A mock-up 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.
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: `demos\src`

[IMAGE]

- The demo program JAR files are located in the **demos\bin** folder of this release.
- In addition to the command-line instructions below, on Microsoft Windows sytems you can run these programs by double-clicking on their icons.

IDE Demo

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

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.
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 on the [IMAGE] button, then move the field-level cursor [IMAGE] 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. This activates the help viewer and loads 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. This activates the custom *ClassViewer* navigator described above, but embeds it within the application. Note that as you navigate, the corresponding topic is displayed in the lower pane of the application.

[IMAGE] To run the IDE demo:

```
C:\> java -jar idedemo.jar
```

Once the demo starts, choose Help -> Demo JDE - Help to activate the help viewer.

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.

[IMAGE] To run the Object demo:

```
C:\> java -jar object.jar
```

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 2 SDK). The API doclet is not included in this release of the JavaHelp system.

[IMAGE] To run the API Viewer demo:

```
C:\> java -jar apiviewer.jar
```

Merge Demo

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

[IMAGE] To run the Merge demo:

```
C:\> java -jar merge.jar
```

Browser Demo

Demonstrates how the JavaHelp system can be used with applet-based applications running in web browsers. The demonstration can be run using both Navigator 4 and Internet Explorer 4. The demo creates an applet button on an HTML page, when the button is clicked the JavaHelp viewer displays the holiday HelpSet. This demo requires some setup. Instructions for the demo can be found in

```
demos\browser\demo_instructions.
```

Search Example

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

Localized HelpSets

Localized demonstration HelpSets (English, German, Japanese) for the IDE demo are included in the `demos\hsjar` folder. Be sure to use the JavaHelp viewer Options->Set Fonts dialog box to choose the appropriate font.

[IMAGE]
The appropriate Unicode fonts must be installed to view the Japanese HelpSet.

Example 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.

JavaHelp System User's Guide

The HelpSet that comprises the *JavaHelp System User's Guide* (this document) can be found in:

```
doc\jhug
```

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:

```
demos\hsjar\holidays.jar
```

[IMAGE] To view this HelpSet:

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

IDE Demo

This is the HelpSet used in the IDE Demo. This HelpSet can be found in:

```
demos\hsjar\idehelp.jar
```

[IMAGE] To view this HelpSet:

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

Localized HelpSets

Two localized HelpSets (German and Japanese) are included in the release. For more information about localizing HelpSets, see Localizing Help Information.

[IMAGE] To view these HelpSets the correct fonts must be installed on your system.

German

This HelpSet is a portion of the *idedemo* HelpSet localized in German. This HelpSet can be found in:

```
demos\hsjar\idehelp_de.jar
```

[IMAGE] To view this HelpSet:

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

Japanese

This HelpSet is a portion of the *IDE demo* HelpSet localized in Japanese. To view the Japanese HelpSet, the correct Unicode fonts must be installed. This HelpSet can be found in:

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

[IMAGE] To view this HelpSet:

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

The JavaHelp Libraries and Tools

The JavaHelp libraries and tools are located in the `javahelp` folder of the release. All classes work with both JDK1.1 and the Java 2 Platform, Standard Edition and use Swing1.1.

Libraries

The JavaHelp classes are distributed in five different JAR files. The JAR files are located in the `javahelp\lib` folder.

jh.jar [IMAGE]	The standard library, it 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 may be useful for very simple help systems that do not require a full-text search database or help systems where size is critical.
jhall.jar	Contains all of 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.

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 [IMAGE]	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 created with the `jhindexer` command. The `jhsearch` command can be used to verify the validity of a search database without invoking the help viewer.

Limitations and Bugs

HTML Viewer

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

Images Distorted

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

You can avoid this problem by explicitly specifying "height" and "width" attributes with the <IMG> tag.

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, if you start `hsviewer` with the following command:

```
hsviewer -helpset C:\my_app\help\myhelpset.hs
```

The `hsviewer` command sets the classpath to be:

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

You cannot reference files above the `C:\my_app\help` folder. For example, in this case an image in `C:\my_app\images` referenced like this:

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

will not be displayed.

You can work around this problem by using the `demos\bin\hsviewer1_1` command. This command allows you to specify the classpath separately from the HelpSet file, that way you can set the classpath to include the folder that contains the image and specify the HelpSet file relative to that folder. For example:

```
hsviewer1_1 help\myhelpset.hs C:\my_app
```

Duplicate Lines Displayed (JDK1.2.2)

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), Swing1.1.1 (also part of JDK1.2.2) positions the lines incorrectly, this results in lines being displayed twice.

Anchor Targets

- Anchor targets (<A NAME="foo"></A>)are displayed in the center of the viewer display rather than at the top, as is the case in most browsers.
- If the TOC or index is used to access a topic file that contains anchor targets within 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 anchor within a tag. For example:

```
<H2>Working With Widgets<A NAME="widgets"></H2>
```

Cascading Style Sheets

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

**<P> and
 Tags**

In some cases (for example, within <UL> and <OL> lists), the <P> tag does not create the expected vertical space. In these cases, the <BR> tag does (incorrectly) create the expected vertical space.

The <SUP> <SUB> Tags (JDK1.2.2)

The <SUP> <SUB> tags are ignored on JDK 1.2.2 systems.

The Width Attribute of the <TD> Tag

The width attribute of the <TD> tag is ignored on JDK 1.2. The viewer assigns its own width to table columns.

On JDK 1.2.2, the width attribute works when specified in absolute pixels, the use of percentages is not supported.

Named Anchors in Ordered and Unordered Lists

If the first item after a list tag (<UL> <OL> <DL>) is a named anchor (<A NAME>), the list is rendered incorrectly:

This list is rendered incorrectly:

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

This one is rendered correctly:

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

TABS in <PRE> Tag not Recognized

TABS used in text within <PRE> tags are not recognized. Space characters are recognized correctly.

Viewer Cannot Load Image Files Directly

The help viewer aborts if you attempt to load a graphic file (*.gif, *.jpg) directly. The images must be included in an HTML file using the <IMG> tag.

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.

Null Pointer Exception (fixed in JDK1.2.2)

Due to a timing problem in the underlying Swing JEditorPane component, the following exception occurs occasionally:

```
Exception occurred during event dispatching:
java.lang.NullPointerException
   at javax.swing.text.View.preferenceChanged(View.java:121)
   at javax.swing.text.html.ImageView.run(ImageView.java:414)
```

This problem is benign.

Client-side Image Maps (fixed in JDK1.2.2)

The tags required to implement client-side image maps are not implemented (<MAP>, <IMG USEMAP>).

<TT> and <CODE> Tags (fixed in JDK1.2.2)

The <TT> and <CODE> tags do not change characters to a monospaced font. Also, other formatting tags (for example, <b>, <i>) located within these tags are not recognized.

<OL> Tag After Headings (fixed in JDK1.2.2)

If the <OL> tag is used directly after a heading, the first number in the list is rendered in the same font as the heading. You can work around this by inserting a tag (for example, <BR>) between the heading and the list.

Empty Entities (fixed in JDK1.2.2)

Using the <BR> tag to create empty entities such as:

```
<DD><BR>
<TD><BR></TD>
```

causes the following error:

```
Exceptions occurred during event dispatching:
javax.swing.text.StateInvariantError: infinite loop in formatting
```

Full-text Search

Parsing of Asian Languages

The JDK 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 that is 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 using the following process:

1. Double-click on words in the content pane
2. Copy/paste them into the search navigator Find field
3. Press Return

Match Limit

To enhance full-text search performance in this release, the search navigator only reports the 100 most relevant matches. For example, in the idedemo program, if you search for the word "build", you will notice 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 should not be a problem with more complex, multi-word, natural language queries.

NullPointerException in Search Navigator

A NullPointerException sometimes occurs when search results are displayed. This can be ignored, program execution proceeds normally.

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.

Context Sensitive Help

Field-level help ActionListener (JDK 1.1)

The field-level help ActionListener in the default implementation of the HelpBroker is disabled for JDK 1.1. JDK 1.1 doesn't allow the creation of custom cursors and doesn't support a predefined question-arrow cursor. Developers who want to support field-level help on JDK 1.1 can implement a field-level ActionListener that sets the cursor to one of the predefined cursors in java.awt.Cursor and then follow the steps described in the *JavaHelp System User's Guide* chapter "Implementing Context-Sensitive Help".

Field-Level Help Button

In the idedemo, if the field-level help button is clicked before the help viewer has been accessed (for example, Help->Demo JDE Help), a NullPointerException occurs.

F1 Help (Solaris OpenWindows)

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

Limitations on Running in Web Browsers

It is only possible to use the JavaHelp system in applets running in web browsers using JRE/JDK1.1.6 or later with the Java Plug-in.

JavaHelp does not run correctly in web browsers and the appletviewer due to a limitation in Swing 1.1. This limitation will be corrected in a future version of Swing. In the meantime, a description of how to work around this issue will be posted to the JavaHelp website.

Other Bugs

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

jar: Protocol

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

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.

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 accessibly 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 so that all the information can be displayed in a single scroll zone.

Fonts/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 JDK, 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.

Installation

If you uninstall the JavaHelp release while you are viewing any of the release directories (using Explorer, the MSDOS command prompt, or Solaris shell), the directories are not removed.

List of Files in the JavaHelp 1.1.3 Release

The libraries include in this release support both JDK1.1 and the Java^{TM} 2 Platform. They should run in any JDK1.1-compliant Java platform. Executable jar files are included for tools and demonstration programs. They can only be executed on the Java 2 Platform. They will not work with JDK1.1 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           - 1.2.2 JEditorPane default stylesheet
doc\spec                      - 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_1_0.dtd - HelpSet file DTD
javahelp\lib\dtd\index_1_0.dtd   - Index file DTD
javahelp\lib\dtd\map_1_0.dtd     - Map file DTD
javahelp\lib\dtd\toc_1_0.dtd     - TOC file DTD

javahelp\bin\                 - JavaHelp executable programs
javahelp\bin\jhindexer        - Creates the search database
javahelp\bin\jhsearch         - Queries the search database
javahelp\bin\jhwrap           - Auxiliary wrapper script

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\idedemo.jar         - Starts a mock-up of an IDE
demos\bin\hsviewer.jar        - HelpSet viewer
demos\bin\apiviewer.jar       - JavaHelp API viewer
demos\bin\merge.jar           - Demonstrates 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 (JDK1.2)
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\hsjar                   - Demo HelpSets in JARs
demos\hsjar\apidoc.jar        - api.hs HelpSet
demos\hsjar\holidays.jar      - HolidayHistory.hs HelpSet
demos\hsjar\idehelp.jar       - IdeDemo HelpSet
demos\hsjar\object.jar        - Object demo 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\src                     - Sources for the demos (JDK1.2)
demos\src\sunw\demo\browser     - Browser demo
demos\src\sunw\demo\classviewer - Classviewer Navigator used in apiviewer
```

```
demos\src\sunw\demo\idedemo      - Mock-up of an IDE using JavaHelp
demos\src\sunw\demo\searchdemo   - Alternate search engine
demos\src\sunw\demo\merge        - Example of how to merge HelpSets
demos\src\sunw\demo\object       - Object demo
```

Keeping in Touch

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

Feedback

Comments and questions about the JavaHelp system software are welcome. Please review the FAQ at our home page before sending email to:

```
javahelp-comments@eng.sun.com
```

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

Mailing List

The JavaHelp team maintains a mailing list for disseminating information about JavaHelp system updates and events. To subscribe, send mail to *listserv@javasoft.com*. In the body of the message type:

```
SUBSCRIBE JAVAHELP-INFO
```

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, send mail to:

```
listserv@javasoft.com
```

In the body of the message type:

```
SUBSCRIBE JAVAHELP-INTEREST
```

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

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

Web Site

Other information can be obtained at our web site:

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

We hope to hear from you!

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.

Introduction

Most interactive applications require online help -- Java applications are no exception. The JavaHelp 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 standard that is part of the Java platform (standard extension to the JDK^{TM}) relieves developers and authors of the task of inventing their own proprietary help systems.

The JavaHelp system consists of a fully featured, highly extensible specification and an implementation of that specification written entirely in the Java language. The implementation, based on the Java Foundation Classes (JFC), provides a very simple interface that allows application developers and authors to quickly and easily add online help to their applications. The specification enables developers to optionally 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

Authoring support for the JavaHelp system will be available through the major online help authoring tool vendors. Tool vendors, including Blue Sky Software, ForeFront, Quadralay, CreativeSoft, Hyperact, and Virtual Media, have publically announced that their products will provide authoring support for the JavaHelp system.

[IMAGE] **Next Overview Topic:** JavaHelp System Features

JavaHelp System Features

The following is an overview of the main features of the JavaHelp system.

Help Viewer

The standard JavaHelp system viewer consists of a toolbar and two panes:

Content pane	Displays help topics formatted using HTML 3.2, plus embedded lightweight Java components.
Navigation pane	A tabbed interface that allows users to switch between the table of contents, index, and full text search displays.

Help Viewer Window

Table of Contents

Collapsible/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) standards. The TOC display is synchronized with the content viewer--the topic being displayed is highlighted in the TOC.

Index

Supports merging of multiple indexes. The underlying file format follows World Wide Web Consortium (W3C) standards. The index display is synchronized with the content viewer--the topic being displayed is highlighted in the index.

Full-Text Search

The flexible full-text search engine can be used in a variety of network environments. Matches returned from searches are ranked for relevancy using "relaxation rules."

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.

Embeddable Help Windows

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

Context-Sensitive Help

Help can be activated from within programs using a number of different mechanisms.

Flexible Packaging

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

Customization

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

Merging

Help information from different sources can be combined and presented to the end user.

JavaBeans Support

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

[IMAGE] **Next Overview Topic:** Descriptive Scenarios

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.

[IMAGE] **Next Overview Topic:** JavaHelp System Lightweight Components

Invocation Mechanisms

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

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.

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.

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.

[IMAGE] **Next Overview Topic:** Presentation and Deployment

Presentation and Deployment

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.

Standalone Application

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

Stand Alone App

The application:

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

Network Application

The JavaHelp system provides the means 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.

Network App

The application:

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

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.

Embedded2

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.

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 may be shipped with its own help data. The following illustrates such a case.

Component Help

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 the JavaHelp 1.0 release, merging is accomplished by appending TOC and index information and searching merged full-text search databases.

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 may be wrapped in a library and be invisible to the application developer).

Help Server

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.

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 may have been delivered with the browser, or it may have been downloaded by the client into the CLASSPATH. The implementation may use the JavaHelp system content pane, or it may use the HTML viewer that is part of the web

browser.

<p align="center">Web Pages1</p>

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).

Applet(2)

In the second scenario, the JavaHelp system classes are downloaded along with the applet. Because the JavaHelp system is a Java "optional package", it is possible that a fully compliant Java^{TM} 2 SDK browser may 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 may choose to provide additional means for installing extensions downloaded through this mechanism.

<p align="center">Web Pages2</p>

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).

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.

<p align="center">Web Pages2</p>

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).

[IMAGE] **Next Overview Topic:** Full-text Search

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

Standalone

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

<div align="center">Standalone Search</div>

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.

<div align="center">Client-side Search</div>

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.

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.

<div align="center">Client-side Search</div>

This option also works well for applets. It permits developers to use alternate search engines (for example, Excite 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.

[IMAGE] **Next Overview Topic:** JavaHelp System Lightweight Components

JavaHelp System Lightweight Components

Functionality can be added to help topics using the lightweight component facility. Lightweight components are similar to Java applets, but are lighter-weight and load and execute more quickly.

The first release of the JavaHelp system includes a lightweight component that implement popup windows and secondary windows. For more information about popup windows and secondary windows see Using Popup and Secondary Windows.

Lightweight components can also be created by Java language developers to add functionality such as animation and multimedia to help topics. The Java^{TM} 2 Platform includes high quality audio and a video viewer that supports the common formats. For more information see Creating Lightweight Components.

Authoring Help Information

The topics in this chapter of the *JavaHelp System User's Guide* describe the aspects of the JavaHelp system of primary interest to online help authors. These topics assume that the author is responsible for creating the metadata files that JavaHelp 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*.

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

The following is a summary of the tasks required to create a HelpSet. All of these tasks are described in this chapter 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 `idedemo`. 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 `idedemo` (`demos\hs\idehelp`).
3. Create your own HelpSet:
    1. Create HTML topics using an editor of your choosing (be sure to use only HTML 3.2 tags).
    2. Copy the metadata files from `demos\hs\idehelp` to the folder that contains your topic files.
    3. Edit the metadata files to match your help information.
    4. Use the `hsviewer` command to display your HelpSet.

Viewing HelpSets

A HelpSet viewer is provided with the release to enable you to view HelpSets that you create. The viewer is only available on the Java™ 2 Platform and will not work with JDK/JRE1.1. Use the following command to execute the viewer:

```
java -jar demos/bin/hsviewer.jar
```

- The JDK/JRE that you use to run `hsviewer` does not have to be the one your application is deployed on. You can for example, use the Java 2 Platform to run the demo programs (including `hsviewer`) even if your application is deployed on JDK/JRE1.1.
- For a list of limitations, bugs, and "idiosyncrasies" that pertain to the JavaHelp system HTML viewer, see Limitations and Bugs.
- In this release of JavaHelp, the viewer toolbar does not include a reload button. The easiest way to reload a file after you change it is to click the "previous" and "next" buttons.

Java™ 2 Platform

If you use the Java 2 Platform to run the JavaHelp system demos, use the `hsviewer.jar` executable JAR file to view HelpSets.

On a Microsoft Windows system you can start this program by double-clicking `hsviewer.jar`. When the program starts, specify the name of a HelpSet file and the path to that HelpSet file. You can also use the file chooser to load a HelpSet (Browse button). Once loaded, click Display to view the HelpSet in the JavaHelp system viewer.

You can also use the `hsviewer.jar` file to create batch files, script files, or JAR files that automate the viewing process. The following describes the syntax of the `hsviewer.jar` command-line interface:

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

The `-helpset` option specifies the HelpSet name *hs_name*

The name of a HelpSet file. The `hsviewer` searches the CLASSPATH and loads the first HelpSet file it encounters with that name. For example:

```
C:\> hsviewer -helpset HolidayHelpSet.hs
```

[IMAGE]
If no arguments are specified on the command line, the graphical interface is used.

Displaying a HelpSet with an Executable JAR File

Using this release you can display a specific HelpSet using the `sunw.demo.jhdemo.Runner` class in `hsviewer.jar`. Using a manifest file to define the `Class-Path`, `Run-Class`, and `Arguments` it is possible to display a HelpSet in a standalone environment. Below is the manifest file used to display the *JavaHelp User's Guide* (this HelpSet):

```
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 following describes the syntax of the manifest file:

```
Main-Class: sunw.demo.jhdemo.Runner
Run-Class: sunw.demo.jhdemo.JHLauncher
Class-PATH: [jar file | directory]
Arguments: [valid arguments for Run-Class]
```

Main-Class   The main class that is executed when this JAR file is run. This is a standard
              argument for executable JAR files. For HelpSets this value is always
              `sunw.demo.jhdemo.Runner`. For example:

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

Run-Class   The class that the `sunw.demo.jhdemo.Runner` executes. For HelpSets this
              value is always `sunw.demo.jhdemo.JHLauncher`. For example:

              `Run-Class: sunw.demo.jhdemo.JHLauncher<`

Class-Path   The class path to use with the `Run-Class`. A space separated list of JAR files
              or directories to be added to existing `classpath` environment variable. Note
              that the files must be specified relative to the location of the executable JAR file.
              For example:

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

Arguments   Arguments passed to the `Run-Class` when executed. For viewing HelpSets,
              this always includes the `-helpset` option with the path to the HelpSet file. For
              example:

              `-helpset jhug.hs`

For the final step you use the `jar` command to create an executable JAR file. The JAR file contains only one file -- the manifest file. All other files specified in the `Class-Path` argument must be specified relative to the JAR file. The `jar` command to create the manifest file is:

   jar -cmf *manifest_file jar_file*

where *manifest_file* is the name of your manifest file and *jar_file* in the name of the JAR file you wish to create.

An example usage of the `jar` command is:

   C:\> **jar -cmf manifest UsersGuide.jar**

Setting Up a JavaHelp System

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

- 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.

Authoring

The JavaHelp system is *file* [1] 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.

It is also important to organize topics to easily package them into a compressed JAR file for delivery to your users.

For both of 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:

[IMAGE]

Links

In most cases, the destination of a link 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>
```

In contrast, the following link uses a full path name to describe the path to the destination of the link:

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

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

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

All files within the JavaHelp system are specified as URLs. The separator between elements (files) in a hierarchy should be "/". In some cases "\" works on the Win32 platforms, however when the files that contain these references are added to JAR files or moved to different platforms, these references no longer work.

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

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

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

[IMAGE]

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.

[IMAGE]

To JAR or not to JAR

You will generally 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.

[IMAGE]
On JDK1.1 systems, this is the only option--applications running on JDK1.1 (which does not support the `jar:` protocol), must include *all* help information (including the HelpSet file and map file) in the JAR file.

On the Java^{TM} 2 Platform, the `jar:` protocol makes other packaging options available. The inclusion or exclusion of the HelpSet and map files from the JAR file has an effect on how you deliver the help information and how you can later update it. The following two sections describe some of the issues to consider when making that decision.

HelpSet File

Under some installation conditions, the HelpSet file could be excluded from the JAR file, while the map file is included. The following diagram illustrates this arrangement:

[IMAGE]

Note that the map file is referred to using the `jar:` protocol.

The HelpSet file is the only help system file referenced explicitly by the application--the JavaHelp system derives all information about the help system from that file. If the HelpSet file is outside of the JAR file, the installation program can update the HelpSet file so the JAR files can be installed anywhere in the user's file system. This is not possible if the HelpSet file is included in the JAR file.

Another advantage of locating the HelpSet file outside of the JAR file is that it can be updated independent of the rest of the HelpSet. For example, additional help information can be added to the user's help system by adding more JAR files and updating the HelpSet file.

Map File

Excluding the map file from the JAR file is possible, but usually not useful.

[IMAGE]

If the map file is located outside of the JAR file, all URLs in the map must use the `jar:` protocol. For example:

```
jar:file:/c:/product/help/Ajar.jar!/File1.html
```

---

[1] The JavaHelp system actually deals with URLs. The URL may resolve to the contents of a file in a file system, a file on the web, a portion of a JAR file, or may even be generated dynamically by a server.

[IMAGE] **See also:**

> The HelpSet File
> The Map File
> JAR Files
> Table of Contents File
> Index File
> Creating the Full-Text Search Database

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. Information about custom navigators is included here as well.
HelpSet title	The name of the top-level TOC folder.
Home ID	The name of the (default) ID that is displayed when the help viewer is called without specifying an ID.
Sub-HelpSets	Optional section can be used to statically include other HelpSets using the `<subhelpset>` tag. The HelpSets indicated using this tag are merged automatically into the HelpSet that contains the tag. More details about merging can be found in Merging HelpSets.

After your product is installed on your user's system, it 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.

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 1.0//EN"
         "http://java.sun.com/products/javahelp/helpset_1_0.dtd">
<helpset version="1.0">
   <!-- title -->
   <title>Java Development Environment - Help</title>

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

   <!-- views -->
   <view>
     <name>TOC</name>
     <label>Table Of Contents</label>
     <type>javax.help.TOCView</type>
     <data>IdeHelpTOC.xml</data>
```

```
    </view>

    <view>
       <name>Index</name>
       <label>Index</label>
       <type>javax.help.IndexView</type>
       <data>IdeHelpIndex.xml</data>
    </view>

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

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

The HelpSet Tags

The following table describes the HelpSet tags:

`<helpset>`	Defines the HelpSet. It can contain all of 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. Contains the following tags:

`<homeID>`[IMAGE]	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 map files. Note that in the 1.0 release only one map file may be specified. Contains the following attribute:

`location`[IMAGE]	URL of the map file.

`<view>`	Defines the navigators used in the HelpSet. Each navigator is defined by its own <view>. May contain all of the following tags:

`<name>`	Names the view.
`<label>`	Specifies a label associated with the view. This string can be accessed by the application and used in the presentation (for example, in the navigator's tab).
`<type>`	Specifies the path to the navigator class.
`<data>`[IMAGE]	Specifies the path to the data used by the navigator. When used with the search navigator, it contains the following attribute:

`engine`[IMAGE]	Path to the search engine class.

`<subhelpset>`[IMAGE]	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`[IMAGE]	URL of the HelpSet file to be merged.

[IMAGE] **See also:**

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

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). By convention, map file names include the `.jhm` suffix.

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

The following 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 1.0//EN"
         "http://java.sun.com/products/javahelp/map_1_0.dtd">

<map version="1.0">
  <mapID target="toplevelfolder" url="images/toplevel.gif" />
  <mapID target="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" />
</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`.

The Map Tags

The following are descriptions of the map tags:

`<map>`   Defines the map. It contains `<mapID>` tags.

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

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

[IMAGE] **See also:**

   The HelpSet File
   JAR Files
   Table of Contents File
   Index File
   Creating the Full-Text Search Database

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

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 JDK bin folder) must be used from a command-line prompt to create, read, and extract data from JAR files.

Sample Help Hierarchy

The following sections refer to this sample help hierarchy:

[IMAGE]

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`.

[IMAGE]
The `jar` command is located in the `bin` directory of the JDK.

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.

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
```

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.

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/
```

[IMAGE]
"!/" is referred to as the *separator*.

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

[IMAGE] **See also:**

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

Table of Contents File

The table of contents (TOC) file describes to the TOC navigator the content and layout of the TOC. The format of the TOC file is based on the World Wide Web Consortium Extended Markup Language (XML). The 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 1.0//EN"
         "http://java.sun.com/products/javahelp/toc_1_0.dtd">

<toc version="1.0">
<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" />
</tocitem>
</toc>
```

This example produces the following TOC display:

[IMAGE]

The TOC Tags

The following table describes the TOC tags:

`<toc>`	Defines the TOC. It contains `<tocitem>` tags.
`<tocitem>`	Defines a TOC entry. Nesting *entry1* within *entry2* defines *entry2* to be hierarchically contained within *entry1*. Uses the following attributes:

        `text`   Specifies the text for the entry.

        `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.

        `image` (*optional*) Specifies the image displayed in front of a TOC entry. 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/file images are displayed.

[IMAGE] **See also:**

The HelpSet File
JAR Files
The Map File
Index File
Creating the Full-Text Search Database

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 a index file:

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

<index version="1.0">
   <indexitem text=".prof extension (profile data)" target="prof.profile" />
   <indexitem text="accelerators (keyboard), see 'keyboard commands'" />
   <indexitem text="adding an existing portfolio" target="proj.import" />
   <indexitem text="adding an existing project">
      <indexitem text="naming the project" target="proj.importdirectory" />
      <indexitem text="naming the storage directory" target="proj.importdirectory" />
      <indexitem text="procedures for" target="proj.importproject2" />
   </indexitem>
   <indexitem text="analyzing program performance, see 'profiler'" />
</index>
```

This example produces the following index display:

[IMAGE]

The Index Tags

The following table describes the index tags:

`<index>`	Defines the index. It can contains `<indexitem>` tags.
`<indexitem>`	Defines a index entry. Nesting *entry1* within *entry2* defines *entry2* to be hierarchically contained within *entry1*. Uses the following attribute:

> `text`   Specifies the text for the entry.
> `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.

[IMAGE] **See also:**

> The HelpSet File
> The Map File
> JAR Files
> Table of Contents File
> Creating the Full-Text Search Database

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 JavaHelp 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.

Types of Context-Sensitive Help

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

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

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.

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 [IMAGE] or chooses the Help->Field-Level Help menu item to change the cursor to the special field-level help cursor ([IMAGE])
2. Selects a GUI object

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

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.

[IMAGE]

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.

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.

[IMAGE] **See also:**

    Implementing Context-Sensitive Help

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.

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:

[IMAGE]

- 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): [IMAGE]
- 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.

  [IMAGE]
  To avoid confusion, be sure the `<TITLE>` tag corresponds to the title used in the table of contents.

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.

[IMAGE]
The ranking process improves when queries are more complex and include more information.

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.

[IMAGE]
In the 1.0 release, the morphing feature is only available for the Engish (en) locale.

[IMAGE] **See also:**

> Creating the Full-Text Search Database
> The `jhindexer` Command
> The `jhsearch` Command
> Localizing the Full-Text Search Database

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.

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:

[IMAGE]

[IMAGE] **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:

   `jhindexer topic1 topic2 topic3`

   [IMAGE]
   Note that 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:

[IMAGE]

[IMAGE] **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`

[IMAGE] **See also:**

> The `jhindexer` Command
> The `jhsearch` Command
> Localizing the Full-Text Search Database
> Full-Text Search

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* [IMAGE]	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.
`-nostopwords`	Causes stopwords to be indexed in the full-text search database.
`-verbose`	Displays verbose messages while processing.

Stopwords

You can direct the JavaHelp system full-text search indexer to exclude certain words from the database index--these words are called *stopwords*. By default, the indexer ignores (does not index) the following stopwords 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 stopword behavior in two ways:

- Use the `-nostopwords` option with the `jhindexer` command. This forces the indexer to ignore stopwords and to index every word in your help topics.
- Use the `config` file to specify your own list of stopwords.

Config File

You can use the `config` file to:

- Change the path names of the files as they are stored in the search database. This is useful when you create the search database using 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 stopwords.

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.

[IMAGE] **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/
```

[IMAGE] **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/
```

Specifying Files for Indexing

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

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

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

Specifying Stopwords

You can specify your own list of stopwords to the indexer using the `config` file. When you specify your own list, the default stopword list is not used. You can specify a list of stopwords in two ways:

● Add the list of words directly into the `config` file using the following format:

```
StopWords word, word, word...
```

● In the `config` file, specify the name of a file that contains a list of stopwords:

```
StopWordsFile filename
```

StopWordsFile must contain a list of stopwords, one stopword per line.

[IMAGE] **See also:**

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

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`.

[IMAGE] **See also:**

> Creating the Full-Text Search Database
> The `jhindexer` Command

Using Popup and Secondary Windows

The JavaHelp system supports two types of HTML-based secondary windows:

- Popup Windows
- Secondary Windows

Both types of secondary windows are implemented as a lightweight component class that can be added to HTML topics by means of the <OBJECT> tag. This topic describes the functionality and characteristics of these windows. For information about how to use the <OBJECT> tag to add popups/secondary windows to your topics, see The JHSecondaryViewer Component. If you are using an authoring tool, the tool will handle these details for you.

[IMAGE]
Popups and secondary windows are not used in the *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:

```
demos\bin
```

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

Differences Between Popups and Secondary Windows

Popups and secondary windows are functionally very 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.

[IMAGE]

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.

[IMAGE]

Working with Popups and Secondary Windows

This section describes the different aspects of popups/secondary windows that you (the help author) can control.

To see examples of popups and secondary windows, try the *object* demo located in:

```
demos\bin
```

The <OBJECT> tag used to add these windows to your help topics is described in Using the <OBJECT> Tag.

Content

The content of popup/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 popup/secondary windows--including links, graphics, and lightweight components (for example, popup/secondary windows). Topics displayed in the windows can be specified using URLs or JavaHelp system IDs.

Activation

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

Button ([IMAGE])	A standard button provided as part of the popup/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.
Text object	A specified string of text inserted inline with the text of the topic. The author can control font characteristics of the text to make it stand out.
Graphic object	A GIF or JPG image.

Window Size

You can specify the height and width of the popup/secondary window. When content exceeds the size of the window, scroll bars are automatically added to the window.

Window Location (secondary windows only)

You can specify the position of secondary windows on the screen.

Named Windows (secondary windows only)

You can name secondary windows. This enables you to reuse an already active window.

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   You can set the font family to:

- Serif
- SansSerif
- Monospaced
- Dialog
- DialogInput
- Symbol

Additional fonts will be supported in future versions of the Java^{TM} 2 Platform.

Font size   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*)

Font weight   You can set the weight of the font to:

- plain
- bold

Font style   You can set the style of the font to:

- plain
- italic

Font color   You can set the color of the font to:

- black
- blue
- cyan
- darkGray
- gray
- green
- lightGray
- magenta
- orange
- pink
- red
- white
- yellow

[IMAGE] **See also:**

The JHSecondaryViewer Component

The JHSecondaryViewer Component

The JavaHelp 1.0 release provides popup and secondary window support through the JHSecondaryViewer lightweight component. Popups and secondary windows are added to help topics by means of the <OBJECT> tag. This topic describes how to use the <OBJECT> tag for this purpose.

To see how popups and secondary windows work, you can experiment with the `object` demo located in:

```
demos\bin
```

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

The topic files used in that demo provide examples of the <OBJECT> tag in use. These files are located in:

```
demos\hs\object
```

The following example creates a popup that is activated by clicking on the text object "Click here":

```
<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>
```

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

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

Parameter Definitions

viewerStyle

Specifies whether the window is a popup or secondary window (see Using Popup and Secondary Windows for a list of differences). The valid values for this parameter are "`javax.help.Popup`" and "`javax.help.SecondaryWindow`". If the `viewerStyle` parameter is omitted the window is created as a secondary window. Example:

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

content

The URL of the topic displayed in the popup or secondary window. The URL must be specified relative to the base address of the containing topic. Example:

```
<param name="content" value="../topicB/glossary_def.html">
```

id

The ID of the topic displayed in the popup or secondary window. The ID must be defined in the map file for that HelpSet. Example:

```
<param name="id" value="gloss_def">
```

viewerActivator

Specifies how the user activates the window (see Using Popup and Secondary Windows for details about activation). Valid values are:

`javax.help.LinkButton`	User activates the window by clicking on the standard button provided as part of the popup/secondary window lightweight component. You can use the `text` parameter to add text to the button or you can use the `iconByName` or `iconByID` parameters to add images to the button.
`javax.help.LinkLabel` [IMAGE]	User activates the window by clicking on a text object or GIF/JPEG image. The text object is specified by means of the `text` parameter and the image is specified by means of the `iconByName` and `iconByID` parameters.

Examples:

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

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

viewerSize

The width and height (in pixels) of the window. Example:

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

viewerLocation (secondary windows only)

The x,y position (in pixels) of the upper left corner of the secondary window on the screen--0,0 being the upper left corner of the screen. Popups ignore this parameter. Example:

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

viewerName (secondary windows only)

Names a window. This makes it possible to reuse an active window. Popups ignore this parameter. Example:

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

iconByName

The URL of a GIF/JPEG image. Images can be used in two ways:

- The user clicks on an image to activate a window. In this case `iconByName` is used in conjunction with the following parameter:

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

- The images added to the standard button the user clicks on to activate a window. In this case `iconByName` is used in conjunction with the following parameter:

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

Example:

```
<param name="iconByName" value="../topicB/glossary_icon.gif">
```

iconByID

The ID of a GIF/JPEG image. The ID must be defined in the map file for that HelpSet. Images can be used in two ways:

- The user clicks on an image to activate a window. In this case `iconByID` is used in conjunction with the following parameter:

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

- The images added to the standard button the user clicks on to activate a window. In this case `iconByID` is used in conjunction with the following parameter:

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

Example:

```
<param name="iconByID" value="gloss_icon">
```

text

The string of text that the user clicks on to activate the window. The font characteristics of this text can be controlled using these parameters:

- `textFontFamily`
- `textFontSize`
- `textFontWeight`
- `textFontStyle`
- `textColor`

Example:

```
<param name="text" value="Click here">
```

textFontFamily

The font family of the text object. Valid values are:

- `Serif`
- `SansSerif`
- `Monospaced`
- `Dialog`
- `DialogInput`
- `Symbol`

Additional fonts will be supported in future versions of Java™ 2 SDK.

Example:

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

textFontSize

The size of the font in the text object. Valid values are:

- `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 pt 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*)

Example:

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

textFontWeight

The weight of the font in the text object. Valid values are:

- `plain`
- `bold`

Example:

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

textFontStyle

The style of the font in the text object. Valid values are:

- `plain`
- `italic`

Example:

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

textColor

The color of the font in the text object. Valid values are:

- `black`
- `blue`
- `cyan`
- `darkGray`
- `gray`
- `green`
- `lightGray`
- `magenta`
- `orange`
- `pink`
- `red`
- `white`
- `yellow`

Example:

```
<param name="textColor" value="red">
```

[IMAGE] **See also:**

Using Popup and Secondary Windows
Creating Lightweight Java Components

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

jh.jar [IMAGE]	The standard library, it 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 may be useful for very simple help systems that do not require a full-text search database or help systems where size is important.
jhall.jar	Includes all of 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.

Supplemental Information

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

API [IMAGE]	The `javadoc` generated documentation of the JavaHelp API is included with this release. You can view the API using the JavaHelp API viewer (`demos\bin\apiviewer`) or using a web browser (`doc\api\index.html`)
Specification	The JavaHelp 1.0 specification is included in this release and can be found at:

`doc\spec\index.html`

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`

[IMAGE] **See also:**

> Adding the JavaHelp System to Applications
> Implementing Context-Sensitive Help
> Embedding JavaHelp Components
> Creating Lightweight Java Components

Adding the JavaHelp System to Applications

The process of adding the JavaHelp system to your application is summarized in the following steps:

1. Import the JavaHelp system classes:

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

   [IMAGE]
   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:

   ```
   try {
       URL hsURL = HelpSet.findHelpSet(null, "../help/myHelpSet.hs");
       hs = new HelpSet(null, hsURL);
   } catch (Exception ee) {
       System.out.println("HelpSet "+helpsetName+" not found")
       return;
   }
   ```

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(helpsetLabel);
   menu help.addActionListener(new CSH.DisplayHelpFromSource(mainHB));
   ```

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 name of the top-level TOC folder.
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 using the tag. The HelpSets indicated using 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.

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.

[IMAGE] **See also:**

> Programming with the JavaHelp System
> Implementing Context-Sensitive Help
> Embedding JavaHelp Components

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

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
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
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

Basic Elements

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

[IMAGE]
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

Setting and Getting `Component` Help Properties

In order 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 menu item. The JavaHelp system `CSH` class provides the following convenient methods for setting and getting the `helpID` for Components and menu items:

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

Sets the helpID for a component.

`public static String getHelpIDString(Component comp);`

Returns the helpID for a component

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

Sets the HelpSet for a component.

`public static HelpSet getHelpSet(Component comp);`

Returns the HelpSet of a component.

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

Sets the helpID for a menu item.

`public static String getHelpIDString(MenuItem comp);`

Returns the helpID for a menu item.

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

Sets the HelpSet for a menu item.

`public static HelpSet getHelpSet(MenuItem comp);`

Returns the HelpSet of a menu item.

Tracking Context-Sensitive Events

The context-sensitive help class provides a method 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.

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

Implementing Context-Sensitive Help

The following sections describe how to use the JavaHelp system to implement the different forms of context-sensitive help.

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.

[IMAGE]
You can implement context-sensitive help without using the HelpBroker, or use it for some tasks and not for others. For example, if your implementation requires something not provided in the HelpBroker (for instance, you want to display 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.

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:

[IMAGE]

HelpBroker Context-Sensitive Methods

A HelpBroker provides the following context-sensitive methods:

`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.

`public HelpSet getHelpSet();`

Gets the current HelpSet for the HelpBroker.

`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 than the current HelpSet (and not contained in the current HelpSet), the `setHelpSet` method is executed.

`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 than the current HelpSet (and not contained in the current HelpSet), the `setHelpSet` method is executed.

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

Enables the Help key on a Component. 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.

```
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`

```
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`

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

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.

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

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.

CSH Inner Classes

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

`CSH.DisplayHelpAfterTracking`

An ActionListener that displays help on a selected object after tracking context-sensitive events. Its constructor takes a HelpBroker object.

`CSH.DisplayHelpFromFocus`

An ActionListener that displays the help of the object that currently has focus. This method is used to enable HelpKey action listening for components other than the RootPane or window. This listener determines if the object with the current focus has a helpID, 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).

An actionListener that gets the helpID for the action source and displays the helpID in the help viewer. Its constructor takes a HelpBroker object.

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);
                    :
```

[IMAGE]
After you set the helpID/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:

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

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/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));
```

Help Menu/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) 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.enableHelpButton(helpButton, "browse.strings", null);
```

System Initiated Help

Although system initiated help is rarely implemented using the help viewer, it is simple to do. 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);
}
                       :
```

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);
        }
                    :
```

[IMAGE] **See also:**

Context-Sensitive Help
Programming with the JavaHelp System
Adding the JavaHelp System to Applications
Embedding JavaHelp Components

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 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());
   ```

   Use the same model for both the content pane and navigator components (viewer1) so changes generated by one component are reflected in the other component. Note that the TOC can be created 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.

5. Add the navigator to the container:

   ```
   content.add(xnav, "Center");
   classViewerIFrame.setContentPane(content);
   ```

[IMAGE] **See also:**

Programming with the JavaHelp System
Adding the JavaHelp System to Applications
Implementing Context-Sensitive Help

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.

[IMAGE]
References to supplemental information is included at the end of this topic.

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 setable parameters, can be used without modification.

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 Swing API:

```
http://java.sun.com/products/jfc/swingdoc-api-1.1/javax/swing/text/Document.html
```

Text formatting information can be derived from the View by querying its attribute set. Use the method `getAttributes` as illustrated 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. Methods exist to determine a number of attributes that include: the font family, font size, font weight, font style, underlining, background color, foreground color. For example, to determine what the default background color of an object:

```
Color color=StyleContants.getBackground(as)
```

For a full list of formatting attributes and corresponding methods see:

Using Parameters

If your component takes parameters, you should follow these two additional steps:

1. Add accessor methods for each setable parameter
2. Create a BeanInfo class that corresponds to the lightweight component class

[IMAGE]
The component must accept parameter elements in any order.

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;
}
```

[IMAGE]
Even if the internal representation is not a String, both the returned value for the "getter" method and the parameter in the "setter" must be a String.

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;
    }
  }
  }
```

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/
```

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. The "getter" and "setter" methods must operate on Strings. 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 using the <param> tag. A parameter is only
recognized if the component has "getter" and "setter" methods for that parameter. The "getter" and
"setter" methods must operate on Strings. 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>
```

Supplemental Information

The following information is provided to supplement the information in this topic.

Lightweight Java Components

For general information about lightweight Java components see:

```
http://java.sun.com/products/jdk/1.2/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 The JHSecondaryViewer Component.

HTML 4.0 Specification

You can find a detailed description of the `<OBJECT>` tag as part of the HTML 4.0 specification:

```
http://w3c.org/TR/REC-html40/
```

[IMAGE] **See also:**

> Using Popup and Secondary Windows
> The JHSecondaryViewer Component

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 Help Presentation	Describes how the presentation aspects of JavaHelp (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.

Localizing 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.

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.

[IMAGE] **See also:**

> Localizing Help Information
> Localizing HelpSets
> Localizing XML Data
> Localizing HTML Data
> Localization and Fonts
> Localizing the Full-Text Search Database

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.

[IMAGE]

Legend:
[IMAGE] Described in Localizing Help Presentation
[IMAGE] Described in the following sections
[IMAGE] Described in Localizing XML Data
[IMAGE] Described in Localizing HTML Data

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.

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`

[IMAGE]
The defaults are derived from the system using the `Locale.getDefault` method.

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.

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.

Shipping Multiple Locales

JavaHelp 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:

[IMAGE]

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>
```

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.

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.

[IMAGE] **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

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.

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

[IMAGE]
Only one encoding can be specified for any file.

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.

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' ?>
```

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

[IMAGE]
It is possible to mix languages within these files. A different language can be specified for each tag; however, only one character encoding can be specified for each file.

The `xml:lang` Attribute

The language for any element (tag) in XML files can be set using the `xml:lang` attribute. For example:

```
<tocitem xml:lang="de" target="jde.intro">Homepage der JDE Online-Hilfe</tocitem>
```

sets the language for that table of contents entry to German. Any elements (`<tocitem>` tags) nested within that tag automatically inherit that language.

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/`).

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`).

[IMAGE] **See also:**

Localizing Help Information
Localizing Help Presentation
Localizing HelpSets
Localizing HTML Data
Localization and Fonts
Localizing the Full-Text Search Database

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.

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)

[IMAGE]
Only one encoding can be specified for any file.

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.

<META> Declaration

The HTML `<META>` declaration can be used to specify the character encoding. Encoding is specified using the `charset` parameter:

```
<META HTTP-EQUIV="Content-Type" CONTENT="text/html;charset=x-euc-jp">
```

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

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/`).

<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">
```

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`).

[IMAGE] **See also:**

Localizing Help Information
Localizing Help Presentation
Localizing HelpSets
Localizing XML Data
Localization and Fonts
Localizing the Full-Text Search Database

Localization and Fonts

The JavaHelp system displays information using the host 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/jdk/1.2/docs/guide/internat/
   ```

[IMAGE] **See also:**

> Localizing Help Information
> Localizing Help Presentation
> Localizing HelpSets
> Localizing XML Data
> Localizing HTML Data
> Localizing the Full-Text Search Database

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).

[IMAGE] **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

Merging HelpSets

The JavaHelp system provides a mechanism for merging HelpSets. You can use the merge functionality to append TOC, index, and full-text search information from one or more HelpSets to that of another HelpSet.

An example of where this functionality might be useful is in an application suite. The application suite may consist of a collection of constituent applications. As constituent applications are purchased and installed, their help information can be merged with help information from the other applications in the suite.

HelpSets can be merged statically or dynamically.

Static Merging

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:

-HelpSet1.hs-

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
<!DOCTYPE helpset
  PUBLIC "-//Sun Microsystems Inc.//DTD JavaHelp HelpSet Version 1.0//EN"
         "http://java.sun.com/products/javahelp/helpset_1_0.dtd">
<helpset version="1.0">
   <title>HelpSet 1</title>
   <maps>
     <homeID>hs1_file1</homeID>
     <mapref location="hs1.jhm" />
   </maps>
   <view>
      <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 1.0//EN"
         "http://java.sun.com/products/javahelp/helpset_1_0.dtd">
<helpset version="1.0">
   <title>HelpSet 2</title>
   <maps>
     <homeID>hs2_file1</homeID>
     <mapref location="hs2.jhm" />
   </maps>
   <view>
      <name>TOC</name>
      <label>Table Of Contents</label>
      <type>javax.help.TOCView</type>
      <data>hs2TOC.xml</data>
   </view>
</helpset>
```

[IMAGE]

[IMAGE]

- The HelpSet that contains the `<subhelpset>` tag is considered to be the *master* HelpSet--all HelpSets are merged with (appended to) 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 sub-HelpSets 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.
- Indexes are merged in the same manner as TOCs--by appending the merged indexes to the master HelpSet.
- Merged search databases are searched sequentially, however, query results are collated and presented together.

Dataless Master HelpSets

It is possible to create *dataless* master HelpSets that do not contain any view or map data of their own--a dataless master serves only as a container for specifying sub-HelpSets. This is accomplished by omitting the `<data>` for the views. All the notes listed above still apply.

The following example HelpSet file could serve as an dataless master for a suite of applications. The JavaHelp system constructs a help display with whichever of the application sub-HelpSets (`app1.hs` - `app2.hs`) are found installed on the user's system:

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

<helpset version="1.0">
   <!-- title -->
   <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>
```

[IMAGE]
A master HelpSet can contain a mixture of both dataless and datafull views.

Dynamic Merging

HelpSets can also be merged programmatically using the JavaHelp API.

The basic API consists of the add(HelpSet) method, and its corresponding remove(HelpSet) method. These methods fire HelpSetEvent events to the HelpSetListeners that have registered interest in them. The ComponentUIs for the TOC, index, and search views register for these events and react to changes.

The semantics of merging is implemented by individual NavigatorViews and JHelpNavigators. 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 API documentation. API information can be viewed using a web browser (doc/api/overview-summary.html) or by using the demos\bin\apiviewer command.

A demonstration program (demos\bin\merge) is included with the JavaHelp system release. This program demonstrates how HelpSets can be merged and removed dynamically. The sources for that program are available in demos\src\sunw\demo\merge.

Table of Contents