JavaHelp 2.0 Specification Final Release

Abstract

This is the proposed final release of the JavaHelp APIs proposed in JSR-097.

There is a change history in section Appendix E on page 109 .

Editor: Roger Brinkley (roger.brinkley@sun.com)

Public Review

March 13, 2003

Sun Microsystems, Inc.

Copyright 2003 Sun Microsystems, Inc.

901 San Antonio Road, Palo Alto, California 94303 U.S.A.

All rights reserved. Copyright in this document is owned by Sun Microsystems, Inc.

Sun Microsystems, Inc. (SUN) hereby grants to you at no charge a nonexclusive, non-transferable, worldwide, limited license (without the right to sublicense) under SUN's intellectual property rights that are essential to practice the JavaHelp 2.0 Specification "Specification") to use the Specification for internal evaluation purposes only. Other than this limited license, you acquire no right, title or interest in or to the Specification and you shall have no right to use the Specification for productive or commercial use.

RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure by the U.S. Government is subject to restrictions of FAR 52.227-14(g)(2)(6/87) and FAR 52.227-19(6/87), or DFAR 252.227-7015(b)(6/95) and DFAR 227.7202-1(a).

SUN MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. SUN SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.

TRADEMARKS

Sun, the Sun logo, Sun Microsystems, JavaSoft, JavaBeans, JavaHelp, JDK, Java, HotJava, HotJava Views, Visual Java, Solaris, NEO, Joe, Netra, NFS, ONC, ONC+, OpenWindows, PC-NFS, EmbeddedJava, PersonalJava, SNM, SunNet Manager, Solaris sunburst design, Solstice, SunCore, SolarNet, SunWeb, Sun Workstation, The Network Is The Computer, ToolTalk, Ultra, Ultracomputing, Ultraserver, Where The Network Is Going, Sun WorkShop, XView, Java WorkShop, the Java Coffee Cup logo, and Visual Java are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

THIS PUBLICATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS PUBLICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THE PUBLICATION. SUN MICROSYSTEMS, INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS PUBLICATION AT ANY TIME.

Table of Contents

1 Introduction

JavaHelp^{TM} is an online help system specifically tailored to the Java platform. JavaHelp consists of a fully featured, highly extensible specification and an implementation of that specification written entirely in the Java language.

JavaHelp enables Java developers to provide online help for:

- Applications (both network and stand-alone)
- Server based applications
- Applets
- JavaBean components
- Desktops
- HTML pages

1.1 Status of this Specification

This document describe the JavaHelp 2.0 specification.

We follow the Java Community Process for the development and revision of Java technology specifications. The JCP is an open and inclusive process that produces high-quality specifications in “Internet-time”. Through this process the critical feedback from all reviewers helps us transform early specifications into a high quality final specifications that satisfied the needs of the user community. The release of this draft specification is part of this process.

When JavaHelp V1.0 Specification was released we expected the specification to continue to be extended in future updates. This is a major update to the JavaHelp V1.0 Specification.

1.2 Change in format

The JavaHelp V1.0 Specification was written and maintained in HTML. For maintenance and ease in creating PDF versions of this Specification we have chosen to develop this version of the specification in Frame. Additionally we have changed the format to be more consistent with a variety of JSR that have been posted. During the conversion from the previous format to the new format every effort was made to preserve the content of the JavaHelp V1.0 Specification. Minor changes were made to improve the readability and accuracy of the document. In all cases the original intent of the V1.0 Specification was maintained.

1.3 How to read this Specification

There are two parts of the documents. The first set is the actual specification that describes the JavaHelp API and its use. Also included are several related sections that, while not technically part of the specification, help in understanding it. These documents describe aspects of Sun’s reference implementation.

We suggest that you begin by reading the specification overview section 2 on page 9. In order to make the JavaHelp system features more concrete and easy to understand, a number of scenarios are explained in section Apendix A on page 82. These scenarios describe some of the different ways the JavaHelp system can be used in Java applications.

You may want to complement your reading of this specification by exploring the JavaHelp System 2.0 Reference Implementation section Appendix B on page 104 which corresponds to this specification. This reference implementation also supports some features that are useful for online documentation systems but that we have judged to not be appropriate for inclusion in the specification at this time. The release also includes examples of documentation and applications that use this specification.

1.4 Related Documents

- JavaHelp 2.0 - Scenarios
- JavaHelp System 2.0 Reference Implementation
- JavaHelp 2.0 - Relaxation Searching

1.5 Further Reading

Up-to-date public information on JavaHelp technology, including our latest presentations at public forums, is available at our home page at http://java.sun.com/products/javahelp.

Further information on Java technology can be found at Sun's Java web site at http://java.sun.com.

1.5.1 JavaHelp Software Mailing Lists

JAVAHELP-INTEREST: We maintain a mailing list as a JavaHelp community resource where interested parties can post and exchange information and inquiries about JavaHelp 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, select advanced subscription features, or to unsubscribe: http://archives.java.sun.com/archives/javahelp-interest.html

JAVAHELP_INFO: We maintain a mailing list for occasional information about JavaHelp software updates and events from the JavaHelp team. To subscribe, send mail to listserv@javasoft.com. In the body of the message type SUBSCRIBE JAVAHELP-INFO

1.6 Your Feedback

We encourage your feedback at

jsr-97-comments@sun.com.

We thank you for your help in making this, and future specifications, meet your needs!

2 Overview

2.1 Introduction

This section is an overview of the JavaHelp specification.

2.2 Features

The main features of JavaHelp are:

Help Viewer	The standard JavaHelp viewer consists of a toolbar and two panes: Content pane Displays help topics formatted using HTML. Navigation pane A tabbed interface that allows users to switch between the table of contents, index, and full text search displays.
Table of contents	XML-based. Collapsible / expandable display of topics in the help system. Supports unlimited levels and merging of multiple TOCs.
Index	XML-based. Supports merging of multiple indexes.
Glossary	XML-based. Based on index file format. Used for short technical descriptions
Favorites	XML-based. Collapsible / expandable display of user's favorite topics.
Full text search	The full text of the content is searchable. Different engines can be used.
Compression and encapsulation	Encapsulation and compression are optional. Uses the standard Java JAR format to encapsulate the entire help system into a single, optionally compressed file.
Embedded help windows	Help windows (individually or in combination) can be embedded directly into application interfaces.
Customization	JavaHelp is designed to permit great flexibility in customizing both the user interface and functionality.

The JavaHelp System 2.0 Reference Implementation section Appendix B on page 104 adds the following to this list:

Flexible Search Engine	The full text of the content can be searched with a flexible search engine that supports multi-word queries.
Popups and Active Content	PopUps can be obtained by embedding lightweight Java components in HMTL pages. Active content (e.g. a button that when pressed can act on the application) can be implemented using the same mechanism.

2.3 Supported Platforms

JavaHelp 2.0 is an Optional Package for the Java 2 platform.

2.4 The Specification

The JavaHelp specification has two main parts:

API	The interface between the application and the help system
File formats	Formats of the files that are part of the help system (HelpSet, table-of-contents, map, index, search database)

2.4.1 API Structure

The classes and methods in JavaHelp 1.0 can be partitioned depending on the tasks so that clients of the API need only use as much as they need. The following are the most useful collections:

HelpSet access	A number of classes provide complete access to a HelpSet collection. This includes classes to control the navigation of online content (`NavigatorView`), the mapping of identifiers to content files (`Map`) and access to HelpSet attributes including the ability to locate, create and merge HelpSets (`HelpSet`).
Basic Content Presentation	A set of classes that provide a generic presentation model for a given platform. A `HelpBroker` is used to present a HelpSet to the user using the default HelpBroker. Context sensitive help is available through `CSH` when coupled with a `HelpBroker`.
Swing Classes	JavaHelp 1.0 defines Java Foundation Class components for Navigators, Content Viewer and Help Viewer which can be embedded into an Application if desired. Custom Navigators are also presented to the API as JFC components.

Full-Text Search	The classes in the `javax.help.search` package provide a simple API for full-text search that can also be used independently of help applications.
JSP Tag Extensions	A set of classes to provide access to HelpSet collections through Java Server Pages.

2.5 Main Concepts

This section describes the fundamental concepts in the specification. More details are available in other parts of this specification and in the javadoc comments of the classes.

2.5.1 HelpSet

A HelpSet is a collection of help content files section 2.5.1.4 on page 12 (topics), navigational views section 3.2.5 on page 19, and map section 2.5.1.3 on page 12information. A HelpSet can contain other HelpSets which are merged section 2.5.5 on page 13 together.

2.5.1.1 HelpSet File

The HelpSet file section 3.2 on page 17 describes a HelpSet and contains:

- Title and other global information
- Map section 3.3 on page 24 information that associates topic IDs with topic files
- One or more navigational views on the content

2.5.1.2 Help Views and Help Navigators

JavaHelp provides "context views" for navigating through content information; for example, most HelpSets will have a view displaying a Table of Contents. A view has a *name*, a *NavigatorView Class* identifying its behavior, some information (e.g. URLs, arguments) used by the instance, and a *JHelpNavigator* which is a GUI component that presents the view to the user. Navigational views are visible to the JavaHelp APIs and the client can request to make a specific view active.

The view's class defines what data it reads, its format, how it will be presented visually, and it also defines the merging rulessection 12.2 on page 70. A view is a subclass of NavigatorView section 5.4 on page 32. The createNavigator() method of a view returns a component that is used to graphically present the view; for the standard views section 2.5.1.2.1 on page 11 this component is a Swing component section 13.2.4 on page 80, specifically, a subclass of JHelpNavigator.

Any JavaHelp implementation must support the standard `NavigatorView` classes, but a HelpSet may include views with other classes, as long as they are available (technically, as long as their definitions are available to the ClassLoader instance of the HelpSet). In many cases this means they are either in the implementation of JavaHelp, in the CLASSPATH, or they are listed in the ARCHIVE attribute of an APPLET.

2.5.1.2.1 Standard Help Views and Help Navigators

All JavaHelp implementations must provide the following classes:

javax.help.TOCView javax.help.JHelpTOCNavigator	NavigatorView and JHelpNavigator for parsing and presenting Table of Contents data.
javax.help.IndexView javax.help.JHelpIndexNavigator	The NavigatorView and JHelpNavigator for parsing and presenting Index data.
javax.help.GlossaryView javax.help.JHelpGlossaryNavigator	NavigatorView and JHelpNavigator for parsing and presenting Glossary data.
javax.help.FavoritesView javax.help.JHelpFavoritesNavigator	NavigatorView and JHelpNavigator for parsing and presenting Favorites data.
javax.help.SearchView javax.help.JHelpSearchNavigator	The NavigatorView and JHelpNavigator for interacting with a search engine using the *javax.help.search.** classes.

The formats used by the TOC, Index, Glossary and Favorites Navigators are described in section 3 on page 16. The Search Navigator interacts with its data through a search-engine that extends the SearchEngine class ; one of the Search View arguments is the class name of the search engine, the rest of the data is passed directly to the search engine.

2.5.1.3 Map File

Applications (or navigational data) do not usually directly reference content files, instead they usually reference them through string identifiers (IDs). This use of IDs insulates content development from application development. Identifiers are mapped to content files in a *`mapfile`*. Multiple map files can be combined within a HelpSet, but an identifier must be unique within a HelpSet in the resulting combined map.

2.5.1.4 Content files

Help information (topics) is described through a collection of URLs. These URLs may be files, may be within a JAR file, or they may be generated dynamically by the server.

Content information is presented depending on its (MIME) type. JavaHelp system implementations are required to provide viewers for HTML3.2 content, but there is a registration mechanism in JHelpContentViewer that is built upon the corresponding mechanism in `JEditorPane` in the Swing package.

2.5.2 HelpBroker

A Help Broker object is the abstraction of the presentation to a HelpSet. An application can use a HelpBroker object to interact programmatically with the presentation of information. The default HelpBroker implementation uses a Swing JFrame, but other implementations are possible (for example, embedding help objects).

2.5.3 URL Protocols

JavaHelp authors can use a number of protocols in the URLs when they are used in the HelpSet file and map files. The specific protocols available depend on the underlying platform. For example, JDK1.1 provides `file:`, `http:`, `ftp:`, while Java 2 adds the jar: protocol which provides access to files within a JAR file. Specific implementations may support additional URL formats.

2.5.4 Search

JavaHelp contains a simple search API in the package javax.help.search. This package provides creation and access to the search databases used by JavaHelp. Different search engines will be identified as subclasses of *javax.help.search.SearchEngine*. The search engine included in the JavaHelp reference implementation is `com.sun.java.help.search.DefaultSearchEngine`.

2.5.5 Merging

In simple applications, the help data may be described in a single HelpSet file. Other situations are best described as a collection of HelpSets, for example:

- An application can merge help information available locally on a user's disk, with information on a web site
- Product suites can merge help information when constituent applications are installed
- HelpSets from an application's constituent Beans section 6 on page 36can be merged for a unified presentation

JavaHelp 1.0 provides a basic mechanism for merging the contents of several HelpSets, the resulting HelpSet merges the map information and the navigational views. See section 12 on page 70 for additional information.

2.5.6 Extensibility

The JavaHelp system is designed so it can be extended in several dimensions:

- The `JHelpContentViewer` registration mechanism can be used to provide new content viewers
- The `HelpBroker` registration mechanism can be used to provide new default HelpBroker
- The `NavigatorView` and `JHelpNavigator` mechanisms can be used to provide new file formats, or new presentations
- The `javax.help.search` classes can be used to replace search engines.

For more details see section 5 on page 32.

2.5.7 Updating Help Information

It is often important to be able to update a product's online help after it has been released. The JavaHelp system supports this in several ways--it is possible to entirely replace the information (if in a JAR), or replace parts of it (if spread over multiple files).

Because you can refer to multiple maps in the HelpSet file, the JavaHelp system provides additional flexibility in this update process. The HelpSet file can extend these maps, making it possible to modify the mapping without modifying any existing map files (which may be inside a JAR file). Finally, since the URL protocols support remote access, if the application is running in a connected environment, it is possible to keep some information remotely.

2.5.8 File Formats

The JavaHelp system specifies the following file formats:

- HelpSet encapsulation and compression using JAR files
- HTML topic files
- HelpSet file
- Map files
- Standard navigation view formats (TOC, index, search)

More information is available in the section 3 on page 16.

2.6 An Example

The following is an example of a HelpSet file.

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

    <helpset version="1.0">

    <!-- the title for the helpset -->
    <title>An Example</title>

    <!-- maps -->
    <maps>
        <homeID>top</homeID>
        <mapref location="jar:file:/c:/Program Files/JWS3.0/JWS3.0.jar!/TheMap.map" />
    </maps>

    <!-- A TOC view -->
    <view>
        <name>TOC</name>
        <label>Table Of Contents</label>
        <type>javax.help.TOCView</type>
        <data>jar:file:/c:/Program Files/JWS3.0/JWS3.0.jar!/TOC.xml</data>
    </view>

    <!-- Another TOC view; note that it has a different name -->
    <view>
        <name>LocalTOC</name>
        <label>Appendix One</label>
```

```
        <type>javax.help.TOCView</type>
        <data>jar:file:/c:/Program Files/JWS3.0/JWS3.0.jar!/LocalTOC.xml</data>
    </view>

    <!-- An Index view -->
    <view>
        <name>Index</name>
        <label>Index</label>
        <type>javax.help.IndexView</type>
        <data>jar:file:/c:/Program Files/JWS3.0/JWS3.0.jar!/Index.xml</data>
    </view>

    <!-- A Search view; note the engine attribute -->
    <view>
        <name>Search</name>
        <label>Search</label>
        <type>javax.help.SearchView</type>
        <data engine="com.sun.java.help.search.SearchEngine">
              jar:file:/c:/Program Files/JWS3.0/JWS3.0.jar!/SearchData
        </data>
    </view>
    </helpset>
```

The HelpSet file starts a DOCTYPE identifying the DTD for the file. The DTD is versioned to allow for future changes. Next follows the title of the HelpSet.

The next section provides information about ID->content file mapping. An ID is given indicating what information within the HelpSet to show by default. Next a mapref tag indicates where to locate the map. In our case the mapfile is contained within a JAR file on the local disk.

The next five sections of the HelpSet file provide information about different views of the content information. The first view, “TOC”, is in a local disk. The next section is a different Table of Contents view, (“LocalTOC”), that uses the same information as the first view, while the next section is an index on the local disk. The next section defines search information and the last two section define a Glossary view and Favorites view.

3 File Formats

3.1 Overview

The JavaHelp system defines the file formats for the meta data files: HelpSet file, Map file, and the data for the standard TOC and Index views. The file formats used in JavaHelp are based on industry standards:

- The HelpSet (help content and meta information) is encapsulated and compressed using the JAR (Java Archive) format.
- Map, table of contents and index file models are described in XML.
- The HelpSet file is based on the Extended Markup Language (XML) as defined by the World Wide Web Consortium (http://w3c.org/XML/).
- Localization is done following the I18N Java conventions.

JavaHelp provides for an extensible set of navigational types, but predefines a few types. The standard types are:

- `javax.help.TOCView` for the Table of Contents.
- `javax.help.IndexView` for the Index.
- `javax.help.SearchView` for the Search.
- `javax.help.GlossaryView` for the Glossary.
- `javax.help.FavoritesView` for the Favorites

The typical files involved in a HelpSet are:

- HelpSet file: Identifies the map, and navigational views (e.g. TOCs, indexes and search database files).
- Map file(s): Defines the map that associates topic IDs used by the application to refer to HTML topic files.
- Table of contents: Defines the table of contents entries, their structure, and the IDs to which they map
- Index: Defines the index entries and the IDs to which they map
- Glossary: Defines the glossary entries and the IDs to which they map
- Search Database: The search database searched by the search engine. The default search database is created using the JavaHelp system `jhindexer` command.
- Content: The HTML topic files that provide information to help users

Document Type Definitions (DTDs) for HelpSet, Map, TOC View data, Index View data, Glossary View data and Favorites View data are included in this specification and can be used for validation. In each of these cases, the valid documents are those valid XML documents in conformance with the DTD except that the DOCTYPE section must not have any inner DTD subset (this is the same restriction used in the W3C SMIL recommended specification).

JAR is used to encapsulate and compress a HelpSet into a single file. Encapsulation and compression are not required, but recommended in most production environments.

3.2 HelpSet File

The HelpSet file is localized following the same naming conventions used with ResourceBundle see section 4 on page 31. Once a HelpSet file for a given locale has been found, no additional localization searches are needed, which is very important in a networked environment.

3.2.1 Format

HelpSet files are encoded in an XML-based syntax; The DTD is dtd/helpset_2_0.dtd. The top level tag is `<helpset>`. A version attribute is optional, when present its value must be "1.0" or “2.0”.

Tag	Description	Allowed In	Body	Attributes
helpset	Helpset definition	top-level	none	**`xml:lang`=”lang”** Language for this item **`version`=”1.0” | ”2.0”** version

The HelpSet file is organized into sections within the `<helpset>` tag. There is a section for ID maps, sections for the navigational views, and a final section for subhelpsets. The general outline of a HelpSet file is:

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

<helpset version="2.0">

    <!-- Global properties -->
    <title>My Title</title>

    <!-- maps section -->
    <maps>
        <homeID>my homeID</homeID>
        <mapref location="url"/>
    </maps>

    <!-- Zero or more View sections -->
    <view>
        <name>TOC</name>
        <type>javax.help.TOCView></type>
        <data>TOC.xml</data>
    </view>

    <!-- Optional subHelpSet section >
        <subhelpset location="file:/c:/Foobar/HelpSet1.hs"/>
</helpset>
```

Whenever a relative URL specification appears in a HelpSet, it is to be interpreted relative to the URL of the HelpSet (note that the constructor for a HelpSet takes a URL).

3.2.2 Processing Instructions

The reference implementation ignores the Processing Instructions.

3.2.3 HelpSet properties

A HelpSet has a title that is used mostly in the presentation.

Tag	Description	Allowed In	Body	Attributes
title	Title of the HelpSet	helpset	Actual title	none

3.2.4 ID Map Section

The second section of a HelpSet file contains information on the mapping of IDs to URLs used for context sensitive help. The homeId tag provides the default entry to present when a HelpSet is first shown. The mapref tag provides a reference to a map file.

Tag	Description	Allowed In	Body	Attributes
maps	Map definition	helpset	empty	none
homeID	Default ID of the HelpSet	maps	ID string	none
mapref	URL to map	maps	data location relative to HelpSet	none

Finally, an ID Map section corresponding to a Bean will want to include a topic ID corresponding to the BeanInfo.getHelpId(). If there is a single Bean for this HelpSet file, the value of `<homeID>` could be used. If several Beans share the HelpSet file, several topic IDs are needed

3.2.4.1 Map Example

The following is an example of a map definition in a HelpSet file:

```
<map>
   <mapref>Map.jhm</mapref>
   <mapref>jar:http://www.sun.com/devpro/JWS3.0Encyclopedia.jar!/Map.jhm</mapref>
</map>
```

3.2.5 Navigational Views Section

The largest sections of a HelpSet file describe the navigational views, which include tables of contents, indices, glossary, favorites, and search. There are three mandatory tags for each view: `<label>`, `<name>`, and `<type>`. Additionally, most views will define `<data>`.

Tag	Description	Allowed In	Body	Attributes
view	View definition	helpset	none	**`xml:lang`=”lang”** Language for this item **`mergetype`=”type”** name of a Merge class
name	a name identifying the view	view	a name identifying the view	none
label	a label to show in the presentation	view	text for the label	none
image	image to show in the presentation	view	id of the image	none
type	a subclass of NavigatorView	view	name of class	none
data	URL spec	view	text of spec	**engine=”string”** a class implementing Search Engine

The language specified in the `xml:lang` attribute of name must not be different that of the view, if that was given explicitly.

3.2.5.1 View Example

The following is an example of a view section in a HelpSet file:

```
<view mergetype=”javax.help.UniteAppendMerge>
   <name>TOC</name>
   <label>Table of Contents</name>
   <type>javax.help.TOCView</type>
   <data>toc.xml</data>
</view>
```

3.2.6 SubHelpSet Section

A HelpSet file can statically include other HelpSets using the `<subhelpset>` tag. The HelpSets indicated using this tag are merged automatically into the HelpSet where the tag is included. If the URL spec refers to a non-existing file, the subhelpset tag is silently ignored; this permits an enclosing HelpSet to refer to subhelpsets that may or not be installed. More details about merging can be found insection 12 on page 70.

Tag	Description	Allowed In	Body	Attributes
subhelpset	Static sub-HelpSet to merge	helpset	empty	**location="string"** URL spec

3.2.7 Presentation Section

The presentation section defines the presentations of help information that can be controlled by the help author. There is only one mandatory tag for each presentation: <name>.

Tag	Description	Allowed In	Body	Attributes
presentation	Presentation definition	helpset	none	**`xml:lang`="lang"** Language for this item **`default`="true|false"** Default presentation for this helpset; the default is false **`displayviews`="true|false"** Display the navigational views of this helpset; the default is true **`displayviewimages`="true|false"** Display the navigational views of this helpset; the default is true
name	a name identifying the presentation	presentation	a name identifying the presentation	none

Tag	Description	Allowed In	Body	Attributes
size	the size of presentation	presentation	none	**`width=”xxx”`** Desired width in pixels **`height=”xxx”`** Desired height in pixels
location	the location of the presentation	presentation	none	**`x=”xxx”`** the x coordinate **`y=”xxx”`** the y coordinate
title	the title of the presentation	presentation	title	none
image	image of the presentation	presentation	id of the image	none
toolbar	indicates a toolbar is to be included	presentation	none	none
helpaction	an individual help action.	toolbar	class name of the HelpAction. Must be of type javax.help.HelpAction	**`image=”string”`** destination ID

For additional information on presentations see section 8.2 on page 48.

3.2.7.1 Presentation Example

The following is an example of a presentation section in a HelpSet file:

```
<presentation default=true>
   <name >main window</name>
   <size width=400 height=400 />
   <location x=200 y=200 />
   <title>Project X Help</title>
   <toolbar>
       <helpAction>javax.help.BackAction</helpAction>
       <helpAction>javax.help.ForwardAction</helpAction>
```

```
   </toolbar>
</presenation>
```

This example would be used as the default presentation. The size would be 400,400 at the location 200,200. Since this is a main window the title of the window would be “Project X Help” and the toolbar would contain the back and forward buttons.

Another example of a presentation used for secondary windows follows:

```
<presentation default=true>
   <name>secondary window</name>
   <size width=200 height=200 />
</presenation>
```

In this example the only attribute set is the size of 200,200. Otherwise the implementation defaults are used.

3.2.8 Implementation Section

The implementation section creates a per HelpSet registry to provide key data mapping to define the HelpBroker class to use in the HelpSet.createHelpBroker method and to determine the content viewer to use for a given MIME type. For more information on setting these attributes programatically see section 5.6 on page 35.

Tag	Description	Allowed In	Body	Attributes
impl	Implementation definition	helpset	none	none
helpsetregistry	Registers the default HelpBroker class	impl	none	**`helpbrokerclass`=”class”** (required) class name, must implement HelpBroker
viewerregistry	Registers a viewer class for given mime type	impl	none	**`viewertype`=”mime/type”** (required) mime type **`viewerclass`=”class”** (required) class name

3.2.8.1 Implementation examples

The following is an example of a implementation section in a HelpSet file:

```
<impl>
    <helpsetregistry helpbrokerclass=”javax.help.DefaultHelpBroker” />
    <viewerregistry viewertype=”text/html” viewerclass=”com.sun.java.help.impl.CustomKit” />
    <viewerregistry viewertype=”text/xml” viewerclass=”com.sun.java.help.impl.CustomXMLKit” />
</impl>
```

3.3 Map Files

Each map file provides a mapping of topic IDs to URLs. Map files are encoded in an XML-based syntax; The DTD is dtd/map_2_0.dtd. The top level tag is `<map>`. A version attribute is optional, when present its value must be "1.0" or “2.0”.

The main tag is `mapID` relating a topic ID and a URL specification. Relative URL specifications are to be resolved against the absolute URL for the map file.

A Map can contain only the following two tags:

Tag	Description	Allowed In	Body	Attributes
map	A Map	top-level	empty	**`xml:lang`=”lang”** Language for this item **`version`=”1.0” | ”2.0”** version
mapID	An individual map entry	empty	map	**`xml:lang`=”lang”** Language for this item **`target=”string”`** ID **`url=”string”`** URL spec

The following is an example of a simple map file:</p>

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
<!DOCTYPE map
  PUBLIC "-//Sun Microsystems Inc.//DTD JavaHelp Map Version 2.0//EN"
         "http://java.sun.com/products/javahelp/map_2_0.dtd">
<map version="2.0">
    <mapID target="intro" url="hol/hol.html" />
    <mapID target="halloween" url="hol/hall.html" />
    <mapID target="jackolantern" url="hol/jacko.html" />
    <mapID target="mluther" url="hol/luther.html" />
    <mapID target="reformation" url="hol/inforefo.html" />
</map>
```

Note that the IDs should be unique within the HelpSet (although they may also appear in a subhelpset of this HelpSet).

3.4 Table of Contents

JavaHelp1.0 specifies one table of contents navigator view: `javax.help.TOCView`. This navigational view models a table of contents. TOC files are encoded in an XML-based syntax; The DTD is dtd/toc_2_0.dtd. The top level tag is `<toc>`. A version attribute is optional, when present its value must be "1.0" or “2.0”. The categoryopenimage, categoryclosedimage and topicimage are optional. If the categoryclosedimage is defined and the categoryopenimage is not defined, the categoryopenimage will be set to the categoryclosedimage.

A TOC can contain only the following two tags:

Tag	**Description**	**Allowed In**	**Body**	**Attributes**
toc	Table of contents	top-level	empty	**`xml:lang`=”lang”** Language for this item **`version`=”1.0”`|`”2.0”** version **`categoryopenimage`=”string”** category open image ID **`categoryclosedimage`=”string”** category closed image ID **`topicimage`=”string”** topic image ID

Tag	Description	Allowed In	Body	Attributes
tocitem	Table of contents item. Tags can be nested to create hierarchical entries	toc, tocitem	empty	**xml:lang=”lang”** language for this item **text=”string”** display text - required **image=”string”** image ID **target=”string”** destination ID **mergetype=”string”** name of Merge class **expand=true | false** expand the tocitem and sub tocitems on initial display **presentationtype=”string”** name of presentation class; a subclass of javax.help.Presenation **presentationname=”string”** name of presentation

3.4.1 Table of Contents Example

The following is an example of a table of contents view in the view section:

```
<view mergetype=”javax.help.UniteAppendMerge”>
   <name>TOC</name>
   <label>Table of Contents</name>
   <type>javax.help.TOCView</type>
   <data>toc.xml</data>
</view>
```

The following is an example of a table of contents 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" categoryopenimage=”chapter” topicimage=”topic”>
   <tocitem text=”Introducing JavaHelp”>
      <tocitem text=”JavaHelp API” target="api" image="image/document.gif"/>
      <tocitem text=”JavaHelp platforms” target="platform" image="image/document.gif"
           presentationtype=”javax.help.SecondaryWindow” presentationname=”mainSecondary”/>
   </tocitem>
</toc>
```

3.5 Index

JavaHelp1.0 specifies one index navigator view: `javax.help.IndexView`. This navigational view models an index. Index files are encoded in an XML-based syntax; The DTD is dtd/index_2_0.dtd. The top level tag is `<index>`. A version attribute is optional, when present its value must be "1.0" or "2.0".

An index can contain the following two tags:

Tag	Description	Allowed In	Body	Attributes
index	Index	top-level	empty	**`xml:lang`="lang"** Language for this item **`version`="1.0" | "2.0"** (optional) version
index-item	Index item. `indexitem` tags can be nested to create hierarchical entries	index, indexitem	empty	**`xml:lang`="lang"** Language for this item **`text="string"`** display text - required **`target="string"`** destination ID **mergetype="string"** name of Merge class **expand=true | false** expand the tocitem and sub tocitems on initial display **`presentationtype`="string"** name of presentation class; a subclass of javax.help.Presenation **`presentationname`="string"** name of presentation

3.5.1 Index Example

The following is an example of a index view in the view section:

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

```
</view>
```

The following is an example of an index file:

```
<?xml version='1.0' encoding='ISO-8859-1'  ?>
<!DOCTYPE index
  PUBLIC "-//Sun Microsystems Inc.//DTD JavaHelp Index Version 2.0//EN"
         "http://java.sun.com/products/javahelp/index_2_0.dtd">
<index version="1.0">
   <indexitem text=”Java Applets”>
      <indexitem text=”Overview” target="applet_over"
           presentationtype=”javax.help.SecondaryWindow presentationname=”mainsw”>
      <indexitem text=”Usage”>
         <indexitem text=”Inserting an applet in a content page” target="applet_insert">
         <indexitem text=”Editing an applet in a content page” target="applet_editing">
     </indexitem>
   </indexitem>
</index>
```

3.6 Glossary

JavaHelp1.0 specifies one glossary navigator view: `javax.help.GlossaryView`. This navigational view models a glossary and is an extension of the `javax.help.IndexView` section 3.5 on page 27. It uses the index file encoding.

3.6.1 Glossary Example

The following is an example of a glossary view in the view section:

```
<view mergetype=”javax.help.SortMerge”>
   <name>glossary</name>
   <label>Glossary</name>
   <type>javax.help.GlossaryView</type>
   <data>glossary.xml</data>
</view>
```

The following is an example of an index file used in a glossary:

```
<?xml version='1.0' encoding='ISO-8859-1'  ?>
<!DOCTYPE index
  PUBLIC "-//Sun Microsystems Inc.//DTD JavaHelp Index Version 2.0//EN"
         "http://java.sun.com/products/javahelp/index_2_0.dtd">
<index version="2.0">
   <indexitem text=”applet” target="applet_def"/>
   <indexitem text=”application” target=”application_def”/>
   <indexitem text=”application server” target=”appServer_def”/>
   <indexitem text=”AWT” target="awt_def"/>
   <indexitem text=”beans” target="bean_def"/>
</index>
```

3.7 Favorites

JavaHelp1.0 specifies one favorites navigator view: `javax.help.FavoritesView`. This navigational view models a users favorites. Unlike other navigational views which store the view’s meta-data within the HelpSet, favorites are stored in the user’s directory in the file **`<user.home>/.JavaHelp/Favorites.xml`**. Favorite files are encoded in an XML-based syntax; The DTD is dtd/favorites_2_0.dtd. The top level tag is `<favorites>`. A version attribute is optional, when present its value must be “2.0”.

A favorites can contain the following two tags:

Tag	Description	Allowed In	Body	Attributes
favorites	User favorites	top-level	empty	**`xml:lang`="lang"** Language for this item **`version`="2.0"** (optional) version
favoriteitem	Favorites item. `favoriteitem` tags can be nested to create hierarchical entries	favorites, favoriteitem	text to show in the presentation	**`xml:lang`="lang"** Language for this item **`target="string"`** destination ID **`url="string"`** URL specification **hstitle="string"** title of HelpSet **`presentationtype`="string"** name of presentation class; a subclass of javax.help.Presenation **`presentationname`="string"** name of presentation

3.7.1 Favorites Example

The following is an example of a favorites view in the view section:

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

- *Favorites do not require a view data definition. Additionally mergetype is ignored.*

The following is an example of an favorites file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE favorites
 PUBLIC "-//Sun Microsystems Inc.//DTD JavaHelp Favorites Version 2.0//EN"
        "http://java.sun.com/products/javahelp/favorites_2_0.dtd">

<favorites version="1.0">
    <favoriteitem text="Love Holidays" >
      <favoriteitem text="On Love" target="onlove" hstitle="History of the Holidays"/>
      <favoriteitem text="Valentines" target="valentine" hstitle="History of the Holidays"/>
    </favoriteitem>
    <favoriteitem text="Numbers" >
      <favoriteitem text="Zero" target="0"  hstitle="Master"/>
```

```
         <favoriteitem text=”Zero - note “ url=”file:/usr/test/hs/Zeronote.html” hstitle=”Master”/>
        </favoriteitem>
</favorites>
```

3.8 Help Content

JavaHelp displays help topic files formatted using HTML. Links are resolved using the URL protocols supported by the underlying platform. Lightweight JComponents can be added to topic pages using the `<OBJECT>` tag.

3.9 Search Database

JavaHelp1.0 specifies one search navigator view: `javax.help.SearchView`. This navigational view models a search interacting with a search database though objects that implement the `javax.help.search` package. The view has an `<engine>` tag that is the name of a class that is a subclass of `SearchEngine`. That class is responsible for interpreting the search database that is described by the URL in `<data>`.

4 Localization

4.1 A Network Environment

JavaHelp follows the standard localization conventions used for ResourceBundle.getBundle(). In a networked environment, each such query may require a number of requests across a network to determine the desired bundle for a given Locale. JavaHelp is designed so that only one such search is required to locate the HelpSet file. All other information is obtained by simple requests that start from this file.

Although the HelpSet file is localized following the same naming and lookup conventions as with Java Property Resource Bundle, for technical reasons they are not property files. Instead, the method `HelptSet.getHelpSet()` is used.

An invocation of `HelpSet.getHelpSet(name, locale)` invokes `HelpUtilities.getLocalizedResource()`. `HelpUtitities.getLocalizedResource()` eventually calls into `ClassLoader.getResource()` with resource names that are based on the name passed and on the Desired locale and the Default locale.

If the first argument to `getHelpSet()` is "name", the search is conducted in the order shown below (from most specific to least specific). The extension is fixed to be "`.hs`":

```
name_language_country_variant.hs
name_language_country.hs
name_language
name
name_defaultlanguage_defaultcountry_defaultvariant
name_defaultlanguage_defaultcountry
name_defaultlanguage
```

This search order is the one used for `ResourceBundle`, where it is not exposed. It is captured and exposed in `HelpUtilities.getCandidates()`.

4.2 Localized Documents

The HTML viewers are required to support localization as specified by the W3C HTML 4.0 standard.

4.3 Full Text Search

Java uses Unicode internally and it is well suited to internationalization and localization. One specific requirement is that the search code be able to deal with documents that are written in both English and another language. This combination occurs often when some documents have been translated but others have not.

4.4 More Details

The "Localizing Help Information" section of the *JavaHelp User's Guide* describes the localization process in detail.

5 JavaHelp™ 1.0 - Customization

5.1 Introduction

There are several mechanisms for customizing JavaHelp:

- Defining a different default HelpBroker
- Associating alternate content viewers with MIME types
- Using non-standard NavigatorView or JHelpNavigator
- Choosing SearchEngine
- Exploiting the URL protocols

5.2 Help Broker

A HelpBroker provides abstraction of the presentation details of a HelpSet. There are two ways of obtaining a HelpBroker: through an explicit instantiation of DefaultHelpBroker, or by invoking the createHelpBroker() method on a `HelpSet` instance. The default `HelpBroker` returned by the `createHelpBroker()` call is implementation dependent--the reference implementation returns `DefaultHelpBroker`.

Constructors of `HelpBrokers` take a HelpSet instance as an argument; `DefaultHelpBroker` uses a JHelp for its presentation, adding to it all the `HelpNavigators` that were requested in the `HelpSet` file and arranging them so they all share the same `HelpSetModel`.

A JavaHelp system implementation may choose not to create a `DefaultHelpBroker` as the default `HelpBroker` for any of several reasons, for example to maintain a consistent presentation. Thus, it is often best to use `createHelpBroker()` to obtain the `HelpBroker`.

5.3 Content Viewers

The JavaHelp reference implementation uses `JEditorPane` to present the HTML content of a given URL. This class supports a registration mechanism by which you can add viewers for given MIME types. This mechanism is exported through the JHelpContentViewer JavaHelp class and can be used to display additional MIME types, or to change the presentation of a given type from the default presentation. The mapping can be changed globally or on a per-HelpSet instance.For additional information, see section 5.6 on page 35 below.

5.4 NavigatorView and JHelpNavigator

The NavigatorView class defines a NavigatorView type and provides access to the information in a `<view>` tag in a HelpSet file. A NavigatorView also provides a `JHelpNavigator` through its `create` method. JHelpNavigator is the Swing class used in the JavaHelp system to capture the presentation of a NavigatorView. A `JHelpNavigator` can be created directly, but more commonly it is created implicitly through the `create()` method in a NavigatorView.

5.4.1 View-Specific Knowledge

Specific `NavigatorView` may have additional methods and fields that encode specific information on the view type. For instance, both TOCView and IndexView provide a `parse` method that can be used to parse a URL that conforms to the file format. These methods use a `Factory` class to provide access for customizing the result of the parsing.

The separation of view data and its presentation means that it is possible to access the view data without having to actually create the presentation. It also means that it is easy to modify the presentation without having to duplicate some data-specific information; for example, by reusing the parsing methods.

5.4.2 Different Formats

The Help Navigator mechanism can also be used to provide access to meta-data that is in a "foreign" or "legacy" format. This might enable an application to access information from legacy applications or an alternate meta-data format such sitemap, or meta-data from the Library of Congress, or other library system. This may be done by creating a new NavigatorView that can parse the "foreign" format but that reuses the presentation from the JavaHelp `JHelpNavigator`.

A variation of this last case, the data is not stored anywhere but it is created dynamically. This is easily accomplished by subclassing TOCView (for instance) and redefining the method getDataAsTree() to return the data whenever invoked.

5.4.3 Different Presentations

A JHelpNavigator selects its presentation through the standard Swing method `getUIClassID()` to indicate its ComponentUI class. A new JHelpNavigator that is not capable or willing to reuse an existing ComponentUI needs to return an appropriate class value in `getUIClassID()`. If appropriate, this ComponentUI may be a subclass of the standard ComponentUI classes (`BasicTOCNavigatorUI.java`, `BasicIndexNavigatorUI.java` and `BasicSearchNavigatorUI.java`) with some methods redefined. A useful method to redefine is setCellRenderer which permits to change the presentation details of the Tree in both TOC and Index presentations.

5.4.4 Two Examples of Custom Views

The five standard Views included in JavaHelp 1.0 (TOCView, IndexView, GlossaryView, FavoritesView and SearchView) cover most online documentation needs, but there are other situations where one might want to have custom views and navigators. As a first example, the Java Tutorial could be used to illustrate the concept of a Help Navigator. The Java Tutorial is an online document that describes the Java Platform. The tutorial is organized into trails: groups of lessons on a particular subject. A version of the tutorial could take advantage of a NavigatorView that supported the notion of a *trail*. Such a view could remember the position within the trail, quickly reference examples within the trail, and navigate to other trails.

Another example is an API class viewer. Such a viewer was created for demonstration purposes and is included in the reference implementation. This NavigatorView uses information collected from source files that are annotated using the javadoc system. The traditional data generated by `javadoc` is produced as HTML files. Static HTML indexes and trees are used to provide navigational information. The result is useful but it is difficult to effectively navigate. The classviewer NavigatorView is customized to dynamically display this information. A picture of the presentation is shown next:

Interface javax.help.HelpBroker

Implementing Classes:
DefaultHelpBroker

public interface **HelpBroker**

The HelpBroker is the default presentation of a HelpSet. A HelpBroker i
HelpSet, a straight-forward implementation is a JHelp() on the HelpSet.
Navigational View, and can display a given ID (help topic).

See Also:
HelpSet, JHelpNavigator, javax.help.HelpVisitListener

In this example there are three navigational views: TOC, Index, and Search. Index is an index of all the methods, classes, and packages, and Search provides a full-text search of all the `javadoc` information. The TOC view uses the new classview NavigatorView. When a class is selected in the top pane of the navigator, the `JHelpNavigator` determines if it has already loaded the metadata for that class. If not, it presents the fields, constructors and methods in the bottom pane. When a method is selected, the appropriate content file is presented in the JavaHelp system TOC pane. In this particular prototype, the information presented is only that of the selected class but the navigator could easily provide access to inherited information too.

For this example, we use the new Doclet facility in JDK1.2 to generate the desired metadata.

5.5 Search Engines

The standard `NavigatorView` and `JHelpNavigator` search classes (`javax.help.SearchView` and `javax.help.JHelpSearchNavigator`) provide an interaction with search engines via the classes in the `javax.help.search` package. `SearchView` views may have an optional `<engine>` attribute of their `data` tag indicating the specific `javax.help.search.SearchEngine` subclass to use to perform searches. The default is `com.sun.java.help.search.DefaultSearchEngine`, which is the search engine included in the reference implementation.

The same view and presentation can be used with other search engines following the same protocol, by naming the `SearchEngine` class in the `<engine>` attribute and making the class available.

Different view and or presentations of search can be provided using the standard customization mechanisms for this. These may, or not, reuse the default search engine.

5.6 Key-Data Map

`HelpSet` provides a simple registry mechanism that provides per-instance or global key-data mapping. The mechanism can be accessed via the `setKeyData`, `setDefaultKeyData` and `getKeyData` methods. This mechanism is used by the `JHelpContentViewer` to determine the EditorKit to use for a given MIME type, and also to determine the HelpBroker to use in the `HelpSet.createHelpBroker()` method.

The per-HelpSet registry will be instantiated from the contents of the `<impl>` section of the HelpSet file in the 1.0 version of the JavaHelp system.

5.7 Using new URL protocols

Another mechanism for extending JavaHelp is by providing new protocols that can, for example, provide SGML -> HTTP translation. This is very easy to do in a Java application by defining a few simple URL classes.

6 JavaHelp™ 1.0 - JavaBeans Help data

6.1 Introduction

There are different types of help information associated with JavaBeans components.

- Help information about the JavaBeans component to use by a "container"
- Help information used by the JavaBeans component itself (for example, a popup)
- Help information to be attached to a JavaBeans component instance

In the first case, information is associated with the presence of the JavaBeans component in its container. For example, this is what happens when a JavaBeans component is added to a Builder tool palette, or when a new JavaBeans component plug-in is dropped into JMAPI.

The second case occurs at runtime within a JavaBeans component. For example, the JavaBeans component is a complex plug-in. While in a popup window for that plug-in, we want to display the help information in a form that is consistent with whatever display presentation the container uses for help information.

The third case occurs when a JavaBeans component is instantiated into a container and it is given some semantics by customizing it and by attaching to events and actions. In this case we want an easy mechanism to assign help data that describes the semantics so that a gesture can retrieve that help data.

The mechanisms described in the following section pertain to the first two cases. The third situation is covered by the mechanisms for context-sensitive help and other, more ad hoc, mechanisms.

6.2 Help Information

The needs of the two cases described above require the association and retrieval of two pieces of information per JavaBeans component:

- `helpSetName`: the name of a HelpSet that contains help information
- `helpID`: a home ID within that HelpSet to use to present data

Having two different pieces of information (cf. having the HelpID be a fixed value) provides for additional packaging flexibility and leads to a nice default convention, and useful default values are important to keep within the JavaBeans design philosophy. The default for this information depends on whether the name of the JavaBeans component is in the unnamed package or not:

Name is of the form *OurButton*:

- `helpSetName`: add a *Help.hs* to name: *OurButtonHelp*.`hs`
- `helpID`: add ".topID" to name: *OurButton.topID*

If the name is of the form *sunw.demo.buttons.OurButton*:

- `helpSetName`: drop the shortname, replace '.' with '/' and add a '/Help.hs': *sunw/demo/buttons/Help.hs*.
- `helpID`: add ".topID" to name: *sunw.demo.buttons.OurButton.topID*:

6.3 Mechanism

The proposed mechanism is to use two optional String-valued BeanInfo attributes with the names suggested above: *"helpSetName"*, and *"helpID"*. This mechanism is relatively simple, does not require the JavaBeans component to be initialized, and it is consistent with other uses of BeanInfo attributes (e.g. Swing's use for container information).

To simplify following the default rules described above, we add two methods to a JavaHelp class that take a Class object and return the desired Strings after consulting the appropriate methods.

6.4 An Example:

Below is the buttons example from the BDK, modified to provide Help information. This example uses the default values for HelpSetName and HelpId:

6.4.1 Manifest and JAR File

The manifest file just changes to include the Help files; it would look like:

```
// Beans, Implementation Classes, and Gif images are as before

// the HelpSet file
Name: sunw/demo/buttons/Help.hs

// The Map file
Name: sunw/demo/buttons/help/Map.html

// Actual html data - in this case all in one file
Name: sunw/demo/buttons/help/Buttons.html

// View data
Name: sunw/demo/buttons/help/toc.xml

Name: sunw/demo/buttons/help/index.xml

Name: sunw/demo/buttons/help/search.dat
```

6.4.2 The HelpSet File

All the HelpSet files are the same. The HelpSet file is quite simple (see section 6.4.2 on page 37 for details on the classes view).

```
# ...

# map URL
<homeID>sunw.demo.buttons.topId</homeID>
<map>
    <data>!/sunw/demo/buttons/help/Map.html</data>
</map>
```

```
# data views
<view>
    <name>TOC</name>
    <label>Table of Contents</label>
<type>javax.help.TOCView</type>
    <data>!/sunw/demo/buttons/help/toc.xml</data>
</view>

<view>
    <name>Index</name>
    <label>Index</label>
    <type>javax.help.IndexView</type>
    <data>!/sunw/demo/buttons/help/index.xml</data>
</view>

<view>
    <name>Search</name>
    <label>Search</label>
    <type>javax.help.SearchView</type>
    <engine>com.sun.java.help.search.DefaultSearchEngine</engine>
    <data>!/sunw/demo/buttons/help/search.dat</data>
</view>
```

6.4.3 The Help Map

In this simple example, the Map just handles the top IDs, plus a global introduction to the buttons package.

```
  sunw.demo.buttons.topId="!/sunw/demo/buttons/help/Buttons.html#Top"
  sunw.demo.buttons.OurButton.topId="!/sunw/demo/buttons/help/Buttons.html#OurButton"
  sunw.demo.buttons.ExplicitButton.topId="!/sunw/demo/buttons/help/
Buttons.html#ExplicitButton"
  sunw.demo.buttons.OrangeButton.topId="!/sunw/demo/buttons/help/Buttons.html#OrangeButton"
  sunw.demo.buttons.BlueButton.topId="!/sunw/demo/buttons/help/Buttons.html#BlueButton"
```

6.5 An Alternative Arrangement

A alternative arrangement would have been to place all the help data in a single nested JAR file. For example:

6.5.1 Manifest and JAR file

```
// The Beans, Implementation Classes and Gifs as before
// The Help data
Name: sunw/demo/buttons/Help.hs
// The rest of the Help data
Name: sunw/demo/buttons/help.jar
```

6.5.2 The HelpSet File

The Help file has to change a bit:

```
# no property requests
# map URL
<homeID>sunw.demo.buttons.topId</homeID>
<map>
    <data>!/sunw/demo/buttons/help.jar!/Map.html</data>
</map>
# data views
<view>
    <name>TOC</name>
```

```
        <label>Table of Contents</label>
        <type>javax.help.TOCView</type>
        <data>!/sunw/demo/buttons/help.jar!/toc.xml</data>
    </view>
    <view>
        <name>Index</name>
        <label>Index</label>
        <type>javax.help.IndexView</type>
        <data>!/sunw/demo/buttons/help.jar!/index.xml</data>
    </view>
    <view>
        <name>Search</name>
        <label>Search</label>
        <type>javax.help.SearchView</type>
        <engine>com.sun.java.help.search.DefaultSearchEngine</engine>
        <data>!/sunw/demo/buttons/help.jar!/search.dat</data>
    </view>
```

6.5.3 The Help Map

In this example, we can choose to use exactly the same Help map as what we used in the previous arrangement.

7 Server Based JavaHelp

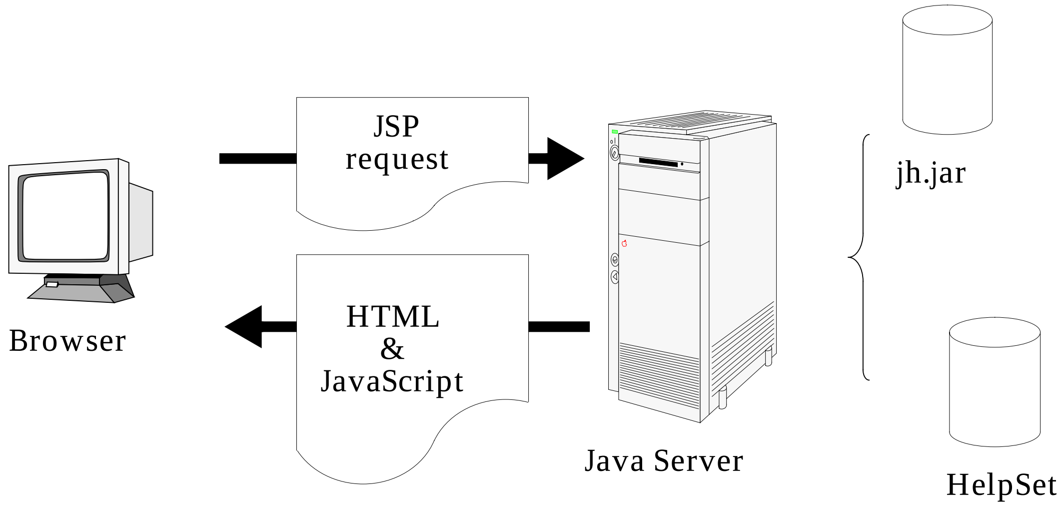
Server based applications have the same need for online help as client based applications. The JavaHelp V1.0 API provided a good foundation for developing online help for server based applications. However, the specification was not complete in defining a standard for a JavaHelp bean and Java Server Pages TM(JSP) tag library for accessing HelpSet data. This section defines the new standards in JavaHelp for server based applications.

7.1 Java Server Pages

JSP allows web developers to develop content-rich, dynamic pages rapidly and easily. JSP uses XML-like tags to encapsulate the logic that generates web content. JSP pages separate the page logic from its design and display, which prevents the overlapping of roles between web designers and programmers. Designers design the web pages and programmers add the logic and code to them. For more information and useful tutorials on JavaServer Pages technology, see JavaServer Pages Dynamically Generated Web Content.

7.2 Server Based JavaHelp Architecture

By combining the JavaHelp API with new JavaHelp JSP tag libraries web developers will be able to provide help for server based applications. The diagram below illustrates the architecture.



A browser initiates a JSP request. Examples of a JSP request are displaying the help content in HelpSet, the navigators, or the data for a given navigator. Typically the JSP request will contain JavaBeans as well as JSP tag extensions. The JavaServer turns the request into a Java Servlet. The servlet access the appropriate information from the HelpSet using the classes in JavaHelp library (jh.jar) and JavaHelp tag library (jhtags.jar) and returns HTML and possibly JavaScript or DHTML to the browser.

7.3 JavaHelp Server Components

Access to HelpSet data on a server is accomplished through a combination of JavaHelp specific Java Beans and JSP tag extensions. This section defines the standard JavaHelp JavaBeans and JSP tag extensions and scripting variables

7.3.1 JavaHelp Server Bean

`ServeletHelpBroker` is the Java Bean that stores help state information such as the HelpSet in use, the currentID, the current navigator and other pieces of help information. While it implements the `javax.help.HelpBroker` interface some of the methods are either not implemented or throw `UnsupportedOperationExceptions` if called. The list of methods not implemented are listed below.

Method	Result
`initPresentation()`	No Operation
`setDisplayed(boolean)`	Ignored
`boolean isDisplayed()`	Always returns true
`enableHelpKey(Component, String id, HelpSet`	No Operation
`enableHelp(Component|MenuItem, String id, HelpSet`	No Operation
`enableHelpOnButton(Component|MenuItem, String id, HelpSet`	No Operation

One new method is added to `ServletHelpBroker`

Method	Definition
`NavigatorView getCurrentNavigatorView()`	Returns the current navigator as a NavigatorView

7.3.1.1 Usage

The `ServletHelpBroker` is used in the JSP request with a session scope. As such it would remain in existence for the duration of a session.

```
<jsp:useBean id=”helpBroker” class=”ServletHelpBroker” scope=”session” />
```

The `ServletHelpBroker` methods can be called either within tag libraries as illustrated below.

```
<jh:validate helpBroker=”<%= helpBroker %>” />
```

or directly in the JSP as illustrated below

```
<FRAME SRC=”<jsp:getPorperty name=”helpBroker” property=”currentURL” />” NAME=”contentsFrame”
  SCROLLING=”AUTO”>
```

7.3.2 JavaHelp JSP Tag Extensions

While it would be possible to retrieve all the HelpSet information required for displaying online help or documentation using Java Beans and JSP scriptlets, a standard set of tag extensions in the JavaHelp tag library enables application functionality to be invoked without the appearance of programming. The JavaHelp tag library is a common set of building blocks that

- conceals the complexity of access to HelpSet data
- introduces new scripting variable into a page
- handles iterations without the next for scriptlets

The JavaHelp tags are define below:

Tag	Tag Class TEI Class	Description	Attributes
validate	ValidateTag	Validate a helpBroker with the various parameters. Allows for easy setup of a helpBroker with a new HelpSet. Also provides for merging HelpSets and the setting of the currentID	**`helpbroker`** *required* HelpBroker object **`setInvalidURL`** *not required* String representing the URL for Invalid HelpSet message. **`helpSetName`** *not required* String representing the URL for the HelpSet name **`currentID`** *not required* String id of desired currentID **`merge`** *not required* boolean value - if true then merge HelpSet into current HelpSet if one exists, otherwise do not merge helpset

Tag	Tag Class TEI Class	Description	Attributes
navigators	NavigatorsTag NavigatorsTEI	Returns NavigatorView information for a given HelpBroker	**`helpbroker`** *required* HelpBroker object **`currentNav`** *not required* String name of the current navigator
tocItem	TOCItemTag TOCItemTEI	Provided with a TOCView, returns TOCItem information	**`tocView`** *required* TOCView object **`helpbroker`** *required* HelpBroker object **`baseID`** *not required* determined by expression String text for the base identification of the TOCItem
indexItem	IndexItemTag IndexItemTEI	Provided with a IndexView, returns IndexItem informatation	**`IndexView`** *required* determined by expression IndexView object **`helpbroker`** *required* HelpBroker object **`baseID`** *not required* String text for the base identification of the TOCItem

Tag	Tag Class TEI Class	Description	Attributes
searchItem	SearchItemTag SearchItemTEI	Provided with a SearchView, returns SearchItem informatation	**`searchView`** *required* SearchView object **`helpbroker`** *required* HelpBroker object **`baseID`** *not required* String text for the base identification of the SearchItem

Unless otherwise specified all attributes values are determined by expression. Also with the exception of validate the body of all tags are JSP.

7.3.2.1 Validate Usage

The validate tag is designed to used once within a jsp as illustrated

```
<jh:validate helpBroker="<%= helpBroker %>" />
```

This verifies that a valid HelpBroker exists and then loads the HelpSet that has either been defined in `validate` using the `helpSetName` argument or as an HTTP POST request.

7.3.3 Navigator Scripting Variables

The navigator, tocItem, indexItem and searchItem tag extensions introduce a predefined set of scripting variables into a page. This allows the calling JSP to control the presentation without performing the processing involved in determining the content.

Unless otherwise specified all scripting variables in JavaHelp create a new variable and the scope is set to NESTED. NESTED variables are available to the calling JSP only within the body of the defining tag.

7.3.3.1 Navigator Variables

The navigator variable are defined in the table below.

Variable	Data Type	Description
classname	java.lang.String	Name of the NavigatorView class
name	java.lang.String	Name of the View as defined in the HelpSet
tip	java.lang.String	Tooltip text for the View

Variable	Data Type	Description
iconURL	java.lang.String	URL for the icon if set with the imageID attributed in the HelpSet
isCurrentNav	java.lang.Boolen	True if current navigator; false otherwise

7.3.3.1.1 Navigator Variable Usage

The navigator tag is useful to return information about the current navigator. In the illustration below the navigator tag is used to determine the navigators that are used in the HelpSet and sets an HTML IMG tag based on the navigator name.

```
<jh:navigators helpBroker="<%= helpBroker %>" >
<A HREF="navigator.jsp?nav=<%= name %>">
<IMG src="<%= iconURL!=""? iconURL : "images/" + className +".gif" %>" Alt="<%= tip %>"
  BORDER=0></A>
</jh:navigators>
```

7.3.3.2 tocItem Variables

The tocItem variable are defined in the table below.

Variable	Data Type	Description
name	java.lang.String	tocItem text as defined in the name attribute
target	java.lang.String	tocItem target as defined in the target attribute
parent	java.lang.String	hex value identifying parent node
parentID	java.lang.String	String identification identifying parent node
node	java.lang.String	hex value identifying this node
nodeID	java.lang.String	String identifying this node
iconURL	java.lang.String	URL for the icon if set with the imageID attributed in the tocItem or the defaults imageIDs in the toc
iconOpenURL	java.lang.String	URL for the open icon if set with the imageID attributed in the tocItem or the default imageIDs in the toc
contentURL	java.lang.String	URL for the content represent by this item

7.3.3.2.1 tocItem Usage

The tocItem tag returns information about the tocItems defined in a TOCView. In the illustration below the TOCView returns tocItems scripting variables that are added to a javascript tag addNode.

```
<% TOCView curNav = (TOCView)helpBroker.getCurrentNavigatorView(); %>
<jh:tocItem tocView="<%= curNav %>" helpBroker="<%= helpBroker %>" >
```

```
addNode("<%= name %>","<%= iconURL!=""?iconURL:"null" %>","","-1","<%=
  contentURL!=""?contentURL:"null" %>","<%= target %>","<%= nodeID %>","<%= parentID %>" );
</jh:tocItem>
```

7.3.3.3 indexItem Variables

The indexItem variable are defined in the table below.

Variable	Data Type	Description
name	java.lang.String	indexItem text as defined in the name attribute
target	java.lang.String	indexItem target as defined in the target attribute
parent	java.lang.String	hex value identifying parent node
parentID	java.lang.String	String identification identifying parent node
node	java.lang.String	hex value identifying this node
nodeID	java.lang.String	String identifying this node
contentURL	java.lang.String	URL for the content represent by this item

7.3.3.4 indexItem Usage

The itemItem tag returns information about the indexItems defined in a IndexView. In the illustration below the IndexView returns indexItems scripting variables that are added to a javascript tag addNode.

```
<% IndexView curNav = (IndexView)helpBroker.getCurrentNavigatorView(); %>
<jh:indexItem indexView="<%= curNav %>" helpBroker="<%= helpBroker %>" >
addNode("<%= name %>","null","","-1","<%= contentURL!=""?contentURL:"null" %>","<%= helpID
  %>","<%= nodeID %>","<%= parentID %>" );
</jh:indexItem>
```

7.3.3.5 searchItem Variables

The searchtem variable are defined in the table below.

Variable	Data Type	Description
name	java.lang.String	Unique name of the searchItem
helpID	java.lang.String	Id associated with this searchItem
confidence	java.lang.String	The quality of the hits as returned the search engine
hits	java.lang.String	number of hits
contentURL	java.lang.String	URL for the content represent by this item
hitBoundries	java.lang.String	A list of boundaries. Returns in the format of {begin, end},...

7.3.3.5.1 SearchItem Usage

The searchItem tag returns information about the searchItems defined in a SearchView. In the illustration below the SearchView returns searchItems scripting variables that are added to a javascript tag addNode.

```
<jh:searchItem searchView="<%= curNav %>" helpBroker="<%= helpBroker %>" query="<%= query %>" >
addNode("<%= name %>","<%= confidence %>","<%= hits %>","<%= contentURL %>","<%= helpID %>" );
</jh:searchItem>
```

8 Presentation of Help Content

8.1 Introduction

In V1.0 help content could only be presented in a single presentation method as defined within the HelpBroker. In the reference implementation this was a tri-paned main window. While the components to provide other presentations were present in V1.0, the six invocation mechanisms (see section A.2 on page 82) for activating help limited the display to a single presentation.

V2.0 improves this functionality by allowing multiple forms of help content presentation. Supported presentations will now include the tri-paned main window, named secondary windows and popups. Additionally, reference implementations and end users have the capability of providing additional presentation forms.

8.2 Presentation Class

Greater flexibility in the presentation of help content is provided in the new `Presentation` class. This abstract class provides developers with a generic interface for the development of alternative presentations. Each implementation of `Presentation` will need to override the static method `getPresentation` according to it's own needs. For instance `Popup` would create a single object whereas `SecondaryWindow` would look for an existing secondary window that matched the parameters before creating a new `SecondaryWindow` object.

The key to `Personation` is the generic methods that are required for all presentations.

Method	Description
`getPresentation(hs, name)`	Static method to return a Presentation for the given type. By default Presentation will return null and concrete class extending Presentation should override this method with their own implementation.
`setHelpSetPresentation (HelpSet.Presentation)`	Set the Presentation attributes from a named presentation in the HelpSet.
`get/setCurrentID(ID)` `setCurrentID(stringId)`	Get/set the current ID for the presentation.
`get/setCurrentURL(URL)`	Get/set the current URL for the presentation.
`get/setFont(Font)`	Get/set the Font for the presentation.
`get/setLocale(Locale)`	Get/set the Locale for the presentation.
`get/setHelpSet(HelpSet)`	Get/set the HelpSet for the presentation.

`is/setDisplayed(boolean)`	is / set the presentation displayed.
`get/setSize(Deminsion)`	Get / set the size for the presentation.

8.2.1 Presentation Extensions

The specification calls for all implementation to provide four extensions of the `Presentation` class. One of the extensions is an abstract class for window presentations

8.2.1.1 Popup

`Popup` is a direct implementation of `Presentation`. A `Popup` contain only a content viewer. It is intended to provide immediate help and then allow the user to continue working. Once a popup loses focus, it is destroyed.

8.2.1.2 Window Presentations

Window presentations require additional controls than what is provided in `Presentation`. An abstract class `WindowPresentation` provides additional methods generic for window based presentations.

Name	Description
`get/setLocation(Point)`	Get / set the location of the presentation.
`get/setTitle(String)`	Get / set the title for the presentation.
`get/setCurrentView(stringView)`	Get / set the current navigational view for the presentation.
`is/setViewDisplayed`(boolean)	Is / set the navigation views to be displayed in the presentation.
`is/setDestroyedOnExit(boolean)`	Is / set the window to be destroyed on exit.
`get/setActivationWindow(Window)` `setActivationObject(Object)`	Get / set the current activation Window for the presentation. Optionally a method is provided to do this from an Object
`is/SetTitleFromDocument (boolean)`	Determines if the title is set from the displayed Document. This is generally usefull for presentations such as SecondaryWindows.
`is/SetToolbarDisplay(boolean)`	Determines if the toolbar is displayed
`getHelpSetPresention()`	Returns the HelpSet.Presentation if one was set.
`createHelpWindow()`	Creates the help window used in the Presentation.

`destroy()`	Destroy this object and any subobjects it created.

Additionally the `WindowPresentation` will maintain a static list of `WindowPresentations`. These are accessed through the protected method `get/putWindowPresentation`.

8.2.1.2.1 Main Window

The `MainWindow` is the main presentation for the JavaHelp system. By default it is a tri-paned fully decorated window consisting of a tool bar, navigator pane, and help content viewer. By default the object is not destroyed when the window is closed.

8.2.1.2.2 Secondary Window

A `SecondaryWindow` is similar to the `MainWindow` in that it is a fully decorated window. By default it only contains a help content viewer though could optionally include a toolbar and / or navigators. Unlike the main window it is destroyed by default on closing. Additionally, secondary windows have a name associated with them. Use of a named secondary window will cause the current contents to be replaced if a named window is visible.

8.3 Help Author Presentation Control

The help author can override any of the presentations attributes in the HelpSet file. For more information on overriding presentation attribute defaults see section 3.2.7 on page 21.

8.4 Activating Help in Presentations

There are six invocation mechanism for activating help:

- Field-level context-sensitive help
- Window-level context-sensitive help
- User initiated context-sensitive help
- System initiated context-sensitive help
- Navigator
- Viewer

For more information on invocation mechanism see section A.2 on page 82.

V1.0 was limited to the presentation of HelpBroker (generally a tri-pane window). Each of the invocation mechanism have been extended to allow presentations in one of the standard or custom `Presentations`. The new methods or file formats are in **Bold**.

Method or File Format	Invocation Mechanism

Field-level CSH

```
CSH.DisplayHelpAfterTracking(HelpBroker)
CSH.DisplayHelpAfterTracking(
     HelpSet,
     StringPresentation,
     StringPrentationName)
```

Window-level CSH

```
CSH.DisplayHelpFromFocus(HelpBroker)
CSH.DisplayHelpFromFocus(
     HelpSet,
     String presentation,
     String presentationName)

HelpBroker.enableHelpKey(MenuItem,
     String id,
     HelpSet)
HelpBroker.enableHelpKey(Component,
     String id,
     HelpSet)
HelpBroker.enableHelpKey(Object,
     String id,
     HelpSet,
     String presentation,
     String presentationName)
```

User Initiated CSH

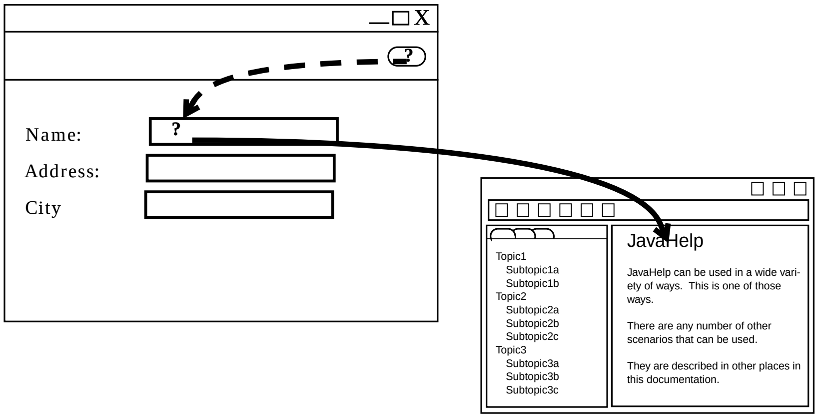
```
CSH.DisplayHelpFromSource(HelpBroker)
CSH.DisplayHelpFromSource(HelpSet,
     String presentation,
     String presentationName)
HelpBroker.enableHelpOnButton(MenuItem
     String id,
     HelpSet)
HelpBroker.enableHelpOnButton(Component,
     String id,
     HelpSet)
HelpBroker.enableHelpOnButton(Object,
     String id,
     HelpSet,
     String presentation,
     String presenationName)
```

HelpBroker.setCurrentID(String id) **Presentation.setCurrentID(String id)** **HelpBroker.showID(String id,** **String presentation,** **String presentationName)** HelpBroker.setCurrentID(ID) **Presentation.setCurrentID(String id)** **HelpBroker.showID(ID,** **String presenation,** **String presentationName)**	System Initiated CSH
TOCItem(target=”id”) **TOCItem(target=”id”** **presentation=”presentation_class”** **presentationName=”presentation_name”)** IndexItem(target=”id”) **IndexItem(target=”id”** **presentation=”presentation_class”** **presentationName=”presentation_name”)**	Navigator

8.4.1 Field-level Context-sensitive Help

No changes to existing code are required to display field-level context-sensitive help in the `MainWindow`. The following invocation would display the field-level help in a main window.

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



JavaHelp can be used in a wide variety of ways. This is one of those ways.

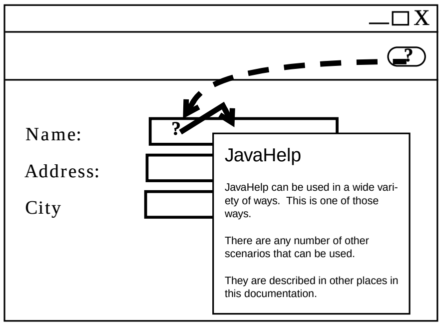
There are any number of other scenarios that can be used.

They are described in other places in this documentation.

View in a Main Window

The following invocation would display the field-level help in a popup as illustrated.

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



JavaHelp can be used in a wide variety of ways. This is one of those ways.

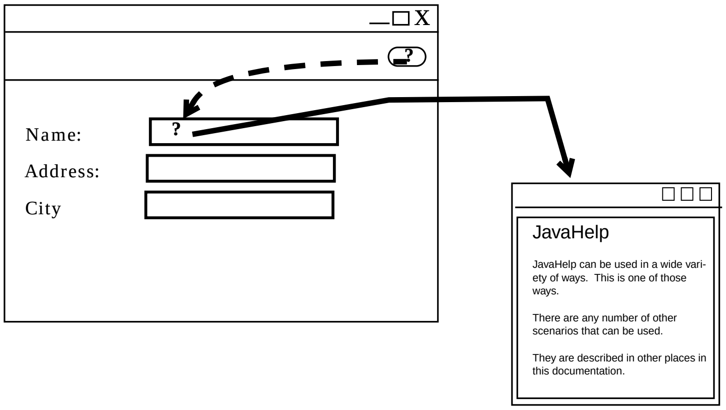
There are any number of other scenarios that can be used.

They are described in other places in this documentation.

View in a Popup

The following invocation would display the field-level help in a secondary window as illustrated.

```
JToolBar toolbar=new JToolBar();
...
helpbutton = addButton(toolbar, "images/help.gif", "help");
helpbutton.addActionListener(new CSH.DisplayHelpAfterTracking(mainHS,
                                                         "javax.help.SecondaryWindow",
                                                         “mainSW”));
```



JavaHelp can be used in a wide variety of ways. This is one of those ways.

There are any number of other scenarios that can be used.

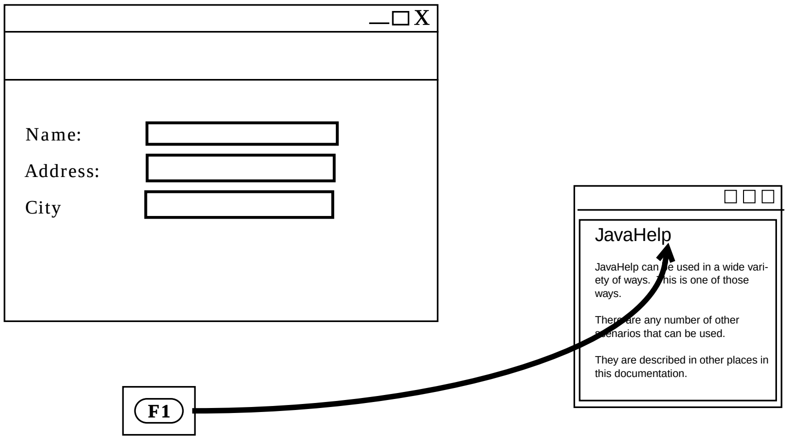
They are described in other places in this documentation.

View in a Secondary Window

8.4.2 Window Level Context-Sensitive Help

No changes to existing code are required to display window-level context-sensitive help in the `MainWindow`. The following invocation would display the window-level help in a main window.

```
JTextArea newText = new JTextArea();
hb.enableHelp(newText, “debug.overview”, hs);
    ...
rootpane = frame.getRootPane();
mainHelpBroker.enableHelpKey(rootpane, “top”, null);
```



JavaHelp can be used in a wide variety of ways. This is one of those ways.

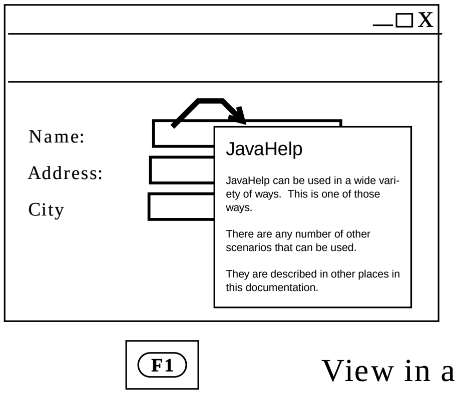
There are any number of other scenarios that can be used.

They are described in other places in this documentation.

View in a Main Window

The following invocation would display the window-level help in a popup as illustrated.

```
JTextArea newText = new JTextArea();
hb.enableHelp(newText, “debug.overview”, hs);
    ...
rootpane = frame.getRootPane();
mainHelpBroker.enableHelpKey(rootpane, “top”, null, “javax.help.Popup”, null);
```



JavaHelp can be used in a wide variety of ways. This is one of those ways.

There are any number of other scenarios that can be used.

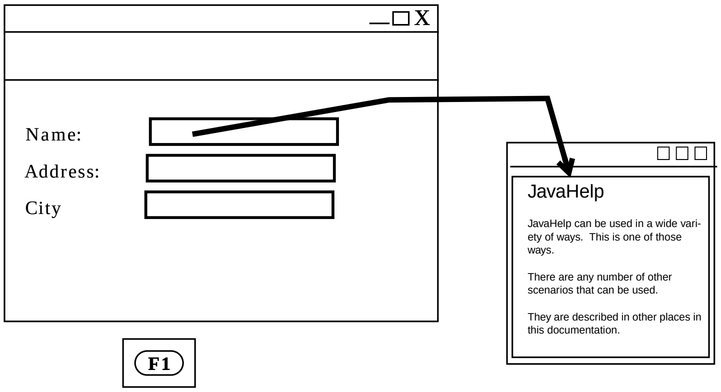
They are described in other places in this documentation.

View in a Popup

The following invocation would display the window-level help in a secondary window as illustrated.

```
JTextArea newText = new JTextArea();
hb.enableHelp(newText, “debug.overview”, hs);
    ...
rootpane = frame.getRootPane();
```

```
mainHelpBroker.enableHelpHey(rootpane, “top”, hs, “javax.help.SecondaryWindow”, “mainSW”);
```



JavaHelp can be used in a wide variety of ways. This is one of those ways.

There are any number of other scenarios that can be used.

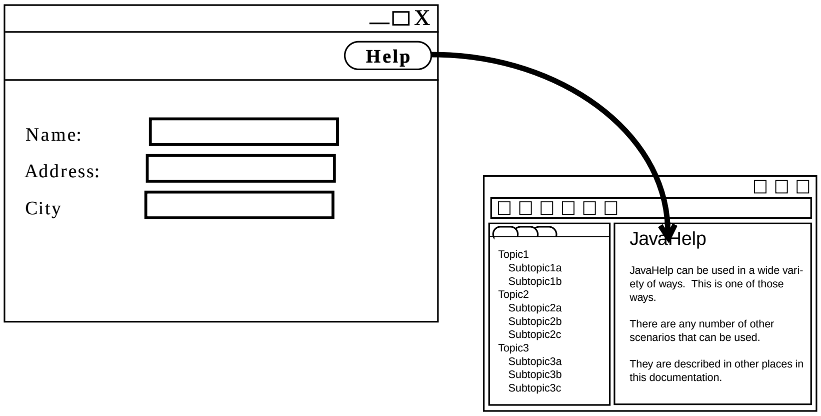
They are described in other places in this documentation.

View in a Secondary Window

8.4.3 User initiated context-sensitive help

No changes to existing code are required to display user initiated context-sensitive help in the `MainWindow`. The following invocation would display the user initiated help in a main window.

```
JButton helpbutton = new JButton(“Help”);
mainHelpBroker.enableHelpOnButton(helpbutton, “browse.strings”, null);
```



JavaHelp can be used in a wide variety of ways. This is one of those ways.

There are any number of other scenarios that can be used.

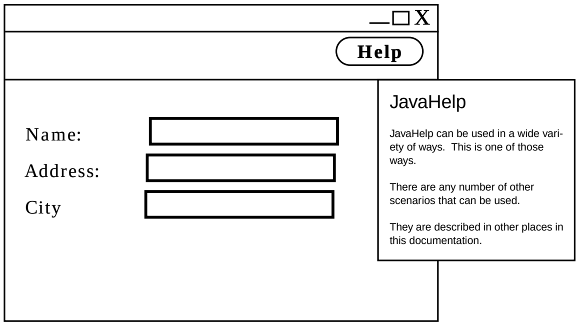
They are described in other places in this documentation.

View in a Main Window

The following invocation would display the user initiated help in a popup as illustrated.

```
JButton helpbutton = new JButton(“Help”);
mainHelpBroker.enableHelpOnButton(helpbutton, “browse.strings”, null,
```

```
                                   “javax.help.Popup”, null);
```



JavaHelp can be used in a wide variety of ways. This is one of those ways.

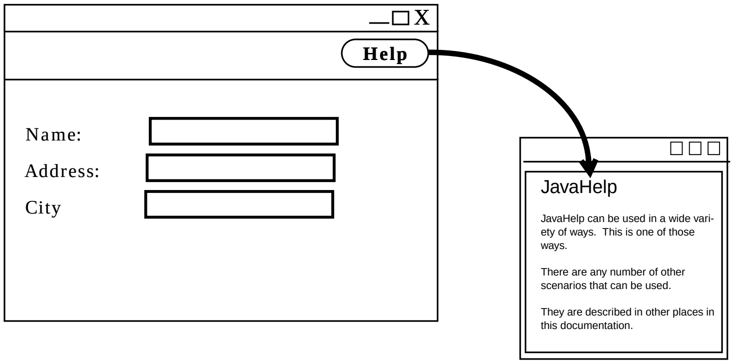
There are any number of other scenarios that can be used.

They are described in other places in this documentation.

View in a Popup

The following invocation would display the user initiated help in a secondary window as illustrated.

```
JButton helpbutton = new JButton(“Help”);
mainHelpBroker.enableHelpOnButton(helpbutton, “browse.strings”, null,
                                  “javax.help.SecondaryWindow”, “mainSW”);
```



JavaHelp can be used in a wide variety of ways. This is one of those ways.

There are any number of other scenarios that can be used.

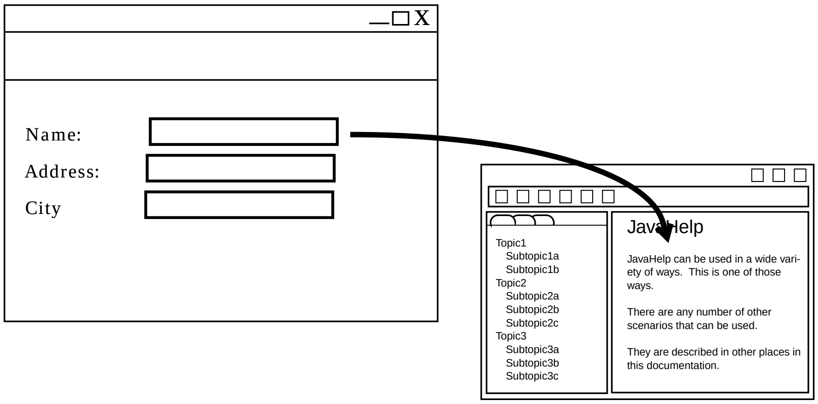
They are described in other places in this documentation.

View in a Secondary Window

8.4.4 System initiated context-sensitive help

No changes to existing code are required to display system initiated context-sensitive help in the `MainWindow`. The following invocation would display the system-initiated help in a main window.

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



JavaHelp can be used in a wide variety of ways. This is one of those ways.

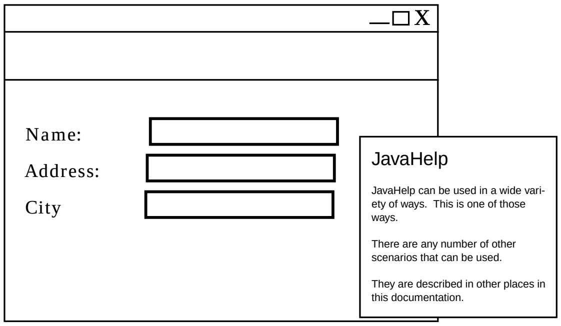
There are any number of other scenarios that can be used.

They are described in other places in this documentation.

View in a Main Window

The following invocation would display the system initiated help in a popup as illustrated.

```
Popup popup = (Popup)Popup.getPresentation(mainHS, null);
popup.setInvoker(component);
popup.setCurrentID(helpID);
popup.setDisplayed(true);
```



JavaHelp can be used in a wide variety of ways. This is one of those ways.

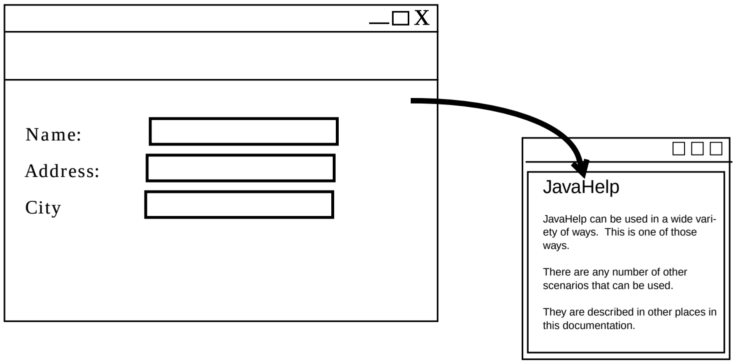
There are any number of other scenarios that can be used.

They are described in other places in this documentation.

View in a Popup

The following invocation would display the system initiated help in a secondary window as illustrated.

```
mainHelpBroker.showID(helpID, “javax.help.SecondaryWindow”, “main”);
```



JavaHelp can be used in a wide variety of ways. This is one of those ways.

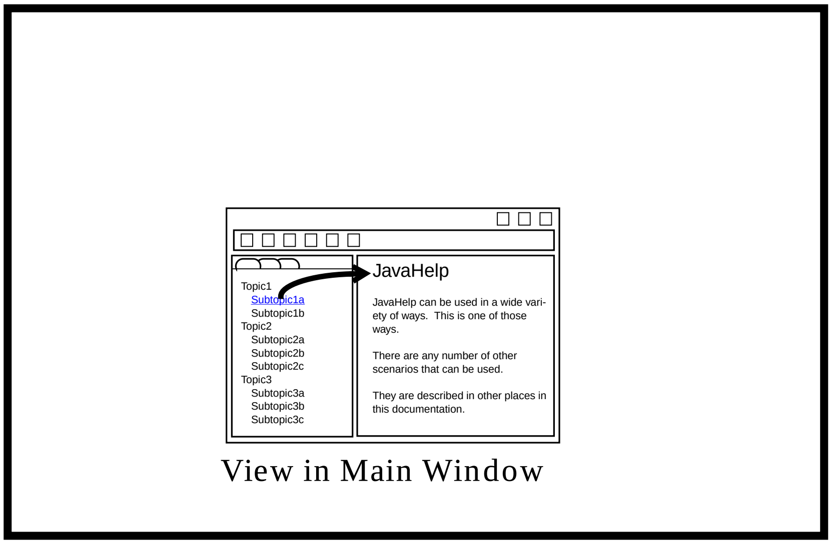
There are any number of other scenarios that can be used.

They are described in other places in this documentation.

View in a Secondary Window

8.4.5 Navigator

No changes to exist code are required to display an item in the main window. Selecting an item in the navigator would cause the content pane to be updated as illustrated.



JavaHelp can be used in a wide variety of ways. This is one of those ways.

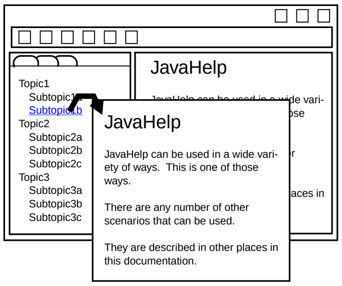
There are any number of other scenarios that can be used.

They are described in other places in this documentation.

View in Main Window

The following `tocitem` would display the presentation in a popup window:

```
<tocitem target="platform" image="image document.gif” presentation=”javax.help.Popup”>
        JavaHelp platforms
</tocitem>
```



JavaHelp can be used in a wide variety of ways. This is one of those ways.

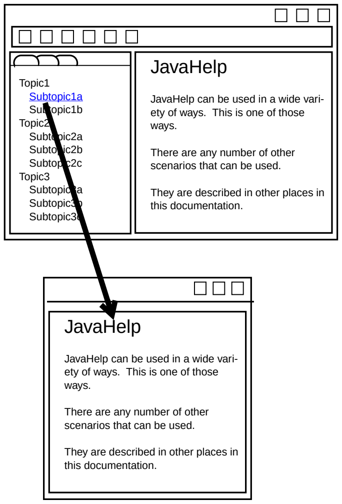
There are any number of other scenarios that can be used.

They are described in other places in this documentation.

View in Popup

Similarly, the following example would display the presentation in a secondary window.

```
<tocitem target="platform" image="image document.gif”
    presentationtype=”javax.help.SecondaryWindow” presentation=”mainSW”>
         JavaHelp platforms
</tocitem>
```



JavaHelp can be used in a wide variety of ways. This is one of those ways.

There are any number of other scenarios that can be used.

They are described in other places in this documentation.

JavaHelp can be used in a wide variety of ways. This is one of those ways.

There are any number of other scenarios that can be used.

They are described in other places in this documentation.

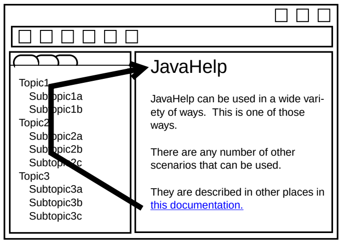
View in Secondary Window

8.4.6 Viewer

Activation of help content from a viewer was not specified in the V1.0 specification but was supported in the reference implementation with through lightweight-components. The content viewer could display help in the viewer, a named secondary window or a popup as illustrated.

The following illustrates the content being displayed in a viewer. It uses the standard `<a href>` in HTML:

```
<A HREF="sbrowse/sbrowse.html">Browsing Source</A>
```



JavaHelp can be used in a wide variety of ways. This is one of those ways.

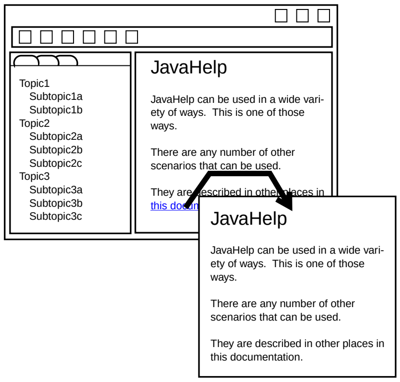
There are any number of other scenarios that can be used.

They are described in other places in this documentation.

View in Main window

The following illustrates the content being displayed in a popup. It uses the object tag from the reference implementation.

```
<OBJECT CLASSID="java:com.sun.java.help.impl.JHSecondaryViewer">
<param name="content" value="popup_gloss.html">
<param name="viewerActivator" value="javax.help.LinkLabel">
<param name="viewerStyle" value="javax.help.Popup">
<param name="viewerSize" value="400,250">
<param name="text" value="popup windows">
<param name="textColor" value="blue">
<param name="viewerName" value="glossary">
</OBJECT>
```



JavaHelp can be used in a wide variety of ways. This is one of those ways.

There are any number of other scenarios that can be used.

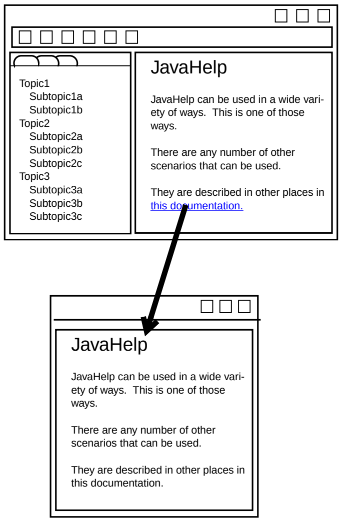
They are described in other places in this documentation.

View in Popup

The following illustrates the content being displayed in a popup. It uses the object tag from the reference implementation.

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

```
<param name="content" value="demo.html">
<param name="viewerName" value="demo">
<param name="viewerLocation" value="500,220">
<param name="viewerSize" value="500,500">
</object>
```



JavaHelp can be used in a wide variety of ways. This is one of those ways.

There are any number of other scenarios that can be used.

They are described in other places in this documentation.

JavaHelp can be used in a wide variety of ways. This is one of those ways.

There are any number of other scenarios that can be used.

They are described in other places in this documentation.

View in Secondary Window

No changes to the specification will be made to support displays to named secondary windows, popups or custom presentations.

9 Toolbar

A supplement of controlling presentations (see section 3.2.7 on page 21) is the ability to define or customize the toolbar in a JHelp component. In order to accommodate the new HelpAction in the HelpSet-Presentation section of a HelpSet file a new HelpAction class has been added.

- *HelpActions could be used in Menus, however, they are not part of the reference implementation and are not specified in the HelpSet.*

9.1 HelpAction Interface

The `HelpAction` interface defines common behaviors for help actions. One use of help actions would be the handling of common behavior for buttons included in a `JHelp` component's toolbar.

Method	Description
`get/putValue(String key, Object value)`	Get/set on of the object's properties using the associated Key.
`is/setEnabled(boolean)`	is/set the HelpAction enabled.
`get/setControl(Object)`	Get/set the control Object for the HelpAction.

9.2 AbstractHelpAction Class

The `AbstractHelpAction` class is a default implementation of the `HelpAction` interface.

Constructor	Description
`AbstractHelpAction(Object control, String name)`	Creates an AbstractHelpAction

The constructor takes two argument: control and name. The control is an object that would exhibit some type of control of the action. In the reference implementation the control object would be the `JHelp` component.

9.3 HelpAction Extensions

Extensions of the `AbstractHelpAction` class are required set the help action name, implement an appropriate event listener and return an appropriate image. The name is used to provide locale specific tooltip for the button. It is possible for an extension to set the name internally in the extension's constructor.

Extensions must implement an appropriate event listeners based on the implementation GUI and action. For instance, the reference implementation GUI is AWT or Swing the event listeners would be an AWT event listener. A button implementing an ActionListener would implement the method `public void actionPerformed(ActionEvent e)`. Similarly a button implementing a MouseListener would implement the MouseListener methods. The implementation then determines the type of listener to add by inspecting the class for a given EventListeners.

Additionally extensions must implement a method to retrieve implementation GUI specific image if no imageID has been set. For instance, in the reference implementation is Swing based so an appropriate image would be `javax.swing.ImageIcon` and the appropriate method would be `getImageIcon()`.

9.4 Supplied AWT/Swing HelpActions

A set of default HelpActions are included in the specification. This extensions are part of AWT/Swing part of the specification and are specific to the AWT/Swing GUI.

Name	Description
`BackAction`	An action to move to the previous page in the content viewer.
`ForwardAction`	An action to move to the next page in the content viewer.
`PrintAction`	An action to print.
`PrintSetupAction`	An action to print through the print setup.
`HomeAction`	An action to load the HomeID in the content viewer.
`ReloadAction`	An action to reload the current document in the content viewer.
`SeparatorAction`	An non-action that creates a separator between the actions.

Valid properties Awt/Swing HelpActions for the method `get/putValue` include `icon` for image used in the button, `tooltip` for the tooltip text, `access` for accessiblity name.

10 Context Sensitive Help

10.1 Context-Sensitive Help

Context-sensitive help in the JavaHelp system is organized around the notion of the ID-URL map referred by the `<map>` section of a HelpSet file. The key concept is that of the Map.ID which is comprised of a String / HelpSet instance pair. The String is intended to be unique within the local map of the HelpSet. This is very important when considering HelpSet merging, otherwise IDs would be required to be unique over all HelpSets that might ever be merged.

There are three parts involved in assigning Context Sensitive Help to an application:

1. Define the appropriate String ID-URL map,
2. Assign an ID to each desired visual object,
3. Enable some user action to activate the help.

10.1.1 Defining the ID-URL map

The Map interface provides a means for associating IDs (*HelpSet.string*) with URLs. One such map is constructed from one or more map files that provide a simpler "String ID" to URL mapping, with the HelpSet being given either explicitly or implicitly.

JavaHelp has two classes that implement the Map interface: FlatMap and TryMap. A `FlatMap` does not support nesting of other maps into it, while a `TryMap` does. A FlatMap is a simple implementation while TryMap should support inverse lookups (for example, `getIDFromURL`) more efficiently. The implementation of `TryMap` JavaHelp 1.0 is not particularly efficient.

Both `FlatMap` and `TryMap` have public constructors. The constructor for `FlatMap` takes two arguments: the first one provides a URL to a property file providing a list of String and URL pairs; the second argument is a HelpSet. The HelpSet is used together with each String-URL pair to create the actual `Map.ID` objects that comprise the `FlatMap`. The constructor for `TryMap` has no arguments: the Map is created empty and other `Maps` are added (or removed) from it.

The `Map` interface can also be implemented by some custom class. One such class could be used to, for example, programatically generate the map.

10.1.2 Assigning an ID to Each Visual Object

The next step is to assign to each desired GUI object an ID that will lead to the desired help topic. There are two mechanisms involved: an explicit ID, either a plain String, or a `Map.ID`, is assigned to the GUI object; and there is a rule that is used to infer the `Map.ID` for an GUI object based on its container hierachy.

The two basic methods to assign IDs are setHelpIDString(Component, String) and setHelpSet(Component, String). If both are applied to a Component, then a `Map.ID` is assigned to that Component. If only `setHelpIDString()` is applied, then the `HelpSet` instance is obtained implicitly, as indicated later. A method overload is provide for `MenuItem` objects.

These methods take a Component as an argument. The implementation may vary depending on whether the argument is a JComponent or a plain AWT Component.

The methods getHelpIDString(Component) and getHelpSet(Component) recursively traverse up the container hierachy of the component trying to locate a Component that has been assigned a String ID. When found, the methods return the appropriate value. As before there is also an overloaded method for `MenuItem`.

10.1.3 Enabling a Help Action

The final step is to enable for some action to trigger the presentation of the help data. CSH currently provides several `ActionListener` classes that can be used:

Name	Description
`DisplayHelpFromFocus()`	Locate the Component currently owning the focus, then find the ID assigned to it and present it on the HelpBroker. This is to be used by "Help" keys.
`DisplayHelpAfterTracking()`	Start tracking events until a mouse event is used to select a Component, then find the ID assigned and present it. This is to be used by a "What's this" type of interface.
`DisplayHelpFromSource()`	Find the ID assigned to the source of the action event and present it.

In addition, `HelpBroker` also provides some convenience methods that interact with these ActionListeners:

Name	Description
`enableHelpKey(root, stringID, helpSet)`	Set the ID and helpset of root which will act as the default help to present, then register an appropriate ActionListener to be activated via the "Help" key. DefaultHelpBroker uses `CSH.DisplayHelpFromFocus` as the ActionListener.
`enableHelp(Component, stringId, helpSet)`	Set the ID and HelpSet to the component. This information is usually recovered either using the "Help" key or through the `DisplayHelpAfterTracking` class.

`enableHelpOnButton(Component, stringId, helpSet)`	Set the ID and HelpSet to the component, which must be a "Button". When the button is "pressed" the Help information given in the arguments will be presented.

Since these methods are from a specific HelpBroker, if a HelpSet is not associated with the GUI object then the HelpSet of the HelpBroker will be used automatically.

10.1.4 Dynamic ID Assignment

For certain objects having a single ID per object is not sufficient. There needs to be a way to programatically determine the ID based on cursor position, selection, or some other mechanism inherent to the object. For example a Canvas might determine the ID based on the object currently selected on the canvas or alternatively from the mouse cursor position.

The following APIs have been added to CSH to support dynamic ID assignment:

Name	Description
`addManager(CSHManager)`	Registers the specified manager to handle dynamic CSH.
`addManager(index, CSHManager)`	Registers the specified manager to handle dynamic CSH at the specified position in the list of managers.
`getManager(index)`	Returns the manager at the specified position in list of managers.
`getManagerCount()`	Returns the number of managers registered.
`getManagers()`	Returns array of managers registered.
`removeAllManagers()`	Remove all of the dynamic CSH managers.
`removeManager(CSH.Manager)`	Remove the specified manager from the list of managers.
`removeManager(index)`	Remove the manager at the specified position in the list of managers.

Additionally a new interface has been defined in CSH.Manager:

Name	Description
`getHelpSet(Object, AWTEvent)`	Returns String representing the MAP.ID of the object based on the AWTEvent

`getHelpIDString(Object, AWTEvent)`	Returns the HelpSet of the object based on the AWTEvent

Instances of the CSHManager work as filters. CSH.getHelpIDString(comp) and CSH.getHelpSet(comp) are required call each registered CSH.Manager's getHelpIDString or getHelpSet methods. If the CSHManager doesn't want to handle the component it returns null. If no CSHManager provides a HelpSet or HelpIDString for the component the CSH methods would use the static HelpSet and HelpIDString (see section 10.1.2 on page 64 for more details on static HelpSet and HelpIDString). As with statically defined HelpSet and HelpIDString a failure in request for HelpSet and HelpIDString is propagated to the component's parent.

10.1.4.1 Example Usage

An example usage for components with dynamically assigned HelpSet or dynamically generated HelpIDString is below:

```
class MyCSHManager implements CSHManager {
    HelpSet hs;
    JEditorPane editor;
    MyCSHManager(JEdtiorPane editor, HelpSet hs) }
        this.editor = editor;
        this.hs = hs;
    }
    public HelpSet getHelpSet(Object comp) {
        if (comp == editor) {
            return hs;
        }
        return null;
    }
    public String getHelpIDString(Object comp) {
        if (comp == editor) {
            return getHelpIDFromCaretPostion(editor);
        }
        return null;
    }
}
```

The CSHManager is added to the CSH list of managers as follows:

```
CSH.AddCSHManager(new MyCSHManager(editor, hs));
```

10.2 Help Support for JDialogs

It is often useful to associate help information with dialog boxes using a Help button. Ideally the standard `javax.swing.JOptionPane` would have direct support for this but, due to timing constraints this was not possible. Expect full support for this feature in a forthcoming version of Swing.

11 Content Search

11.1 Search API

JavaHelp provides full-text searching of help topics. Development of full-text searching raised interesting questions, both in the implementation and in the specification. For example, whether the search database is created before or during queries, and how the format of the search database is specified.

The search API `javax.help.search.*` can be used to create and query the search database. The default NavigatorView, SearchView knows how to interact with any subclass of SearchEngine. Similarly the search database can be created through the IndexBuilder class.

One of the benefits of the `javax.help.search` API is that it enables the use of search engines that require moderately complex database formats without the difficult and constraining task of specifying these formats in full. One such search engine is the one provided in Sun's reference implementation section Appendix B on page 104.

The intention of the `javax.help.search` package is to provide insulation between client and customers of a full-text search database in the context of the `javax.help` package. It is important to emphasize that although the `javax.help.search` API is intended to be of general applicability, it is not intended to be a replacement for more powerful query mechanisms.

11.2 Search Database Creation

Search databases are created through instances of `IndexBuilder`. The parsing of each file is specific to its MIME content; this is encoded in the notion of an IndexerKit. An indexer kit provides a `parse()` method that knows how to parse the specific MIME type and call back into the `IndexBuilder` instance to capture the information of this source.

When capturing search information there are a number of parameters that you can configure using a ConfigFile:

- 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

11.2.1 Stopwords

You can direct the JavaHelp system full-text search indexer to exclude certain words from the database index--these words are called *stopwords*. The default stopwords are:

```
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   shouldso    some
than   that  the   them  then  there these   this
those   thoughto   too   us    was   way   we
what   when  where which   who why   will  would
yes    yet   you
```

11.2.2 ConfigFile Directives

A config file may contain the following directives

Directive	Description
`IndexRemove` *path*	Remove a *path* that is a prefix to the given files
`IndexPrepend` *path*	Prepend *path* to the names of the given files.
`File` *filename*	Request that the *filename* be processed
`StopWords` *word, word, word...*	Set the stopwords to the ones indicated
`StopWordsFile` *filename*	`StopWordsFile` must contain a list of stopwords, one stopword per line.

11.3 Search Database Use

The `javax.help.search` package is used in JavaHelp 1.0 by `SearchView`. This view expects an *engine* property that specifies the name of the subclass of `javax.help.search.SearchEngine` to use when making queries. The default value of this property is `com.sun.java.help.search.SearchEngine`.

The steps involved in using the search engine from a `SearchView` are:

- A SearchView is instantiated, for example, when the default HelpBroker for the HelpSet is created.
- When the first query is made, the *engine* property of the view is obtained to determine what `SearchEngine` to instantiate. The *data* and other attributes are passed to this instance.
- For a query, a `SearchQuery` instance is obtained, then the query is passed to it.
- Hits found are either obtained directly, or they are generated as events.

More details may be added in the next iteration of the specification.

12 Merge

12.1 Introduction

JavaHelp provides a mechanism for *merging* HelpSets. Constituent HelpSets can be dynamically removed from the merged HelpSet, even while the merged HelpSet is displayed. When HelpSets are merged there is always a *master* HelpSet into which other HelpSets are merged.

In addition, a HelpSet file can use the `<subhelpset>` tag to statically include HelpSets (see section 3.2.6 on page 20), this behavior is identical to adding the subhelpset to the enclosing HelpSet, except that if the subhelpset file does not exist, it is ignored.

Here are some examples where merging might be appropriate:

- An application suite may be comprised 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.
- A Builder tool uses JavaBeans to construct programs. Each JavaBean provides help information about its functionality. The help information of the constituent JavaBeans can be listed in the TOC, in the index, and be accessible to searches.
- When JavaBeans are used to dynamically extend the functionality of an application (sometimes this functionality is described as *plug-in*) the JavaBeans contain help information that conforms to the nature of the application. This help information can be meaningfully merged before being presented to the user.

12.2 Merging Rules

The default merging rules depend on the view mergetype (see section 3.2.5 on page 19). There are four mergetypes in JavaHelp:

javax.help.SortMege	Collate with the existing view data according to HelpSet’s locale collation rules.
javax.help.Append	Add to the end of the existing view data.
javax.help.Unite-AppendMerge	Unite any elements, including sub-elements, at the same level in the merged HelpSet to elements, including sub-elements, with the same name in the initial HelpSet. Append remaining elements in the merged HelpSet at the end of the initial HelpSet. Add to the end of the existing view data.
javax.help.NoMerge	No merging is done. Only valid on a View.

Each view type will determine it own default merging type and will specify which, if any, of the merging rules it will support. A view type may provide no support for merging.

12.3 The API

The basic API comprises the HelpSet.add(HelpSet) method, and its corresponding HelpSet.remove(HelpSet) method. These methods fire HelpSetEvent events to the HelpSetListeners that have registered interest in them. This is how the ComponentUIs for TOC, Index, and Search views are notified of these changes and react to them.

When a HelpSet A is added to a HelpSet B, all the views in A are compared to the views in B; if a view in A has a name that is the same as another view in B, then it is considered for merging into B, otherwise it is not.

When considering merging a view V_a into a V_b the following happens:

- The navigator N_b of V_a is obtained.
- Nb.canMerge(Va) is invoked to determine if the views
- can be merged.
- If then can be merged, then Nb.merge(Va) is invoked.

If later the HelpSet A is removed from HelpSet B:

- Nb.remove(Va) will be invoked.

The merging of the view data API comprises the abstract class Merge, consisting of a Merge(NavigatorView, NavigatorView) constructor and Merge.processMerge(TreeNode) method, MergeDefaultFactory.getMerge(NavigatorView, NavigatorView), and a set of merge utilties in MergeUtilities. Each mergetype (see section 12.2 on page 70) will implement a concrete class extending Merge.

When the Nb.merge(Va) is invokedMergeDefaultFactory.getMerge is invoked returning a Merge object. Invoking Merge.processMerge on the toplevel node will merge the data of the slave into the master. Merge.processMerge can recursively called to support changes of mergetypes on each data item. The master view data viewtype will supercede the viewtype settings in a slave in the event of a conflict.

12.4 Merging TOCs

`TOCView` and `JHelpTOCNavigator` implement a merging rule that allows any TOCView with the same name to be merged. The resulting presentation adds the new TOC data as the last subtree of the top level of the original TOC.

A TOCView may have no <data> tag; such a view shows as an empty tree. This is useful for what is sometimes called "dataless" master views into which other views can merge.

The default merge type for a TOCView is append. This will provide backward compatible support for V1.0 implementations. TOCView supports sort, append, and unite-append merge types.

12.5 Merging Indices

`IndexView` and `JHelpIndexNavigator` implement a merging rule that allows any IndexView with the same name to be merged. The resulting presentation adds the new index data as the last subtree of the top level of the original index. No attempt to sort the data is provided in the standard types.

An IndexView may have no <data> tag; such a view shows as an empty tree. This is useful for what is sometimes called "dataless" master views into which other views can merge.

The default merge type for IndexView is append. This will provide backward compatible support for V1.0 implementations. IndexView supports sort, append, and unite-append merge types.0

12.6 Merging Glossaries

The merging rule for Glossaries are the same as those of Indices section 12.5 on page 72.

12.7 Merging Favorites

The default merge type for FavoritiesView is none as favorites are stored in a single file in the user’s directory.

12.8 Merging Full-Text Search Databases

`SearchView` and `JHelpSearchNavigator` implement a merging rule that allows any SearchView with the same name to be merged. The resulting presentation adds the `SearchEngine` from the new view to the previous list--query results from all the `SearchEngines` are collated and presented together.

A SearchView may have no <data> tag; such a view produces no matches against any queries.

The default merge type for a SearchView is sort. This will provide backward compatible support for V1.0 implementations. Append and unite-append merge types are not supported in SearchView.

12.9 Overriding Mergetype

The view mergetype can be overridden with the mergetype attribute on elements of certain view types. Both Table of Contents and Index support overriding the default mergetype by setting the mergetype attribute on <tocitem>, or <indexitem> tags.

If the view supports nesting of elements, the override applys to nested elements as well. If duplicate entries exist in both views, the mergetype specified in the first view will override the second view.

12.10 Examples

The following examples illustrate how the various merge types work

12.10.1 Example: Append Merge

Append merge appends the second view data to the first view. It places a title determined from the view's HelpSet <title> tag as the top level entry and indents the view's data under that title.

Append is useful when the desired result is to combine two views together into one presentation, but still maintain the overall hierarchy of each view. Unlike sort, duplicate entries are retained.

JWS TOC

```
Tutorial and Examples
    Introducing the Java WorkShop
    Tutorial One: Creating the Blink Project
    Tutorial Two: Editing Project Attributes
    Tutorial Three: Fixing Errors in Source Code
JavaBeans
    Beans In Java WorkShop
    Tips on Using Beans Effectively
Visual Java GUI Builder
    GUI Builder Palette
    Laying Out the Interface
The Help Viewer
    Overview
    Menus Descriptions
    Shortcut Keys
    Searching for Topics
The Web Browser
    Overview
    Using the Web Browser
```

JS TOC

```
Java Studio Windows
        Main window
        Design Window
        GUI Window
Java Studio Designs
        Creating a Design
        Adjusting the Layout of a Design
        Debugging a Design
        Saving and Loading a Design
        Generating a Packaged Design
        Printing a Design
The Help Viewer
```

```
        Overview
        Menus Descriptions
        Shortcut Keys
        Searching for Topics
```

Merged result using append

The output using append is as follows:

```
Tutorial and Examples
    Introducing the Java WorkShop
    Tutorial One: Creating the Blink Project
    Tutorial Two: Editing Project Attributes
    Tutorial Three: Fixing Errors in Source Code
JavaBeans
    Beans In Java WorkShop
    Tips on Using Beans Effectively
Visual Java GUI Builder
    GUI Builder Palette
    Laying Out the Interface
The Help Viewer
    Overview
    Menus Descriptions
    Shortcut Keys
    Searching for Topics
The Web Browser
    Overview
    Using the Web Browser
Java Studio Windows
        Main window
        Design Window
        GUI Window
Java Studio Designs
        Creating a Design
        Adjusting the Layout of a Design
        Debugging a Design
        Saving and Loading a Design
        Generating a Packaged Design
        Printing a Design
The Help Viewer
        Overview
        Menus Descriptions
        Shortcut Keys
        Searching for Topics
```

12.10.2 Example: Sort Merge

Sort merge collates at each level of the combined view according to the collation rules of the HelpSet locale. Duplicate entries where the name and the id associated with the entries are the same are ignored. The entry, "The Help Viewer" and all of its subentries, is an example of how duplicate entries in separate views are handled.

If the entry’s name is the same as another entry at a given level but the id associated with the entry is different, then both entries are printed with the HelpSet title (<title>)

applied to the end of the name as a distinquishing characteristic. In the example below the Edit and File Menu entries point to different ids. They have been distinquished with a "(Java Workshop)" and a "(Java Studio)".

Sort merge is useful when you have information, such as an Index, that is collated. It is not useful when you have information that is in a non collated hierarchical form, such as a TOC.

JWS Index

```
Menus
        File Menu
        Edit Menu
        Build Menu
        Debug Menu
        Help Menu
Toolbars
        Main Toolbar
        Edit/Debug Toolbar
```

JS Index

```
Developer Resources
Examples
        Step-by-step Example
        List of Additional Examples
Menus
        File Menu
        Edit Menu
        View Menu
        Help Menu
Toolbars
        Main Toolbar
        Composition Toolbar
```

Merged result using sort

The output of the sort merge is a follows:

```
Developer Resources
Examples
        List of Additional Examples
        Step-by-step Example
Menus
        Build Menu
        Debug Menu
        Edit Menu (Java Workshop)
        Edit Menu (Java Studio)
        File Menu (Java Workshop)
        File Menu (Java Studio)
        Help Menu (Java Workshop)
        Help Menu (Java Studio)
```

```
        View Menu
Toolbars
        Main Toolbar (Java Workshop)
        Main Toolbar (Java Studio)
        Edit/Debug Toolbar
        Composition Toolbar
```

12.10.3 Example: Unite-Append Merge

Unite-append preserves the hierarchy of the masterview. If the master view is a data less view the hierarchy from the first view merged is preserved. At any level it merges entries from the second view with the master view if the entry’s name is the same.

Entries that don’t have the same name are appended to the master view at their respective levels. If the entry’s name is the same but the id is different both entries are displayed and some sort of distinquishing characteristic is applied to the end of the name.

Unite-append is useful to maintain the hierarchy of the master view. That master view may be a "template" for merging the others against. Entries not found in the master view are appended at the end. The hierachy of the second view is maintained subordinate to the master view.

The animal TOC

```
Description
Habitat
Pictures
```

A Wombat

```
Description
        What’s a Wombat
Habitat
        Where a Wombat lives
Pictures
        Cute Wombats
Sounds
        Fierce Wombats
```

A Water Rat

```
Description
        What’s a Water Rat
Habitat
        Where a Water Rat lives
Pictures
        Water Rats
Sounds
        Singing Water Rats
```

Merged result using unite-append

```
Description
       What’s a Wombat
```

```
               What’s a Water Rat
        Habitat
               Where a Wombat lives
               Where a Water Rat lives
        Pictures
               Cute Wombats
               Water Rats
        Sounds
               Fierce Wombats
               Singing Water Rats
```

13 JavaHelp Class Structure

13.1 Packages

JavaHelp V2.0 is a optional package for Java 2. The API is defined in the `javax.help` package, with the exceptions of the search API classes, which are defined mainly in the `javax.help` package, but other packages are also involved. The complete list is:

Package	Description
javax.help	Main package
javax.help.event	Event & Listener classes
javax.help.plaf	Interface to the ComponentUI classes
javax.help.plaf.basic	Basic look and feel; currently no specific PLAF classes are needed
javax.help.resources	Localization classes.
javax.help.tagext	JSP tag extension classes.
javax.help.search	search classes.

An implementation of the extension may also include some implementation classes that are not intended to be used directly. The Reference Implementation also includes additional classes of utility to Help authors.

13.2 API Structure

This section describes the general principles behind the API classes. More details are available in the javadoc information on the classes. The reference implementation also provides code fragments exemplifying the use of these classes.

As indicated in Overview.html, the API classes in javax.help are conceptually structured in several collections. The different collections are addressed to different tasks and users. The boundaries between some of these collections are not sharp, but the classification helps to reduce the number of concepts, and actions, needed to perform simple tasks.

- Basic Content Presentation
- Complete Access to JavaHelp Functionality
- Swing classes
- Full-Text Search
- JSP tag extensions

13.2.1 Basic Content Presentation

Some applications only are interested in presenting some help information to the user, minimizing the interaction between the help author and the application developer. The basic actions to perform are:

- Locating a HelpSet, perhaps after localization;
- Reading that HelpSet, including any related data, like Map files, TOCs, Indices, and Search database; and
- Visually presenting this HelpSet.

The abstraction of a HelpSet is javax.help.HelpSet, while the abstraction of its visual presentation is javax.help.HelpBroker. A `HelpBroker` provides for some interaction with the presentation regardless of the actual visual details; the default presentation is `DefaultHelpBroker`. An application can provide on-line help using only these two classes.

Sub-HelpSets listed in the HelpSet file using the <subhelpset> tag will be merged automatically before presenting them to the user.

These two classes (an ancillary classes, like Exception classes) do not have any dependency on Swing for their definition, although `DefaultHelpBroker` depends on Swing for its implementation.

13.2.2 Detailed Control and Access

The `HelpBroker` interface provides substantial control of the presentation of a HelpSet, without leaking unwanted GUI details of the presentation. For example, this interface can be used to interact with the two-pane default presentation of the reference implementation, as well as to interact with some presentation embedded within the application. Additionally, since the HelpBroker does not use any Swing types or concepts, it does not require Swing for its implementation. But some applications will want access to such details as the map from ID to URLs. JavaHelp provides classes for this.

13.2.3 Extensibility

Content extensibility is described through a `NavigatorView` which provides access to some context information plus a way of presenting this information. `TOCView`, `IndexView`, `GlossaryView`, `FavoritiesView` and `SearchView` are standard views for Table Of Contents, Index, and full-text search.

The standard views yield standard `JHelpTOCNavigator`, `JHelpIndexNavigator`, and `JHelpSearchNavigator` Swing components. The standard views also provide access to the content; this access uses subclasses of `TreeItem`.

New views can be added; for instance a new TOC presentation can be obtained by subclassing TOCView and just changing the JHelpNavigator returned by it. Another view may keep the same JHelpNavigator but use a format for the encoding of the view data (perhaps even generating the data dynamically); this is done by redefining the getDataAsTree method. The presentation of new Views can be derived from the standard ones by subclassing.

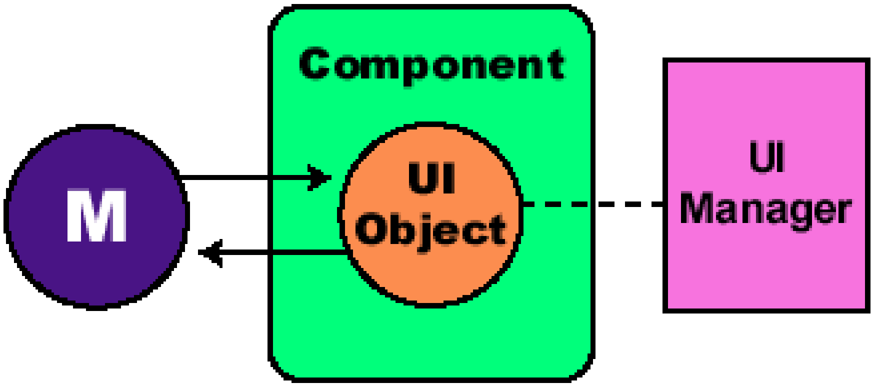
13.2.4 Swing components

JavaHelp provides a collection of Swing components that are used to implement the DefaultHelpBroker and can also be used directly, as in *embedded help*. The components follow the standard MVC from Swing. There are two main models: `HelpModel` and `TextHelpModel`.

`HelpModel` models changes to the location within a HelpSet; components that want to respond to these changes should listen to events originating within the model - this is how synchronized views work. The location within the model is represented by objects of type `Map.ID`; these correspond to a String (an ID), and a HelpSet providing context to that ID. A HelpSet needs to be explicitly given (in general) because of the ability of merging HelpSets. `TextModel` provides additional information when the content is textual. A TextModel can queried for the current *highlights*, which a client may present visually. The `DefaultHelpModel` is the default model implementing both models.

`JHelpContentViewer` is the Swing component for the content, while context corresponds to several subclasses of `JHelpNavigator`. `JHelp` is a common grouping of these classes into synchronized views of content.

The basic structure of the Swing classes is shown in the next figure; for additional information about the Swing classes check the Swing Connection home page



A Swing control acts as the main interface to developers. All `ComponentUI` objects for a particular look and feel are managed by a JFC object called `UIManager`. When a new Swing component is created, it asks the current `UIManager` to create a `ComponentUI` object. Vendors or developers can ship different `ComponentUI's` to suit their specific needs.

A Swing control then delegates the tasks of rendering, sizing and performing input and output operations to the `ComponentUI`. The `ComponentUI's` `installUI` and `deinstallUI` methods add behavior and structure to the raw Swing component by adding listeners, setting the layout manager, and adding children.

The Swing model defines the component's non-view-specific state. The Swing component communicates changes to the model and listens (through listeners) to the model for changes. Finally, the model provides data to the `ComponentUI` for display.

The ComponentUI objects in the JavaHelp Swing classes are currently fully defined in terms of the other components, hence, there are only `javax.help.plaf.basic` classes, and none of the other PLAF packages are needed.

13.2.5 Context Sensitive Help

JavaHelp supports a `Map` between identifiers and URLs. `FlatMap` and `TryMap` are two implementations; sophisticated users can provide their own implementations to satisfy different requirements (for example, the map data may be generated dynamically). The main class used to associate specific content with graphic objects is `CSH`.

13.2.6 Search

JavaHelp supports a standard full-text search view and navigator. The view interacts with a search engine through the types in the `javax.help.search` package. The reference implementation provides a search engine implementing these interfaces but others can also be used; the specific search engine used is part of the information given to the search view. By doing this separation we provide the capability of full-text searching while not imposing specific formats.

The search package has not conceptual dependencies on any other portions of JavaHelp, and it can be used independently. The reference implementation provides one such implementation packaged in a JAR file that depends only on the basic platform.

Apendix A
JavaHelp 2.0 - Scenarios

A.1 Introduction

This document contains a number of scenarios that illustrate 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 flexibility and extensibility of the JavaHelp system.

Scenarios are presented in four areas:

Invocation Mechanisms	Scenarios that describe different ways that the JavaHelp system can be invoked from applications
Presentation	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.
Search Scenarios	Scenarios that describe different ways that full-text searches of JavaHelp system information can be implemented
Packing Scenarios	Scenarios that describe different ways that JavaHelp system data can be encapsulated and compressed using Java Archive (JAR) files
Merge Scenarios	Scenarios that describe ways that JavaHelp system data can be merged. 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.

Code examples complementing these scenarios can be found in the JavaHelp System 2.0 Reference Implementation available at http://java.sun.com/products/javahelp.

A.2 Invocation Mechanisms

These scenarios describe the different ways the JavaHelp system can be invoked. It is divided into two sections: Application Invocation and Internally Initiated Help

A.2.1 Application Invocation

These scenarios describe way of invoking the JavaHelp system within an application.

A.2.1.1 User Initiated Context Sensitive Help

The JavaHelp system is often invoked from an application when a user chooses an item from a Help menu, clicks on a Help button in an application GUI, or uses one of the context sensitive help gestures to request help on a GUI component.

The JavaHelp system provides a simple interface for requesting the creation of a help presentation by requesting that a topic ID (identified by a string) be displayed. Topic IDs are associated with URLs in the map file(s) mentioned in the HelpSet 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 HelpSet file (or in some cases the map file referred to in the HelpSet file) the ID `fc.help` is defined to be a file named `FileChooser.html` using the following syntax:

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

Separating the specification of actual file names from the program code, provides content authors the freedom to control the information that is associated with the topic ID.

A.2.1.2 Field-level Context-Sensitive Help

Field Level Context-sensitive help (sometimes included in the term *What-is help*) is help information that describes graphical components in an application GUI. It is triggered by gestures that activate context-sensitive help and then specify the component in question. See section on page 63 for more details.

A.2.1.3 Window-level Context-Sensitive Help

Window-level help is help information that describes the correct graphic component with focus, or an entire dialog in an application GUI. It is triggered by an operating system specific keystoke, generally either “F1” or “Help” keys, that activate context-sensitive help and specifies the component to get help on based on focus. See section on page 63 for more details.

A.2.1.4 System Initiated Context-Sensitive Help

Recent products are exploring the notion of a Helper, or an Assistant, an example is the assistant in MS's Office 97. A helper is a mechanism that reacts to state and state transitions in applications and provides guidance and suggestions to the user. Such a mechanism requires significant close interaction between the application and the information presented to the user.

A.2.2 Internally Initiated Help

These scenarios describe way of invoking help once inside the JavaHelp system.

A.2.2.1 Navigators

Each navigator provides a mechanism for changing the current topic displayed in the content viewer. For instance the selecting of a TOC item in a TOCNavigator would present the content tied to that TOCItem.

A.2.2.2 View

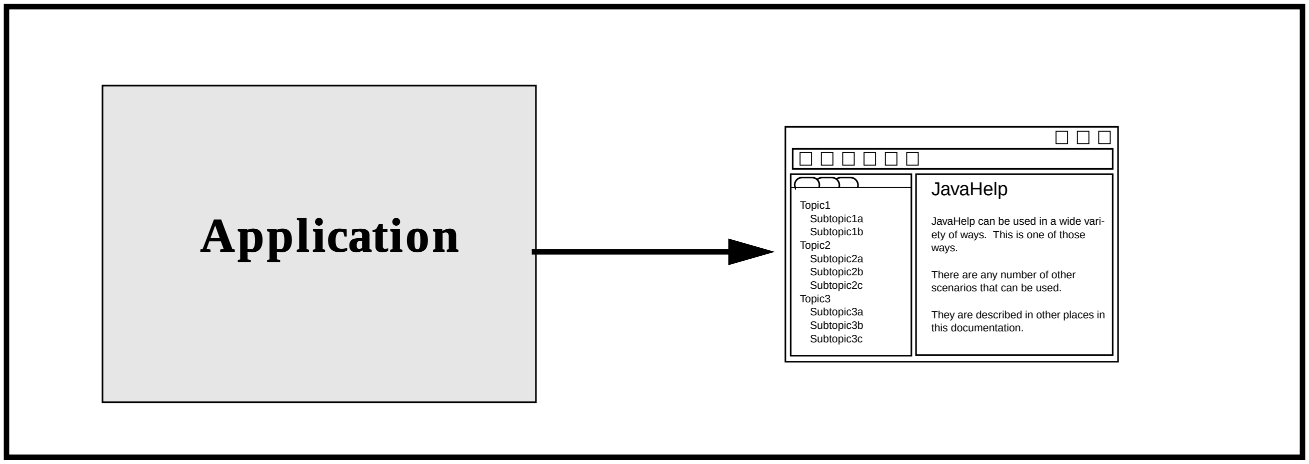
The help content viewer also provides a mechanism for displaying additional content. The new content can either replace the current content or display the new content in a an alternative presenation format.

A.3 Presentation

The following scenarios illustrate different ways the JavaHelp system can be used to present information. Each invocation mechanism is designed to allow presentations in each of the following scenarios.

A.3.1 Main Window

The main window is the main presentation for the JavaHelp system. By default it is a tri-paned fully decorated window consisting of a tool bar, navigator pane, and help content viewer. Most reference implementation would keep the main-window resident in memory when the window is not visible.



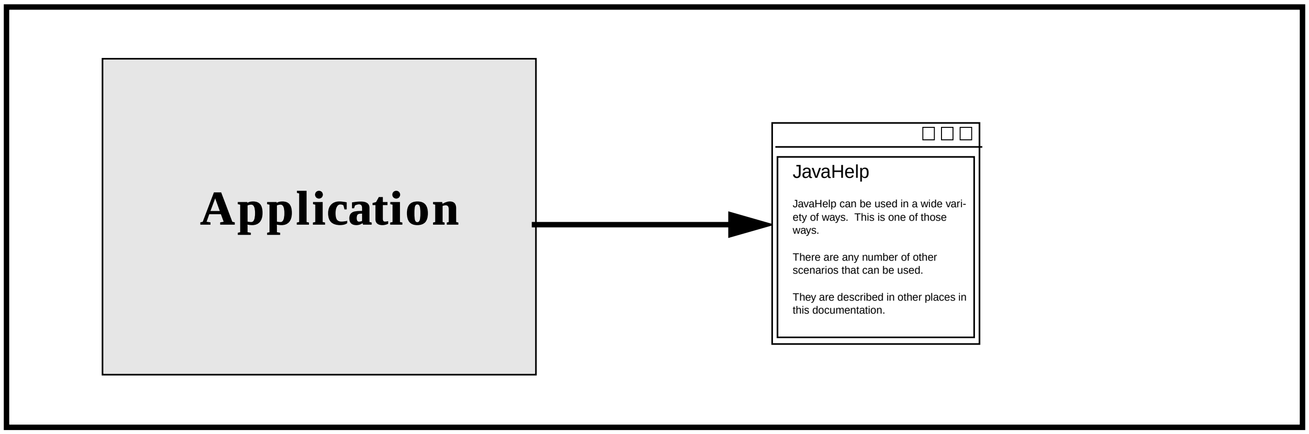
JavaHelp can be used in a wide variety of ways. This is one of those ways.

There are any number of other scenarios that can be used.

They are described in other places in this documentation.

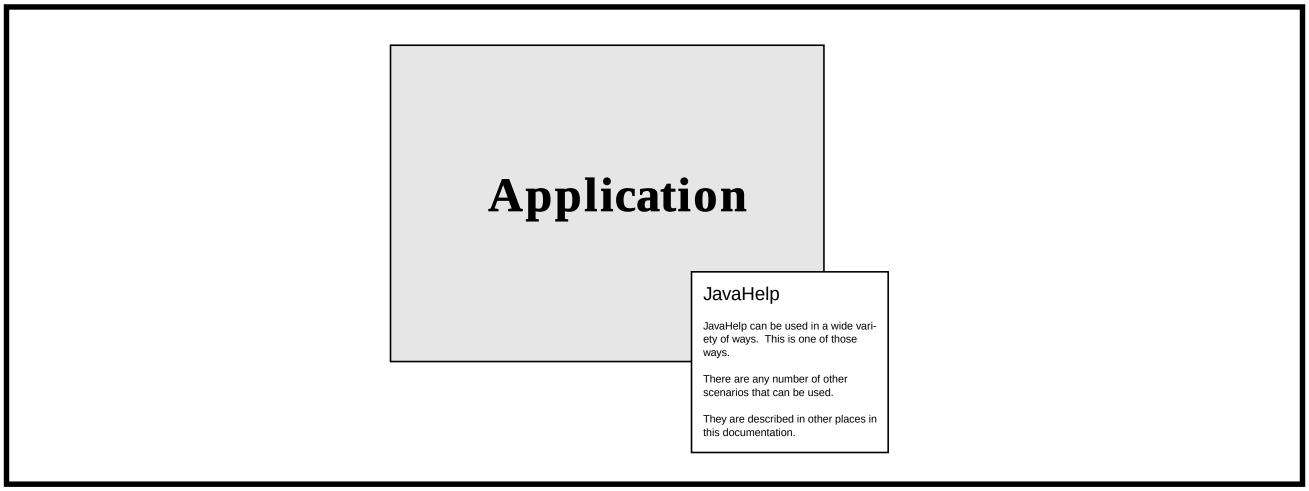
A.3.2 Secondary Window

A secondary window is similar to the main window in that it is a fully decorated window. By default it only contains a help content viewer though could optionally include a toolbar and / or navigators. Unlike the main window it is destroyed by default on closing. Additionally, secondary windows are named. If a named window is visible the current contents to be replaced.



A.3.3 Popups

Popups contain only a content viewer. They are intended to provide immediate help and then allow the user to continue working. Once a popup looses focus, it is destroyed.



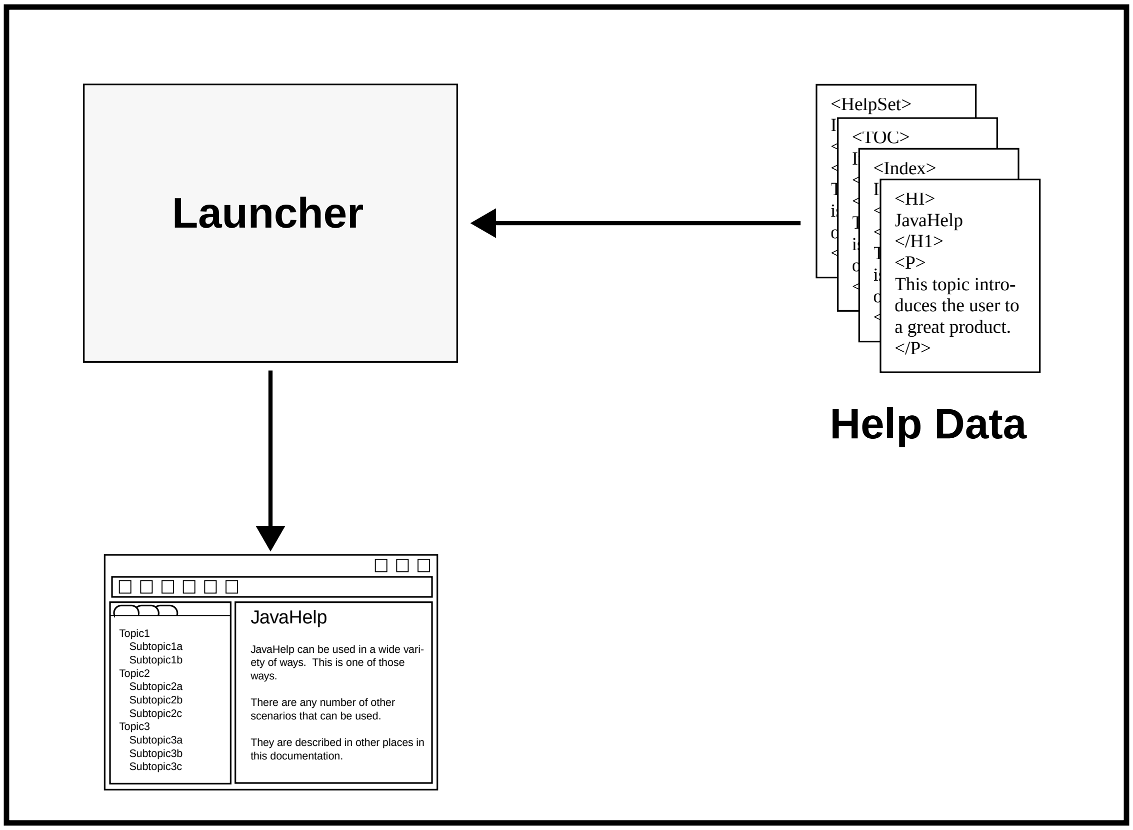
A.4 Deployment

The following scenarios illustrate different ways that the JavaHelp system can be used to present and deploy Help information.

A.4.1 Information Kiosk

The "kiosk" scenario is one where documents are presented independent of an application.

An example on the Solaris platform is AnswerBook -- a technology used to display all of Sun's documentation online. All that is required is a help browser that can be launched to present and navigate through the information.



JavaHelp can be used in a wide variety of ways. This is one of those ways.

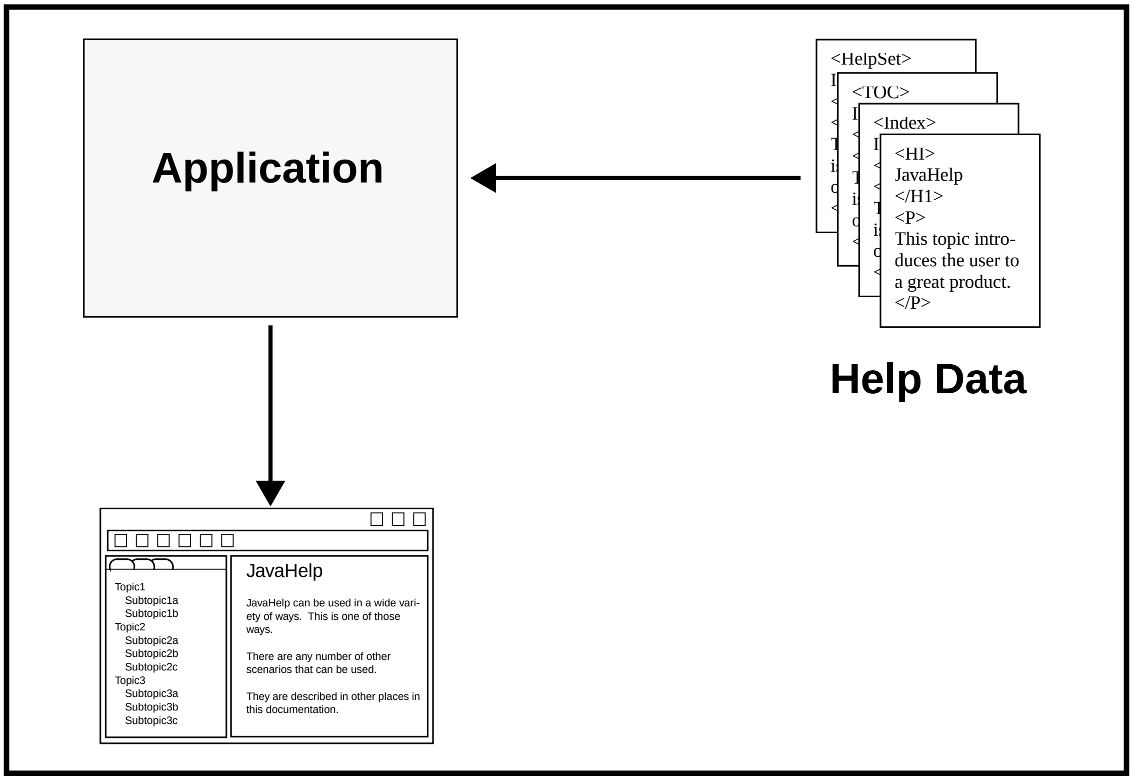
There are any number of other scenarios that can be used.

They are described in other places in this documentation.

In JDK1.2, a JAR file can indicate a containing Application class that will be invoked automatically by the system (by passing it to a "java -jar" command).

A.4.2 Stand-Alone Application

The simplest scenario is one in which the Java application runs locally and accesses help data installed on the same machine.



JavaHelp can be used in a wide variety of ways. This is one of those ways.

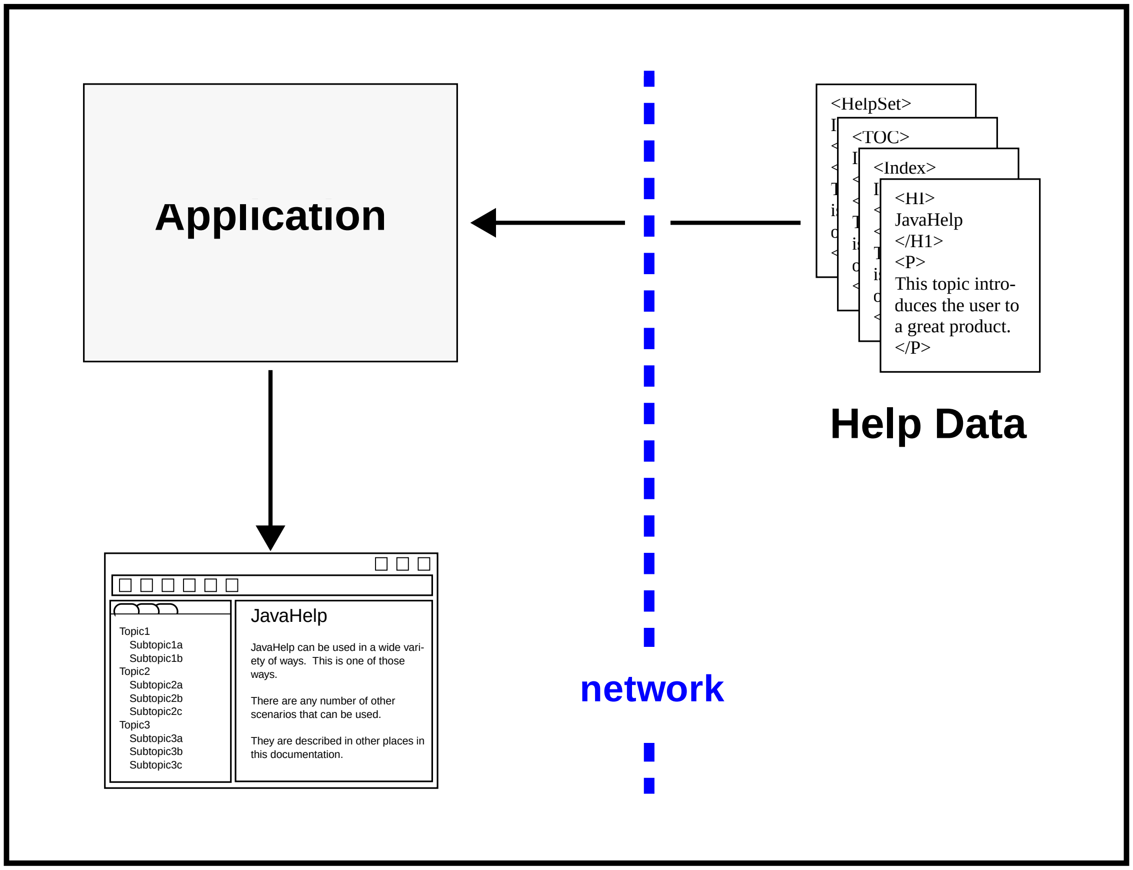
There are any number of other scenarios that can be used.

They are described in other places in this documentation.

The application requests the creation of a JavaHelp instance, loads the help data on it, and then interacts with this instance, either by requesting the help information be presented and hidden, or by requesting a specific topic (identified by an ID) be presented.

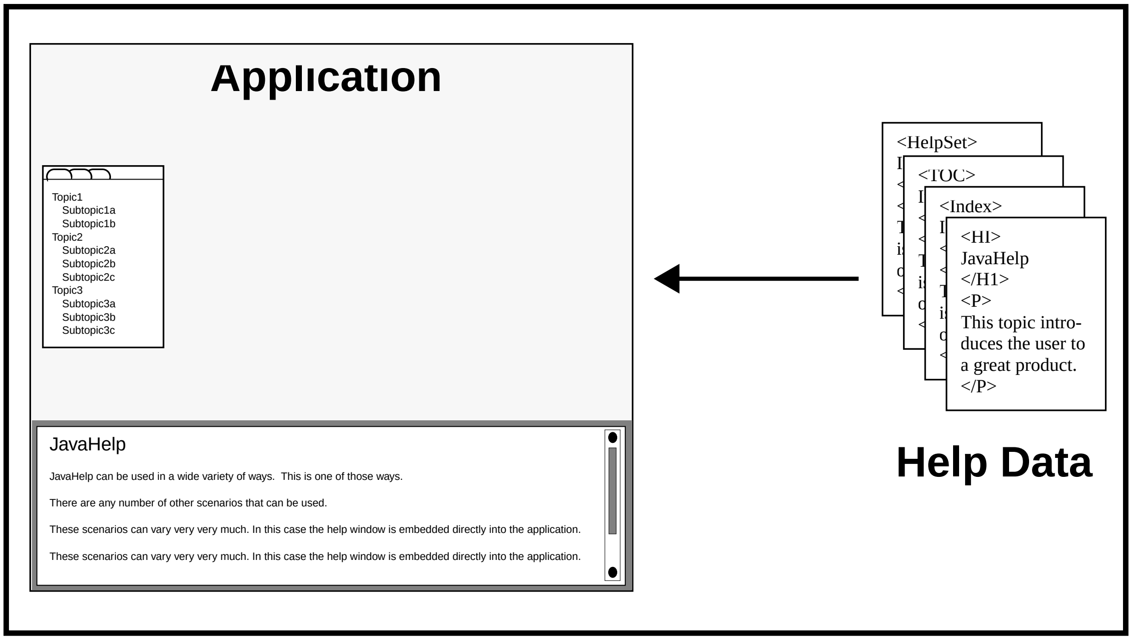
A.4.3 Network Application

When the help data is accessed across the network, the scenario is essentially the same -- the location of the data is actually transparent.



A.4.4 Embedded Help

Information can also be presented embedded directly in application windows. The JFC components that implement the JavaHelp specification are embedded directly into the application frame. The application can create its own customized presentation, by using the JFC components from the reference implementation.



JavaHelp can be used in a wide variety of ways. This is one of those ways.

There are any number of other scenarios that can be used.

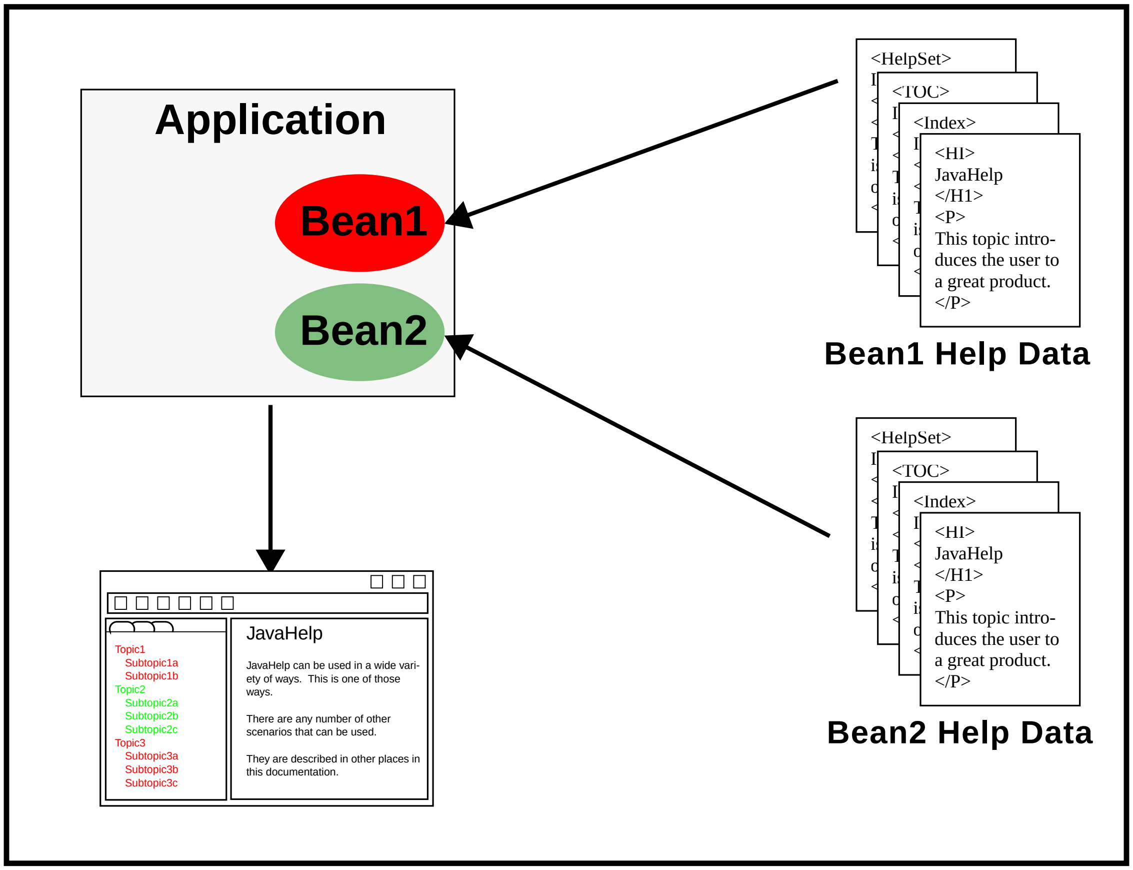
These scenarios can vary very very much. In this case the help window is embedded directly into the application.

These scenarios can vary very very much. In this case the help window is embedded directly into the application.

Embedded help is inherently application-specific since the application controls where each of the presentation UI components are located. The JavaHelp reference implementation is organized so that most applications will be able to achieve their needs very easily.

A.4.5 Component Help

Many current 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.



This topic introduces the user to a great product.

This topic introduces the user to a great product.

JavaHelp can be used in a wide variety of ways. This is one of those ways.

There are any number of other scenarios that can be used.

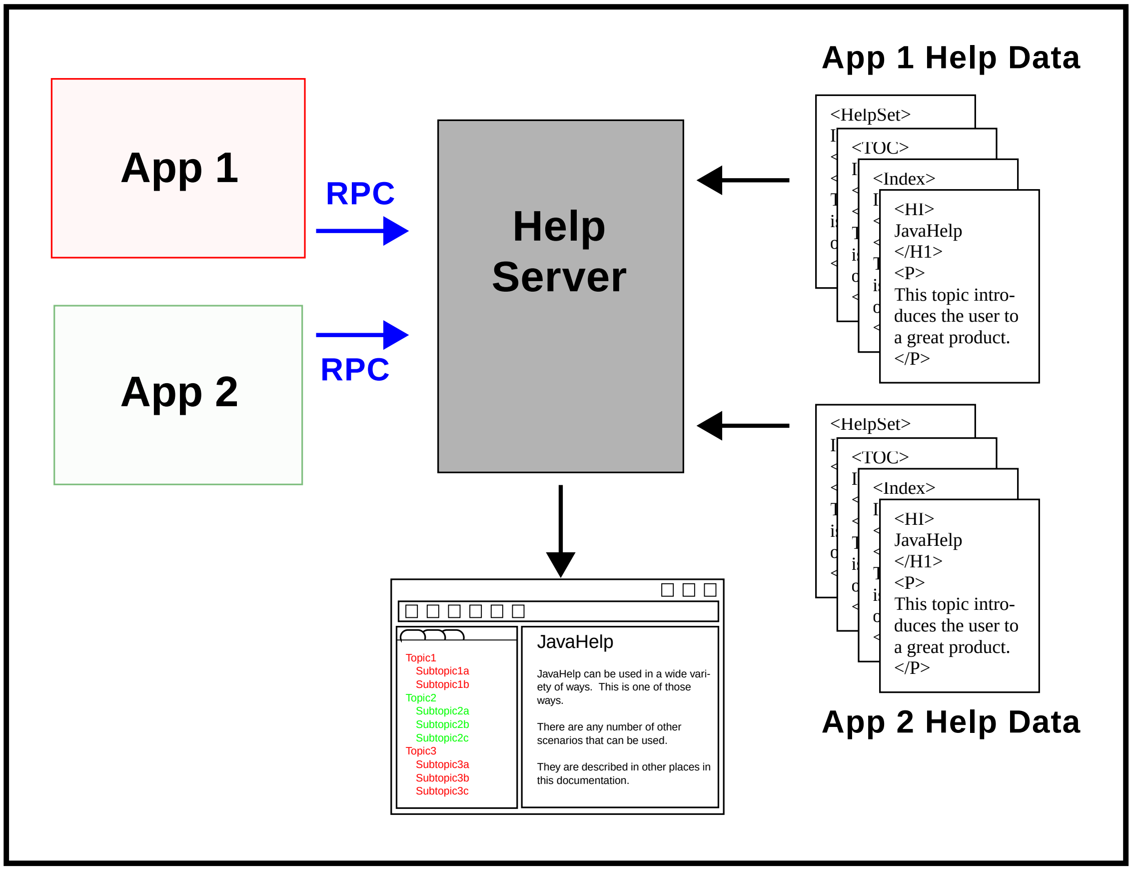
They are described in other places in this documentation.

The help information can be merged in different ways. For instance, the table-of-contents, index, and full-text search information from each component may be displayed one after the other to create a new, unified help system.

As HelpSets are loaded / unloaded into a JavaHelp instance, the information presented needs to be updated. The JavaHelp system provides a flexible mechanism for merging this information.

A.4.6 A Help Server

In some cases, it may be necessary to separate the application from the process that presents the help information. In this case the application process can make requests into a JavaHelp process (help server) through an RPC mechanism (the RPC may be wrapped in a library and be invisible to the application developer).



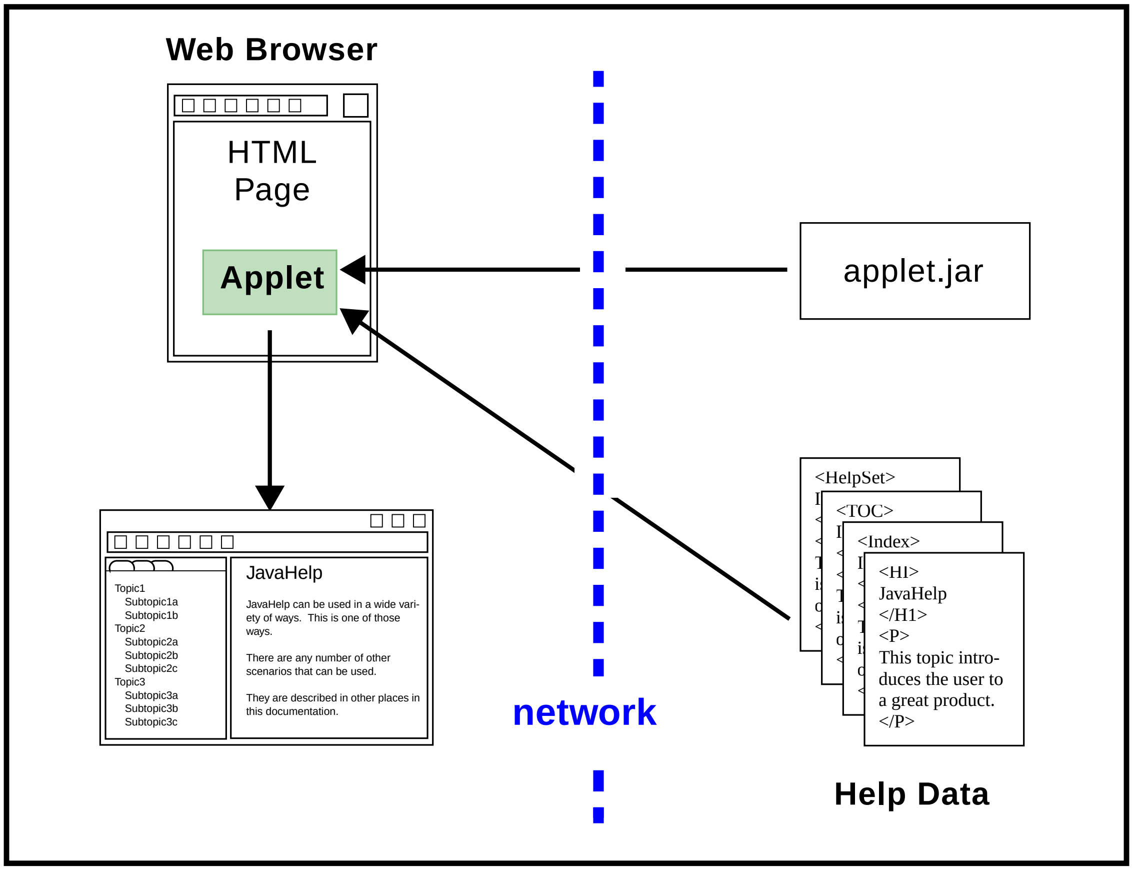
The help server model is useful if the application is not written in Java and does not include a JVM. It would also be useful for a suite of Java applications that can share a common help server.

A.4.7 Web Pages and Applets

This scenario describes how the JavaHelp system is used from within web-based applications. In this case an applet or some other triggering entity (perhaps a JavaScript event) on an HTML page creates a `HelpSet` object and then invokes `HelpSet.createJavaHelp()`.

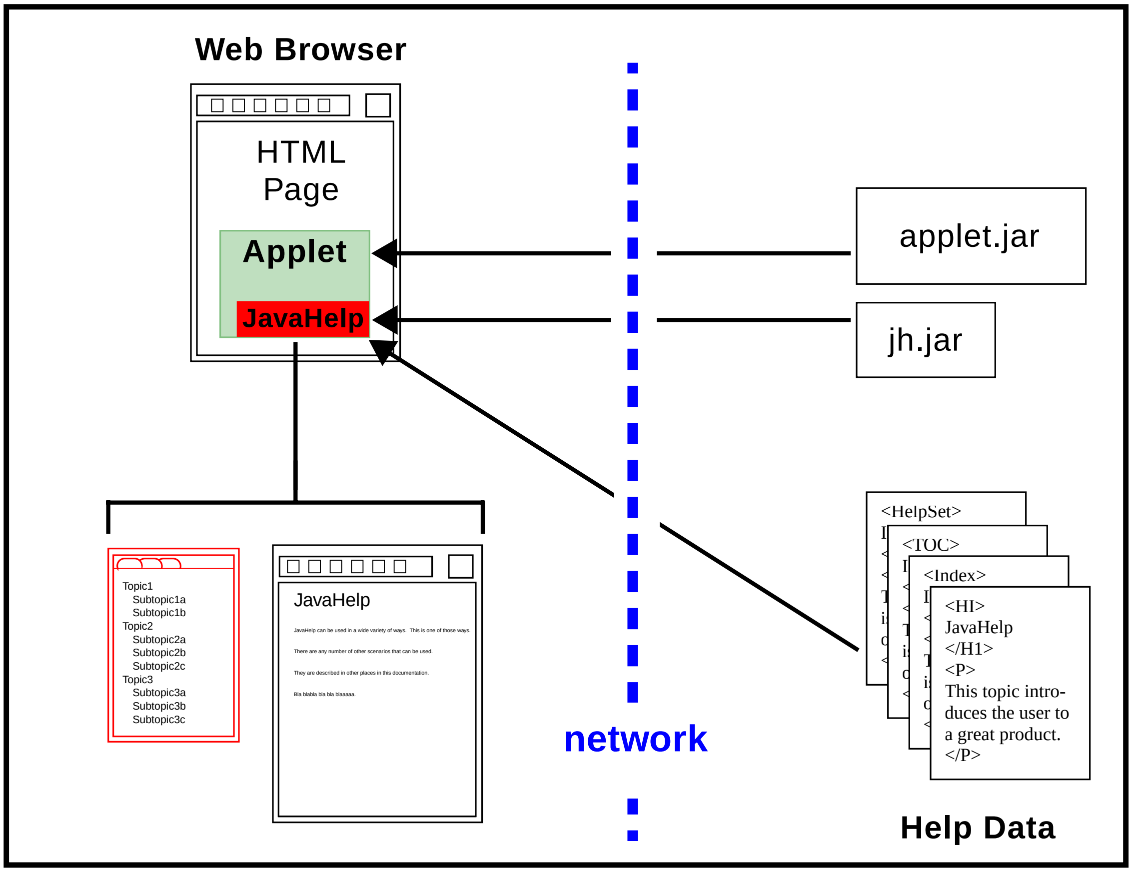
This scenario can have a number of variations. Here are a five:

- In one case, the browser platform contains a customized implementation of the JavaHelp system. This implementation may have been delivered with the browser, or it may have been downloaded by the client into the CLASSPATH. The implementation may use the Swing HTML viewer, or, more likely, it may use some the HTML viewer that comes with the Web Browser.



- Since the JavaHelp system is a Java "standard extension," it is possible that a fully-conforming JDK browser may not have it in its CLASSPATH. In this case, if the HTML page refers to the standard JavaHelp system implementation, the standard extension machinery will automatically download the implementation and execute it. Since our implementation is quite small, this approach will often be practical. Browsers may choose to provide some way of easily installing extensions downloaded through this mechanism.

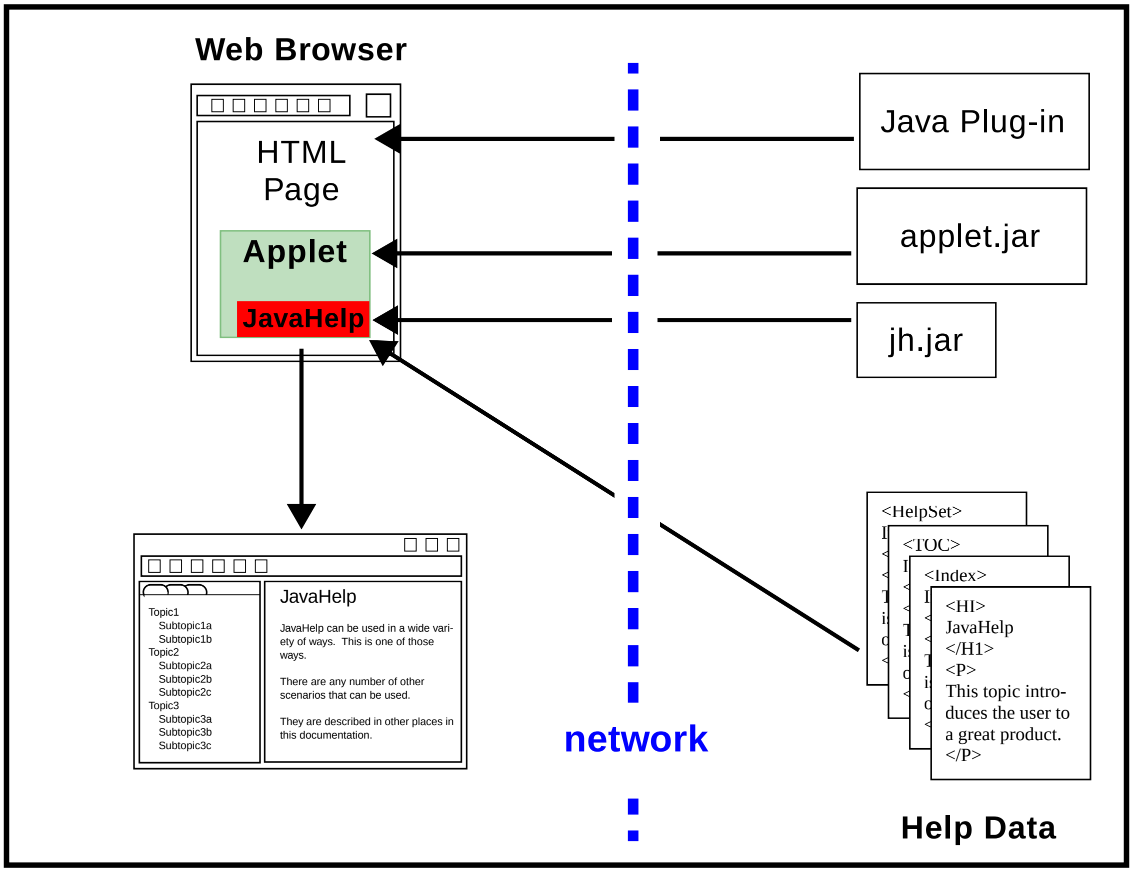
This situation is depicted in the next picture where, for variety sake, we have changed the help presentation so the navigator is separate from the content.



The corresponding APPLET tag may look something like this:

```
<APPLET
    CODE=javax.help.HelpButton
    ARCHIVE="JavaHelpDefault1_0.jar"
>
<PARAM
    NAME=HelpSet
    VALUE=MyHelp.JAR>
</APPLET>
```

- In some cases, some client browsers may not have a fully-conforming Java Virtual Machine. In that case we can use the Java Plug-in technology to request a compliant Java Virtual Machine. The request may lead to a download request if the virtual machine is not available locally; once installed later requests will proceed with no download step. Once the appropriate JVM has been started, the situation is equivalent to the previous two steps. The following figure illustrates this:



This topic introduces the user to a great product.

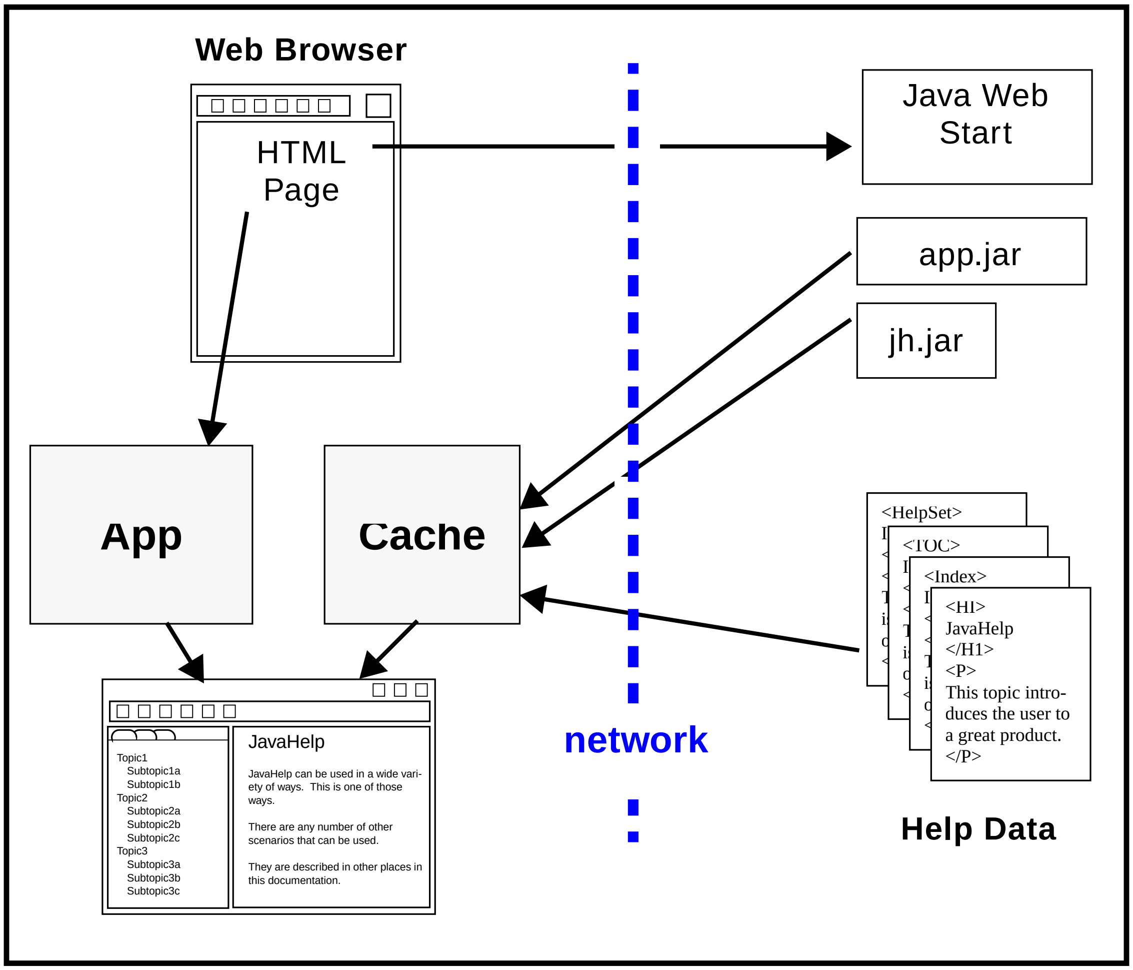
JavaHelp can be used in a wide variety of ways. This is one of those ways.

There are any number of other scenarios that can be used.

They are described in other places in this documentation.

The JavaHelp system provides mechanisms for extending navigational views and content display, the classes providing this can be downloaded automatically using the standard classloader mechanisms of the Java platform (e.g. using ARCHIVE or CLASSPATH).

- In the next scenario the client browser is used to launch a full-featured application with a single click. In this case we can use the Java Web Start technology. Java Web Start will download the application if it isn't already present on the user's computer. The application is then activated. The following figure illustrates this:



JavaHelp can be used in a wide variety of ways. This is one of those ways.

There are any number of other scenarios that can be used.

They are described in other places in this documentation.

- In the final scenario the client browser will handle all of the display using HTML or some combination of HTML, DHTML and / or JavaScript. In this scenario the server is a Java Server supporting Java Server Pages (JSP). The client browser submits a JSP request to the server. The server transforms the JSPs into Java Servlets and accesses the HelpData on the server. Results are returned to the client browser in the form of HTML, DHTML and / or JavaScript. An illustration of this scenario can be found in section 7.2 on page 40.

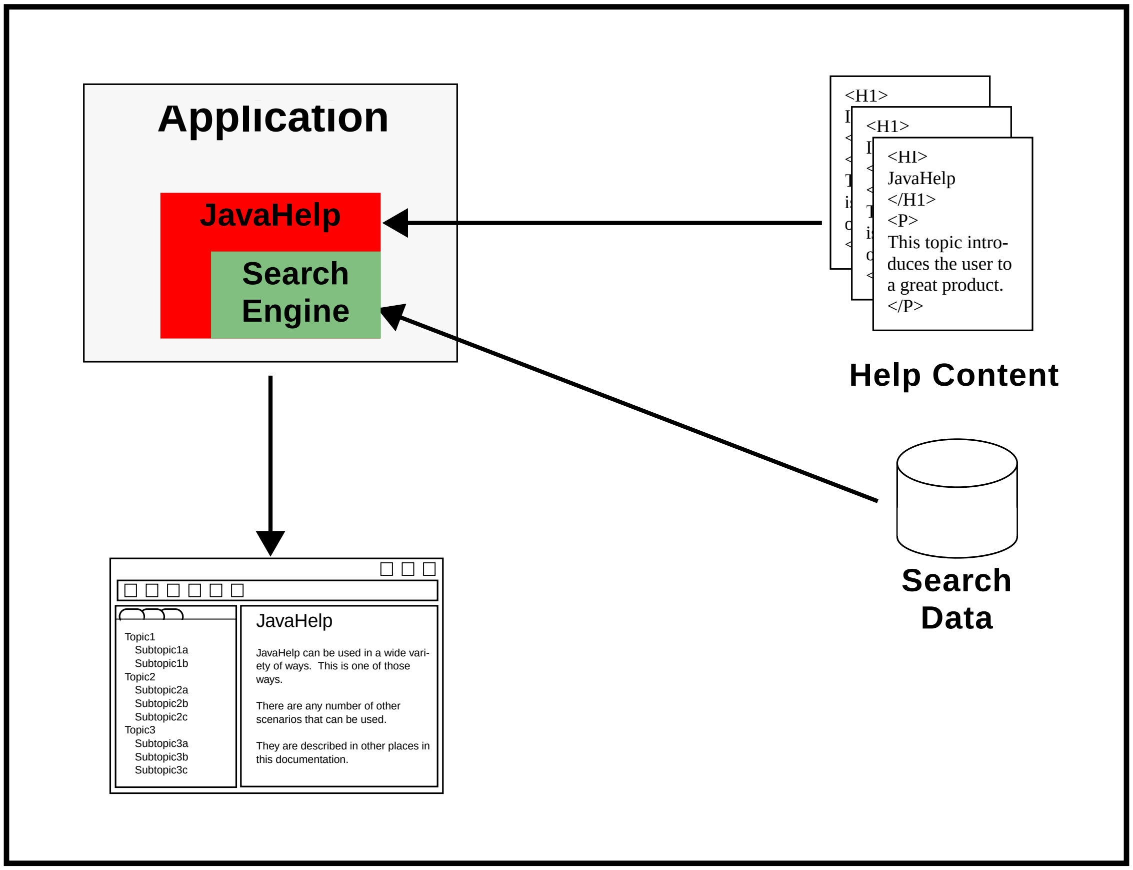
A.5 Search Scenarios

The JavaHelp system supports an extensible full-text search mechanism using the extension framework (see section 2.5.6 on page 13) mechanism, plus a Search interface. The JavaHelp1.0 specification requires all implementations to support some search types and formats. This mechanism can be used to support a number of different search scenarios:

Client-Side	The search database is downloaded from the server, then searched on the client
Server-Side	The search database and search engine are located on the server
Stand-Alone	The search database is included as part of the HelpSet and the search occurs in the application

A.5.1 Client-Side

In a client-side search, searching is done locally on the "client-side", but the search data originates on the "server-side". This commonly occurs with web-based applications (applets). 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 content files are downloaded only when they are presented.

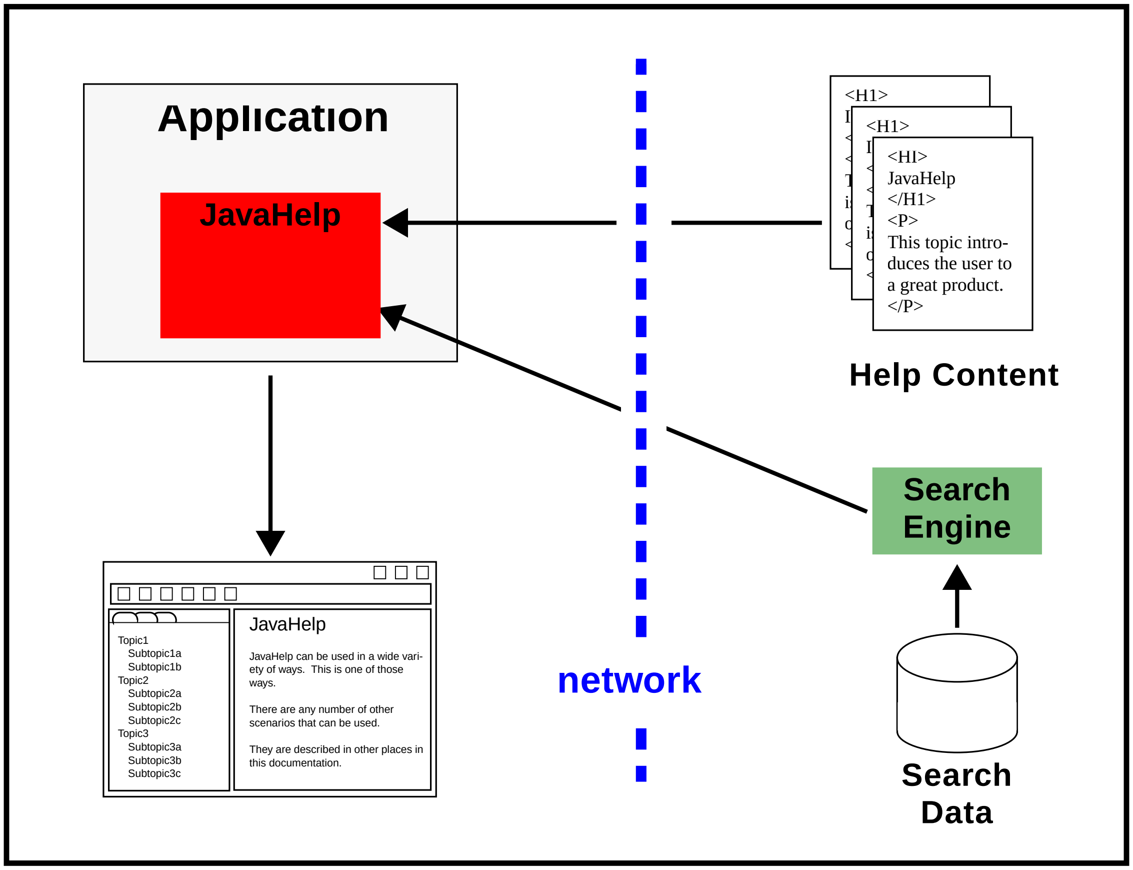


1. Search is initiated
2. Search engine loads database
3. User (or application) chooses a "hit"
4. Content is loaded and displayed

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

A.5.2 Server-Side

In a server-side search, both the search data and the content files are located on the server side; only the results of the search are downloaded to the client.

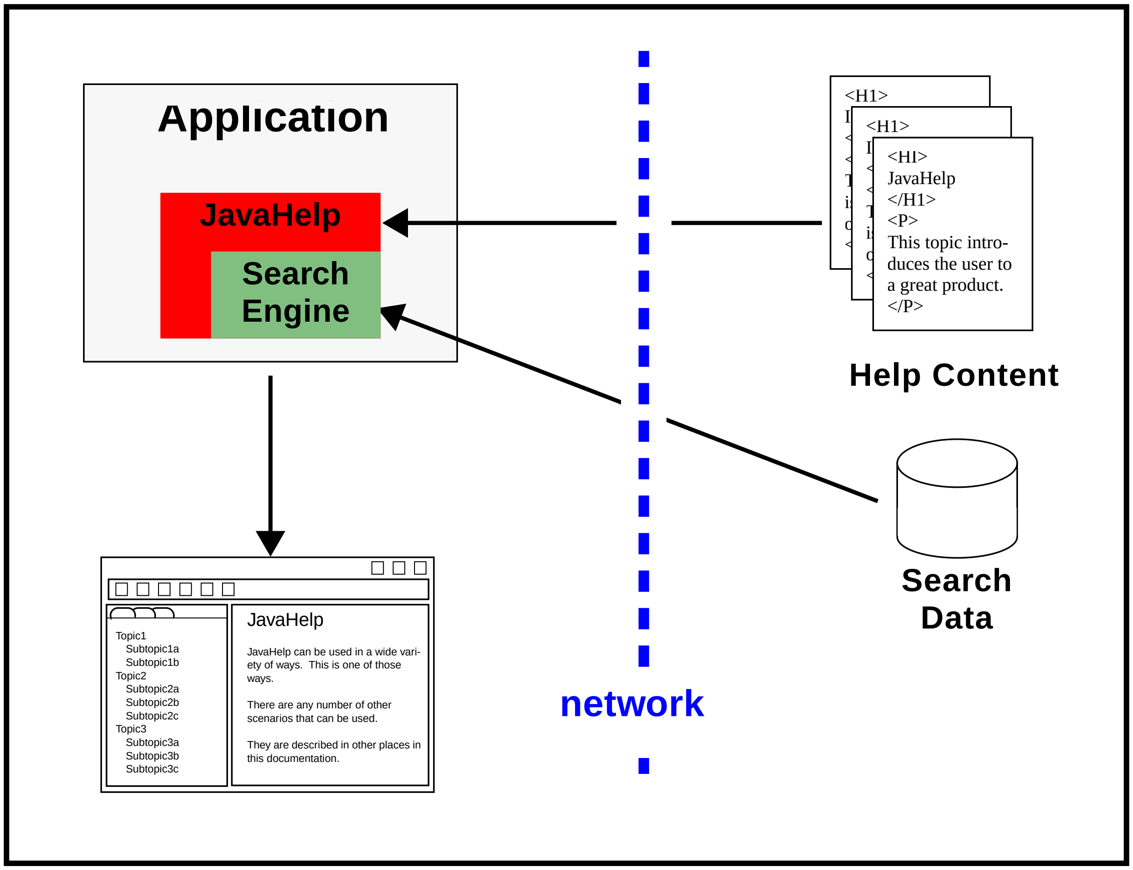


1. Search is initiated
2. JavaHelp requests search from server
3. Server-side search engine searches database and delivers “hits” to application
4. User (or application) chooses a “hit”
5. Content is loaded and displayed

This is another option for applets. It permits developers to use a choice of commonly available search engines and can provide quick start-up time (especially if the search engine is started ahead of time). On the other hand, it requires additional administrative work to get the search engine installed. Note that this approach works very well with Java servlets.

A.5.3 Stand-Alone

In a stand-alone search, all of the components are local (search engine, search database, help content). From an implementation point-of-view, the stand-alone search is quite similar to the client-search except that there is no need to cache the search data in memory or in local files.



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

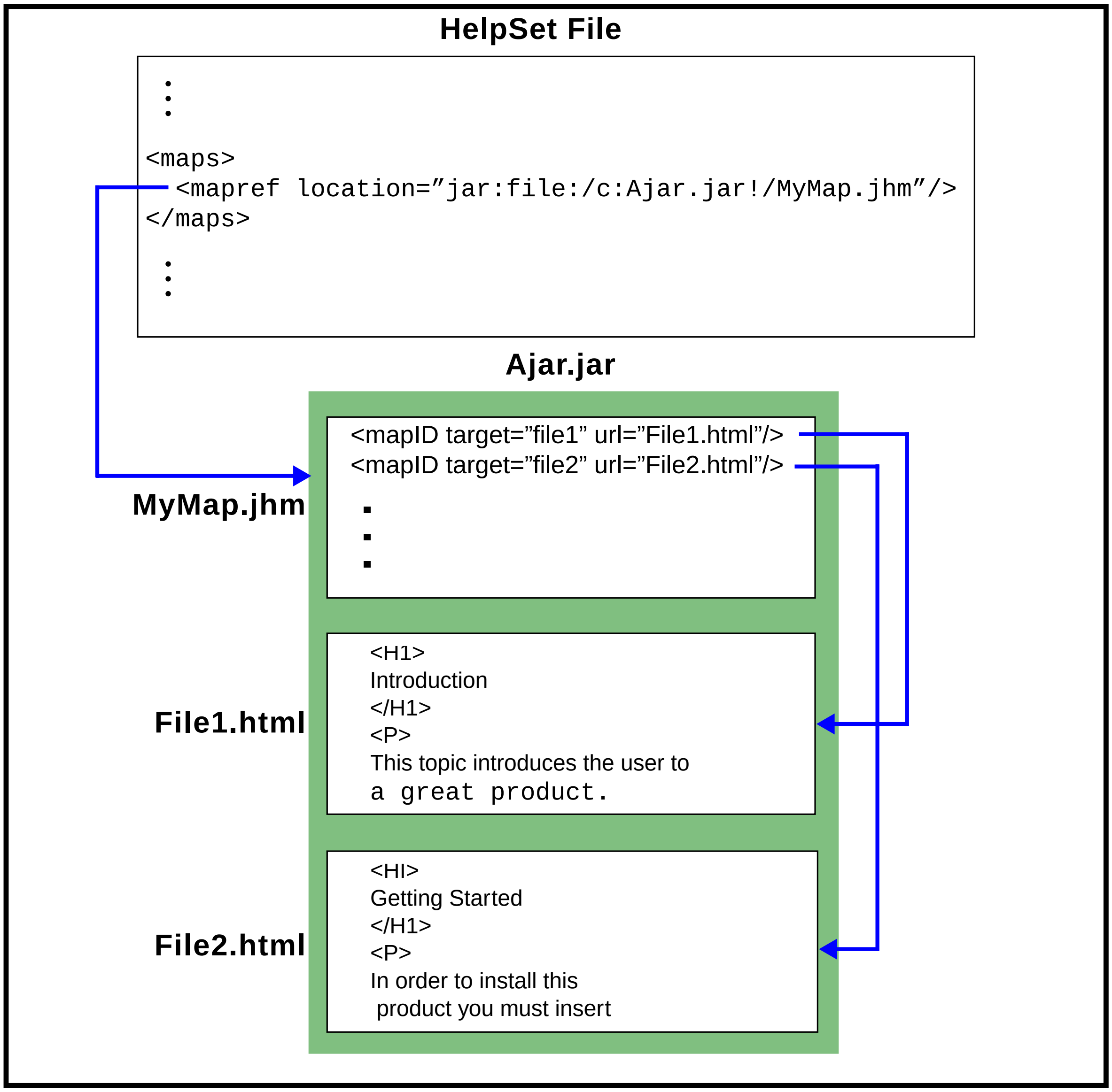
Note that help content files can be accessed locally and / or across a network.

A.6 Packaging Scenarios

The following diagrams represent typical packaging scenarios. These scenarios are intended to be exemplary and are not exhaustive.

The first picture represents a project in which the map file is packaged together with most (all?) of the content files. The "!" syntax is used to specify the URLs relative to the

JAR where the map is located. The HelpSet file is packaged outside of the JAR file, perhaps to simplify updates later on.



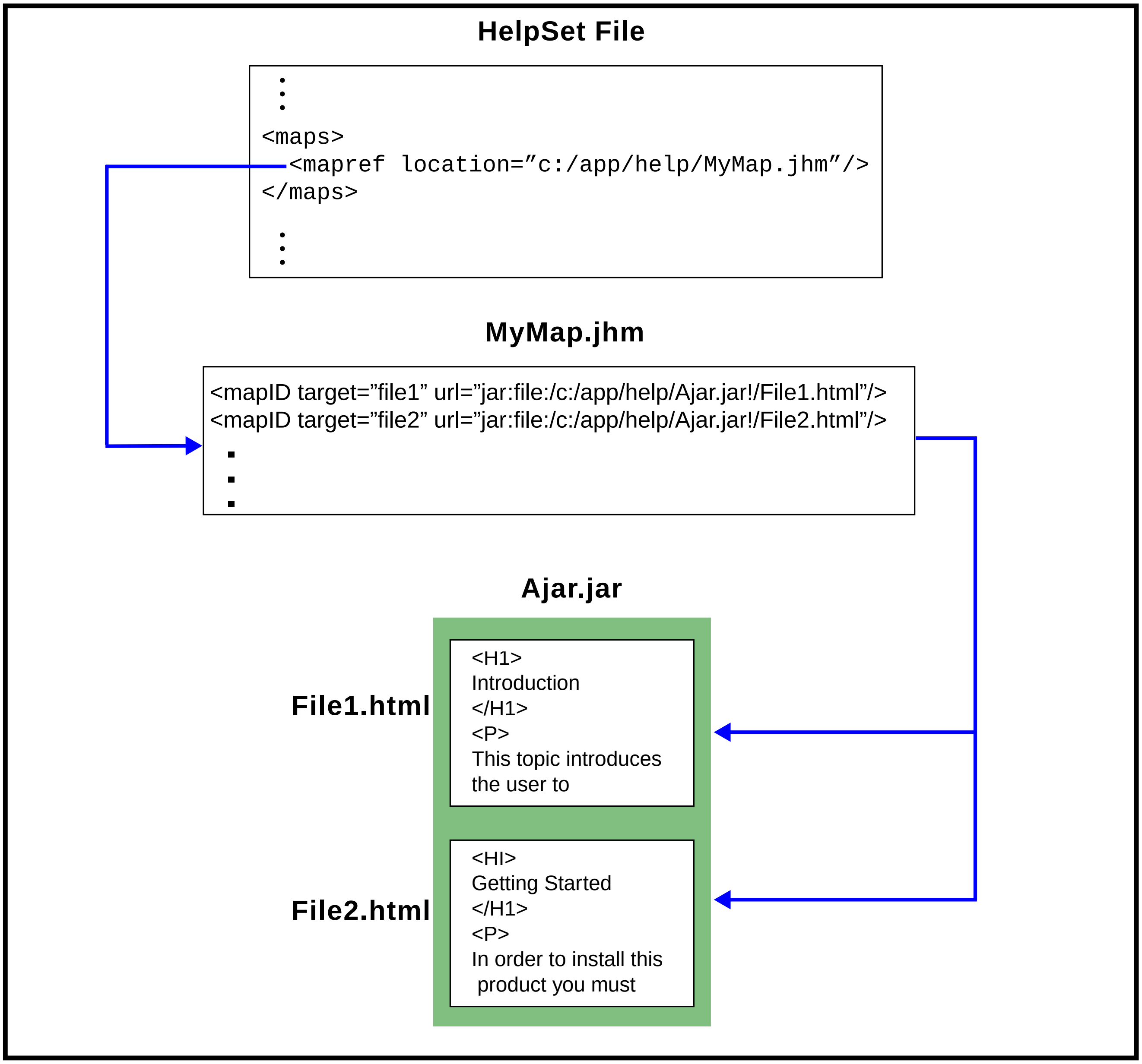
```
<maps>
  <mapref location="jar:file:/c:Ajar.jar!/MyMap.jhm"/>
</maps>
```

```
<mapID target="file1" url="File1.html"/>
<mapID target="file2" url="File2.html"/>
```

```
<H1>
Introduction
</H1>
<P>
This topic introduces the user to
a great product.
```

```
<HI>
Getting Started
</H1>
<P>
In order to install this
 product you must insert
```

In the following scenario, the map file and the JAR file are in different locations. This is probably not a common scenario, but is shown to illustrate packaging flexibility.



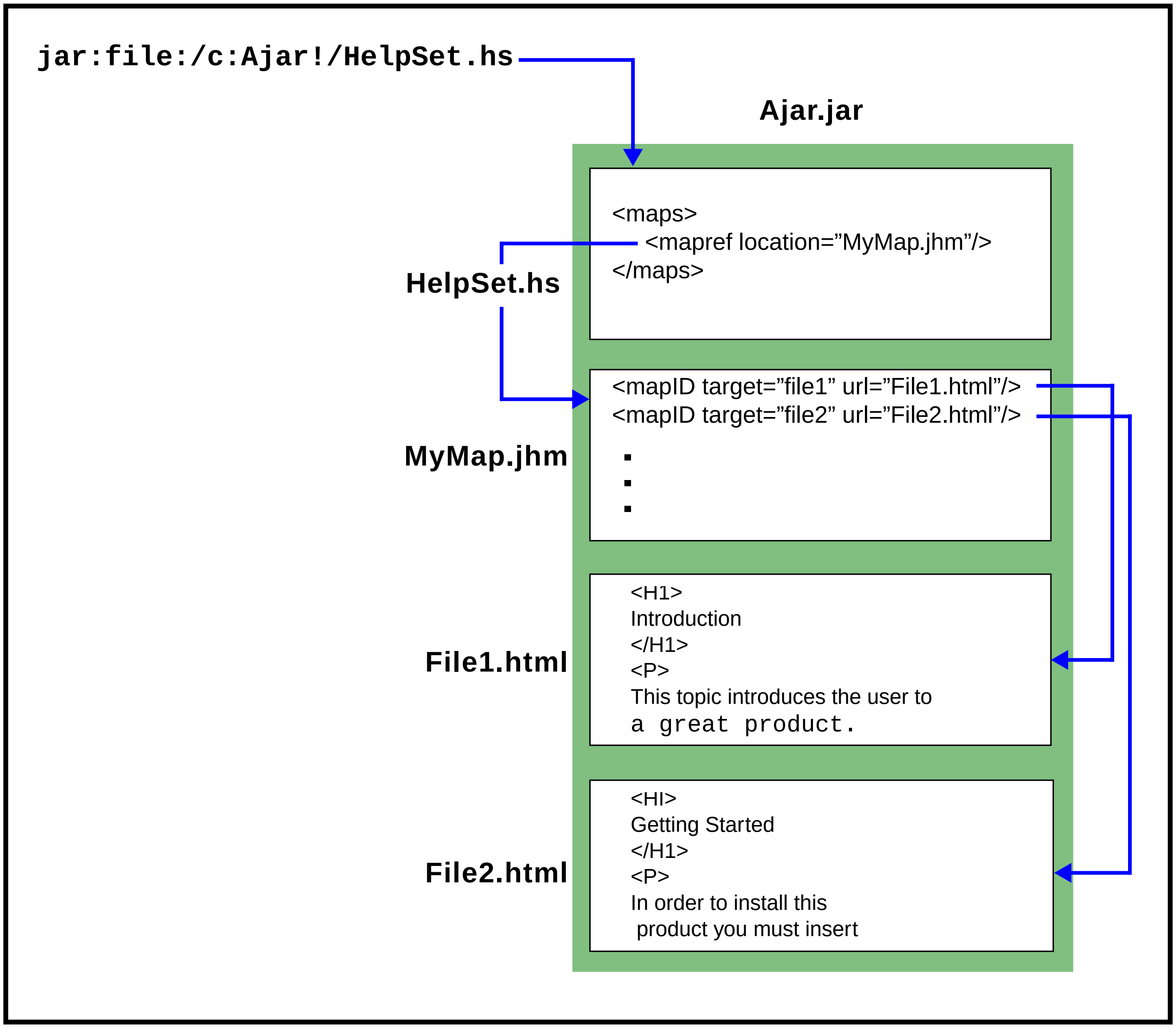
```
<maps>
  <mapref location="c:/app/help/MyMap.jhm"/>
</maps>
```

```
<mapID target="file1" url="jar:file:/c:/app/help/Ajar.jar!/File1.html"/>
<mapID target="file2" url="jar:file:/c:/app/help/Ajar.jar!/File2.html"/>
```

```
<H1>
Introduction
</H1>
<P>
This topic introduces
the user to
```

```
<HI>
Getting Started
</H1>
<P>
In order to install this
 product you must
```

In the final scenario, the HelpSet file is bundled in the JAR file with the rest of the JavaHelp system data.



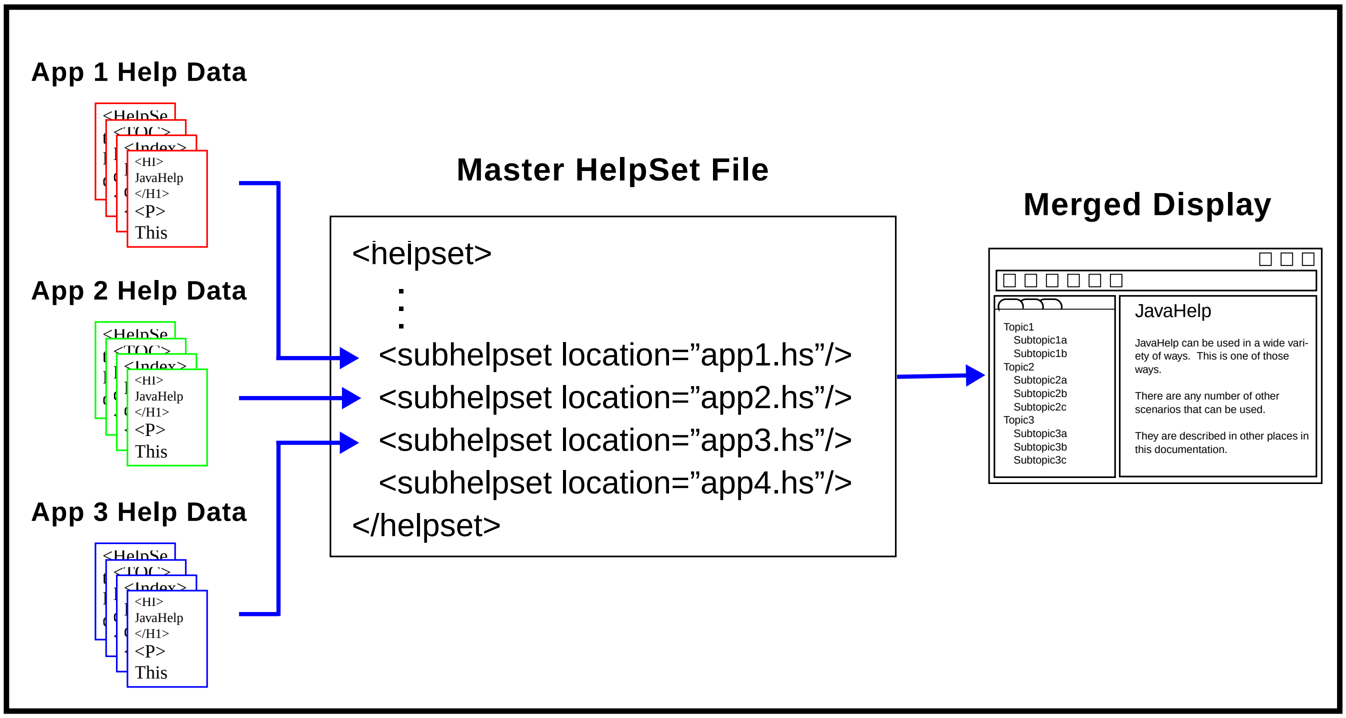
The advantage of this arrangement is that all the URLs are relative to the base URL of the HelpSet file, and that there is no need to mention the jar: protocol within any JavaHelp system file. This JAR, when placed in a CLASSPATH, permits a JDK1.1 application to refer to the HelpSet within the JAR file transparently. A similar situation occurs with Applets, when the JAR file is listed in the ARCHIVE attribute.

A.7 Merge Scenarios

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 be comprised 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.

In the following scenario an application suite is comprised of three possible suite components. The help data for each component in the suite is delivered as its own HelpSet. The suite is shipped with a master HelpSet that lists the subcomponent HelpSets. When the HelpSet object for the suite HelpSet file is created, each subcomponent HelpSet file (specified by means of the <subhelpset> tag) is read to create HelpSet objects that are then merged into the containing HelpSet. Subcomponent HelpSet that are not installed are ignored.



```
<helpset>
    :
  <subhelpset location=”app1.hs”/>
  <subhelpset location=”app2.hs”/>
  <subhelpset location=”app3.hs”/>
  <subhelpset location=”app4.hs”/>
</helpset>
```

JavaHelp can be used in a wide variety of ways. This is one of those ways.

There are any number of other scenarios that can be used.

They are described in other places in this documentation.

For more information about merging see section 12 on page 70 or "Merging HelpSets" in the JavaHelp System User's Guide.

Appendix B
JavaHelp System 2.0 Reference Implementation

Sun's reference implementation of the JavaHelp system implements the JavaHelp system specification and supports additional useful features that are not appropriate for inclusion in the specification at this time. Some of these features may move to the specification unchanged, others may be replaced by equivalent or more powerful features in future versions of the specification, and others may never show up in the specification. In all cases, these features will be supported in future versions of the reference implementation and their presence can be assumed when writing content targeted to this implementation.

The latest release available at the time of writing is the FCS release, released in April 1999. The FCS release implements this version of the specification. This specification is also supported by `javadoc` API documents.

Sun's reference implementation provides a search engine that can be used to create and access a search database created from HTML-base topic files. The reference implementation also supports lightweight section Appendix D on page 108that can be embedded in HTML pages using the <OBJECT> tag. Two example components are provided: one component provides HTML *popup* functionality, the other provides in-line glossary definitions.

Information about the JavaHelp system reference implementation as well as other JavaHelp system information is available at http://java.sun.com/products/javahelp.

B.1 HelpBroker

The `HelpBroker` created by default upon invocation of the `createHelpBroker()` method of HelpSet is a DefaultHelpBroker.

B.2 Search Engine

The reference implementation includes a `com.javax.help.search.DefaultSearchEngine` search engine. This search engine uses a single *data* attribute that is a relative URL that specifies the directory that contains the search database. Multi-word queries are supported and are interpreted using a relaxation algorithm described in section C.2 on page 106.

The implementation of the search engine is independent and does not depend on the rest of the JavaHelp system. The client classes do not depend on Swing, the classes that create the search database (the indexer) depend only on the Swing parser for the HTML IndexerKit.

B.3 Java Components in <OBJECT> Tag

The reference implementation supports a powerful `<OBJECT>` tag. In the reference implementation the `CLASSID` that denotes the class name is used to instantiate the class. The result is expected to be a lightweight AWT Component. This class is interpreted as a JavaBeans component --the `<PARAM>` tag associated with the `<OBJECT>` tag is used to provide `NAME`/`VALUE` pairs. Each `NAME` is interpreted as the name of a String property of the JavaBeans component and the value is assigned to it.

If the created Component supports the `ViewAwareComponent`, then the `javax.swing.text.View` is passed to the object through a call to `setViewData`. This mechanism is very powerful and provides access to much useful information, for example, the URL to the document where the `<OBJECT>` tag is present. See the documentation about the Swing text package for more details.

B.4 Launcher Application

A simple application (`hsviewer`) that can be used to create a HelpBroker on a given HelpSet is included in the FCS release. The `hsviewer` is described in the reference implementation release documentation.

B.5 Packaging

The reference implementation includes the following JAR files in the FCS release:

JAR file	Description
`jh.jar`	Client-side JAR. Includes all default types, and the client-side search engine.
`jhall.jar`	Complete JAR. Like `jh.jar` but also includes the indexer classes.
`jhbasic.jar`	Minimal client-side JAR. Includes all default types except SearchView.
`jhtools.jar`	Tools JAR. Includes the indexer and search classes, as well as a simple launcher class.
`jsearch.jar`	Search JAR. Includes only the Search classes, both indexer and the search classes.

Appendix C
JavaHelp 2.0 - Relaxation Searching

C.1 Introduction

The default search engine in `com.sun.java.help.search.DefaultSearchEngine` uses an effective natural language search technology that not only retrieves documents, but locates specific passages within those documents where the answers to a request 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.

C.2 Relaxation Ranking

The query engine makes use of a technique called "relaxation ranking" to identify and score specific passages of material where the answer to a request is likely to be found. This is referred to as "specific passage retrieval" and is contrasted with the traditional "document retrieval" which retrieves documents but leaves the user with the task of finding the relevant information within the document (or finding that the desired information is not in the document after all).

The relaxation ranking algorithm looks at the search terms and compares them to occurrences of the same or related terms in the documents. The algorithm attempts to find passages in the documents in which as many as possible of the query terms occur in as nearly as possible to the same form and the same order, but will automatically relax these constraints to identify passages in which not all of the terms occur or they occur in different forms or they occur in different order or they occur with intervening words, and it assigns appropriate penalties to the passages for the various ways in which a passage departs from an exact match of the requested information. Passages with words in the same order as the search terms are scored better than passages with the matching words in some other order. Passages with matching words in any order are scored better than passages which do not contain matches for all of the requested terms.

C.3 Conceptual Indexing

Conceptual index consists of the following linguistic resources

- tokens
- lexicons
- lexicons - domain specific
- morphology
- classification

The more of the linguistic resources built into an indexer the better the conceptual index. The best indexer incorporate all of the above resources.

IMPORTANT: Although the core search engine in the reference implementation supports all these concepts, the indexer (search builder) available in JavaHelp 1.0 only incorporates tokens. Details of the other concepts are included below just for the interested reader.

The indexing engine can perform linguistic content processing of the indexed material to analyze the structure and interrelationships of words and phrases and to organize all of the words and phrases from the indexed material into a conceptual taxonomy that can be browsed and can be used to make connections between terms in a query and related terms in the material that you'd like to find.

C.4 Morphological and Semantic Relationships

The relaxation ranking algorithm is a very effective retrieval method all by itself, but can produce significantly improved results by using morphological and semantic relationships from the conceptual taxonomy to automatically make connections between query terms and related terms that may occur in desired passages.

Morphological relationships refer to relationships between different inflected and derived forms of a word, such as the relationship between "renew" and "renewed" (past tense inflection) and "renew" and "renewal" (derived normalization). Derived and inflected forms of a word are treated as more specific terms in the conceptual taxonomy, so that a request for "renew" will automatically match "renewed" and "renewal" (with a small penalty).

Semantic relationships refer to relationships between terms that are more general or more specific than other terms or that imply other terms. For example, "washing" is a kind of "cleaning" and since it is more specific than "cleaning" it will automatically be matched by a request for "cleaning" (again with a small penalty).

Passages with exact word matches are scored better than passages with morphological matches or matches using semantic relationships.

Appendix D
JavaHelp™ 2.0 - Java Components

The reference implementation has two JComponents that can be used in HTML pages

Secondary Window	Presents a secondary window for presentation of supplementary HTML-based information
PopUp	Presents a popup for presentation of supplementary HTML-based information

Appendix E
History of Changes

The following is a list of changes from the V1.0 Specification

- Added Glossary and Favorites Navigators
- Removed JDK 1.1 as a supported platform
- Added Server based JavaHelp through JSP Extensions and ServletHelpBroker
- Added comprehensive merge support
- Added Presentation controls to HelpSet file and navigator files
- Added Presentation Class and CSH changes to support presentation class
- Added customizable Toolbar support in HelpSet file
- Added implementation section to HelpSet file
- Added Dynamic CSH for components
- Update Invocation Mechanism scenarios
- Added category and topic default images to TOC file definition