JavaHelp™ 1.0 Specification

Sun Microsystems, Inc.

Copyright 1998 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, nontransferable, worldwide, limited license (without the right to sublicense) under SUN's intellectual property rights that are essential to practice the JavaHelp 1.0 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.

JavaHelp 1.0 - Introduction

Status of this Specification

JavaHelp™ is the Help system for the Java™ Platform. These documents describe the JavaHelp 1.0 specification. As of the publication of this document, the closest released implementation is the JavaHelp 1.0 release which follows this specification.

We followed Sun’s Open Development Process for the Java Platform, an open and inclusive process that produces high-quality specifications in "Internet-time". Through this process the critical feedback from all reviewers helped us transform early specifications into a high quality final specifications that satisfied the needs of the user community. The release of this specification is part of this process.

We expect the specification to continue to be extended in future updates. Please send us your feedback to guarantee that future specifications best suites your needs.

How to read this Specification

Two sets of documents are included. The first set is the actual specification that describes the JavaHelp API and its use. Also included are several related documents 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. In order to make the JavaHelp system features more concrete and easy to understand, a number of usage scenarios are explained in a companion document. 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 1.0 release 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.

Table of Contents of Specification

- Introduction to the JavaHelp API (this document)
- Overview of the JavaHelp API
- Formats of JavaHelp Data Files
- Localizing JavaHelp
- Customizing JavaHelp
- JavaBeans Help Data
- Context-Sensitive Help
- Search API
- Merging Help Information
- Specification Change History
- **The JavaHelp Classes**
  - JavaHelp Class Architecture
  - Package javax.javahelp

Related Documents

- JavaHelp Scenarios
- The JavaHelp 1.0 Reference Implementation
- Java Components
- `jar:` Protocol Specification

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

Further information on Java technology can be found at Sun's Java web site at http://java.sun.com . Of special interest is the description of the next version of the Java Development Kit JDK™ 1.2 , and within it the information on Swing, standard extensions and the jar: protocol. Slightly more up-to-date information on Swing can be obtained at The Swing Connection's home page.

Your Feedback

We encourage your feedback at javahelp-comments@eng.sun.com.

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

JavaHelp™ 1.0
Send your comments to javahelp-comments@eng.sun.com
Last modified: Mon Apr 12 19:13:58 MDT 1999

JavaHelp™ 1.0 - Overview

Copyright 1998-1999 Sun Microsystems

Introduction

JavaHelp™ 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)
- Applets
- JavaBean components
- Desktops
- HTML pages

This document is an overview of the JavaHelp specification. API documentation generated using `javadoc` can be found starting at api/index.html

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.
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.
Embeddable 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 reference implementation 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.

Serialization

Serialized objects of JavaHelp Swing components will not be support in V1.0. A future release of JavaHelp will provide full serialization including support for long term persistence.

Supported Platforms

JavaHelp 1.0 is a standard extension for both the JDK1.1 and the JDK1.2 platforms.

Although JDK1.1 is in wider use at the time of writing of this document, JDK1.2 is fundamentally a better platform and it offers many features useful to JavaHelp customers, including:

- The `jar:` protocol in the platform, which provides a general and consistent (and efficient) way to refer to files within a JAR file.
- Temporary files, providing improved full-text search performance.
- More flexible ClassLoader classes that can, for example, easily be used to create a ClassLoader instance to a given URL.
- An improved security model.
- Safe access to the SystemEventQueue from an Applet to support some of the Context-Sensitive features.
- Improved I18N support, including input methods.
- Printing support.
- Other improved set of APIs, including sound, 2D and 3D graphics.

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)

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:

Basic Content Presentation	`HelpSet` and `HelpBroker` are used to locate and create HelpSets and then to present these to the user using the default HelpBroker.
Complete Access to JavaHelp Functionality	A number of classes provide for access to Help Data and for control of the navigation of the online content. For example, the `NavigatorView` class provides access to the data in a Navigator View.
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.
Swing Classes	Finally, JavaHelp 1.0 defines Swing 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 Swing components.

Main Concepts

Below we describe the fundamental concepts in the specification. More details are available in other parts of this specification and in the `javadoc` comments of the classes.

HelpSet

A HelpSet is a collection of help content (topics), navigational views, and mapping information. A HelpSet can contain other HelpSets which are merged merged together.

HelpSet File

The HelpSet file describes a HelpSet and contains:

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

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).

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 rules. A view is a subclass of NavigatorView. The `createNavigator()` method of a view returns a component that is used to graphically present the view; for the standard views this component is a Swing component, 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.

Standard Help Views and Help Navigators

All JavaHelp implementations must provide the following classes:

`javax.help.TOCView` `javax.help.JHelpTOCNavigator`	NavigatorView for parsing Table of Contents data and the JHelpNavigator for its presentation.
`javax.help.IndexView` `javax.help.JHelpIndexNavigator`	The NavigatorView and JHelpNavigator for parsing and presenting Index 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 and the Index Navigators are described in FileFormat. 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.

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.

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 JDK1.2 adds the `jar:` protocol which provides access to files within a JAR file. Specific implementations may support additional URL formats.

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 *map file*. Multiple map files can be combined within a HelpSet, but an identifier must be unique within a HelpSet in the resulting combined map.

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`.

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 can 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 Merge for additional information.

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 `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 Customization

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.

File Formats

In summary, the JavaHelp system specifies the following file formats:

- HelpSet encapsulation and compression using JAR files
- HTML topic files - HTML 3.2 minus the APPLET tag (use the OBJECT tag to implement lightweight JComponents in the reference implementation)
- HelpSet file
- Map files
- Standard navigation view formats (TOC, index, search)

More information is available in FileFormat.html.

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 1.0//EN"
     "http://java.sun.com/products/javahelp/helpset_1_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 three sections of the HelpSet file provide information about different views of the content information. The first view, "TheTOC", is in a local disk. The next section is a different Table of Contents view, ("MyLocalTOC"), that uses the same information as the first view, while the next section is an index on the local disk. The final section defines search information.

JavaHelp™ 1.0 - File Formats

Copyright 1998-1999 Sun Microsystems

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.

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
- 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 and Index 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 conformant 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.

HelpSet File

The HelpSet file is localized following the same naming conventions used with ResourceBundle. 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.

Format

HelpSet files are encoded in an XML-based syntax; The DTD is dtd/helpset_1_0.dtd. The top level tag is `<helpset>`. A version attribute is optional, when present its value must be "1.0".

Tag	Description	Allowed In	Body	Attributes	
helpset	HelpSet definition	top-level	none	**xml:lang="lang"**	Language for this item
				version="1.0"	(optional) 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 1.0//EN"
         "http://java.sun.com/products/javahelp/helpset_1_0.dtd">

<!-- next is a Processing Instruction (PI).
  This is ignored by the Reference Implementation -->
<?MyFavoriteApplication this is data for my favorite application ?>

<helpset version="1.0">

    <!-- 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>jar:file:/c:/Program Files/JW3.0/JW3.0.jar!/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).

Processing Instructions

The reference implementation ignores the Processing Instructions.

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

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	empty	location, the spec relative to HelpSet

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

Map Example

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

```
<map>
   <data>jar:file:/c:Program Files/JWS3.0/JW3.0.jar!/TheMap.map</data>
   <data>jar:http://www.sun.com/devpro/JWS3.0Encyclopedia.jar!/TheMap.map</data>
</map>
```

NOTE: There is a bug in the JavaHelp 1.0 reference implementation which only supports one map.

Navigational Views Section

The final sections of a HelpSet file describe the navigational views, which include tables of contents, indices, 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`
name	a name identifying the view	view	text of the name	none
label	a label to show in the presentation	view	text for the label	none
type	a subclass of NavigatorView	view	name of the class	none
data	URL spec	view	text of the spec	optional "engine", a class implementing SearchEngine

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

View Example

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

```
<view>
   <name>TOC</name>
   <name>Table of Contents</name>
   <type>javax.help.TOCView</type>
   <data>jar:file:/c:Program Files/JWS3.0/JW3.0.jar!/toc.xml</data>
</view>
```

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 in Merge.

Tag	Description	Allowed In	Body	Attributes
subhelpset	Static subHelpSet to merge	helpset	empty	location="URL spec to HelpSet file"

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_1_0.dtd. The top level tag is `<map>`. A version attribute is optional, when present its value must be "1.0".

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"	(optional) version
mapID	An individual map entry	empty	map	**target="string"**	ID
				url="string"	URL spec
				xml:lang="lang"	Language for this item

The following is an example of a simple map file:

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

<map version="1.0">
```

```
    <mapID target="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).

Table of Contents

JavaHelp1.0 specifies one table of contents type: `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_1_0.dtd. The top level tag is `<toc>`. A version attribute is optional, when present its value must be "1.0".

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"** **version="1.0"**	Language for this item (optional) version
tocitem	Table of contents item. Tags can be nested to create hierarchical entries.	Text to show in the presentation	toc, tocitem	**target="string"** **image="string"** **xml:lang="lang"**	destination ID destination ID Language for this item

Table of Contents Example

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">
   <tocitem>Introducing JavaHelp
      <tocitem target="api" image="image/document.gif">
         JavaHelp API
      </tocitem>
      <tocitem target="platform" image="image/document.gif">
```

```
         JavaHelp platforms
      </tocitem>
   </tocitem>
</toc>
```

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_1_0.dtd. The top level tag is `<index>`. A version attribute is optional, when present its value must be "1.0".

An index can contain the following two tags:

Tag	Description	Allowed In	Body	Attributes	
index	Index	top-level	empty	**xml:lang="lang"** **version="1.0"**	Language for this item (optional) version
indexitem	Index item. `indexitem` tags can be nested to create hierarchical entries.	index, indexitem	text to show in the presentation	**target="string"** **xml:lang="lang"**	destination ID what language to use

Index Example

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 1.0//EN"
         "http://java.sun.com/products/javahelp/index_1_0.dtd">
<index version="1.0">
   <indexitem>Java Applets
      <indexitem target="applet_over">
         Overview
      </indexitem>
      <indexitem>Usage
         <indexitem target="applet_insert">
            Inserting an applet in a content page
         </indexitem>
         <indexitem target="applet_editing">
            Editing an applet in a content page
```

```
        </indexitem>
      </indexitem>
   </indexitem>
</index>
```

Help Content

JavaHelp displays help topic files formatted using HTML version 3.2. Links are resolved using the URL protocols supported by the underlying platform. Lightweight JComponents can be added to topic pages using the `<OBJECT>` tag.

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>`.

JavaHelp™ 1.0 - Localization

Copyright 1998-1999 Sun Microsystems

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 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()`.

Localized Documents

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

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.

More Details

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

JavaHelp™ 1.0 - Customization

Copyright 1998-1999 Sun Microsystems

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

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 `HelpBroker`s take a HelpSet instance as an argument; `DefaultHelpBroker` uses a JHelp for its presentation, adding to it all the `HelpNavigator`s 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`.

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 Key-Data Map below.

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.

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.

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 acomplished by subclassing TOCView (for instance) and redefining the method getDataAsTree() to return the data whenever invoked.

Different Presentations

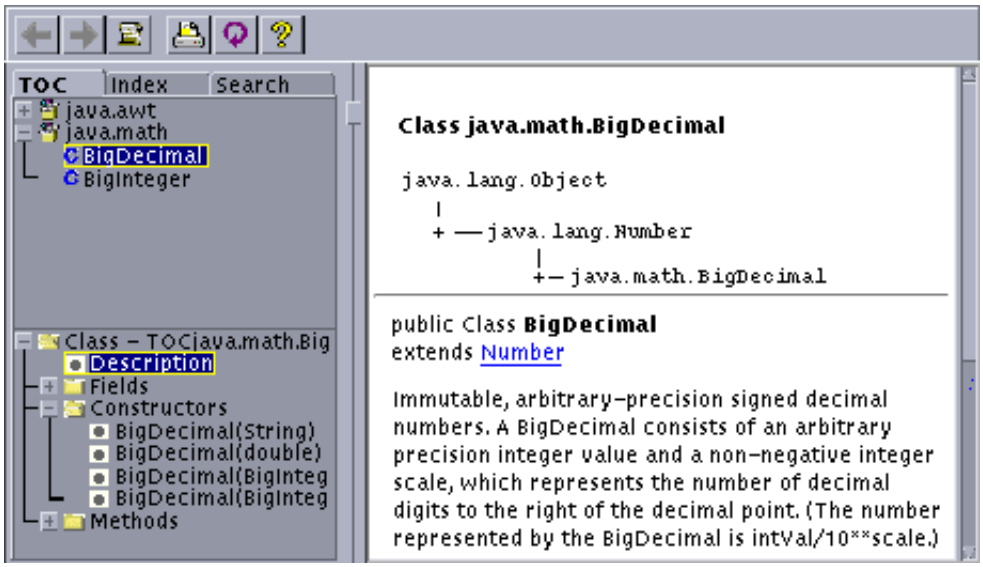
A JHelpNavigator selects its presentation through the standard Swing method `getUIClassID()` to indicate its ComponentUI class. A new JHelpNavigator that is not capable or willing to reuse an existing ComponentUI needs to return an appropriate class value in `getUIClassID()`. If appropriate, this ComponentUI 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.

Two Examples of Custom Views

The three standard Views included in JavaHelp 1.0 (TOCView, IndexView, 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 an early version of the presentation is shown next:



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.

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.

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.

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; it is not possible to do in an Applet in JDK1.1 since there is no support for downloadable URL protocols.

JavaHelp™ 1.0 - JavaBeans Help data

Copyright 1998-1999 Sun Microsystems

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.

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*:

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.

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:

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
```

The HelpSet File

All the HelpSet files are the same. The HelpSet file is quite simple (see below 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>
```

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"
```

An Alternative Arrangement

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

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
```

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>
```

The Help Map

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

JavaHelp™ 1.0 - Context Sensitive Help

Copyright 1998-1999 Sun Microsystems

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 intented 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.

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 `Map`s 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.

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`.

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.

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.

JavaHelp™ 1.0 - Content Search

Copyright 1998-1999 Sun Microsystems

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 moderatedly 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.

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 intented to be of general applicability, it is not intented to be a replacement for more powerful query mechanisms.

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

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

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.

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.

JavaHelp™ 1.0 - Merge

Copyright 1998-1999 Sun Microsystems

Introduction

JavaHelp provides a mechanism for *merging* HelpSets. For example, when two indexes are merged, the second index is appended to the first index. 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 marged.

In addition, a HelpSet file can use the `<subhelpset>` tag to statically include HelpSets, this behavior is identical to `add`ing 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.

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.
- N_b.canMerge(V_a) is invoked to determine if the views can be merged.
- If then can be merged, then N_b.merge(V_a) is invoked.

If later the HelpSet A is removed from HelpSet B:

- N_b.remove(V_a) will be invoked.

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.

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.

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 `SearchEngine`s are collated and presented together.

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

JavaHelp™ 1.0 - Change History

Copyright 1998-1999 Sun Microsystems

Changes from 0.90 to 1.0

This is the final set of changes so they are documented in more detail than the other sets.

- Refined the documentation of several methods. Fully defined when NullPointers are allowed and the resulting actions.
- The method getIcon in BasicSearchCellRenderer was made private.
- The method getCurrentNav was added to BasicHelpUI and HelpUI.
- The method setNavigatorDisplay was removed from HelpUI and BasicHelpUI.
- The method reloadData was changed to private in BasicTOCNavigatorUI, BasicIndexNavigatorUI, and BasicSearchNavigatorUI.
- The method installLookAndFeelDefaults was changed to package in HelpUtilities.
- Changed occurances of IllegalArgumentException to throw NullPointerException, except for classes that were copied from Swing.

Changes from 0.70 to 0.90

This is intented to be the last set of changes so they are documented in more detail than the other sets.

- Package name is now javax.help, rather than javax.javahelp. This is consistent with other packages in the Java Platform.
- Adjusted examples so they follow the Reference Implementation, in particular, the package names in the RI have also changed from *.javahelp.* to *.help.* and the default search database directory has changed from JavaHelpSearch to JavaHelpSearch.
- Better use of XML:
  - Moderate cleanup of HelpSet format to improve consistency and to better employ XML features, including Processing Instructions and empty tags.
  - Removed <impl> tag from the HelpSet (use PIs instead).

    - Dropped claim that the keydata registry of HelpSet would be initialized from the <impl> section.
    - Map files are now described as XML files.
    - All XML formats (HelpSet, TOC, Index, Map) are now formally described using a DTD. DTDs are versioned to allow for future changes.
- Standard Views now can be "empty" (i.e. without any <data>; this is useful when merging HelpSets) and clarified that non-existing subHelpSets are not to give an error.
- The DialogSupport class is no longer in the specification as it had too many limitations. We are planning direct support for Help in a future version of Swing's JOptionPane.
- CSH methods now accept both AWT Components as well as JComponents.
- The default CSH listener classes moved from DefaultHelpBroker to CSH and changed their name.
- The enabling methods on HelpBroker were modified to work on AWT components and some also changed their names.
- Refinements to the API to either fix bugs or improve reusability of functionality:
    - TOCView and IndexView now include a `getDataAsTree()` method.
    - Small changes in HelpSet to allow the creation of HelpSets programatically.
    - Cleaned up some more the parsing interface used in HelpSet, TOCView and IndexView.
    - Normalized a number of exceptions in corner cases (like when a null is passed in, etc).
    - Added `removeNavigator()` to several UI classes, `setCellRenderer()` to the ComponentUI classes, and `merge()` and `remove()` to `BasicSearchNavigatorUI`.
    - Made some additional methods public or protected so they can be reused.
    - Promoted UnsupportedOperationException from an inner to a top-level class.

Changes from 0.50 to 0.70

- Significantly improved support for internationalization and localization.
- We no longer claiming to comply with RDF, although we are still trying to be consistent with it.
- Created the notion of a NavigatorView to provide additional flexibility.
- Added the notion of Factories to access TOC, Index, and HelpSet parsers, including errors.
- Added support for a `<subhelpset>` tag in HelpSet files.
- Added a method to HelpUtilities to better support `jar:` in JDK1.2.
- Switched to using `<OBJECT>` tags instead of `<APPLET>` tags
- `<OBJECT>` tags are no longer part of the specification
- Clarified the role of the search engine.
- Added a section to this specification about the reference implementation.
- Classified Scenarios as a companion doc to the specification.
- Changed the package name of the search engine in the reference implementation.

Changes from 0.33 to 0.50 (unpublished but implemented in jhEA2)

- Added proper XML prefix to XML-based formats (TOC, Index, HelpSet).
- Now supports both JDK 1.1 and JDK 1.2 platforms.
- Removed mandated protocols from specification; instead we rely on the underlying platform.
- Added HelpBroker, Context-Sensitive Help, Search API.
- Dropped *format* from Views; *type* is enough.
- HelpSet nest (for merging) and IDs are now scoped within them.

Changes from 0.30 to 0.33

- First public draft.
- Improved the Beans proposed mechanism
- Cleaned up some the Merge examples
- Clarified some activation issues
- Added some references to the forthcoming Search Database Spec.

Changes from 0.20 to 0.30

- Improved pictures in the Scenarios. Improvements to the language of the document.
- The Navlet API was folded into JHelpNavigator and cleaned up
- Improvements on merging of HelpSets.
- Added initial description of DB format.
- Added Changes.html file.
- Added an intro to the JavaSoft API evolution process in preparation for a more public draft release.

Changes from 0.10 to 0.20

- Added a number of JavaHelp usage scenarios.
- Improved on applicability of map files using jar: format.
- Added navigational views.
- Added an overview to the spec.
- First cut at JavaBeans support.
- Provided examples of code fragments.

JavaHelp™ 1.0 - JavaHelp Class Structure

Copyright 1998-1999 Sun Microsystems

Packages

JavaHelp is a standard extension for JDK1.1 and JDK1.2. 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.search	search classes.

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

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

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.

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.

Extensibility

Content extensibility is described through a `NavigatorView` which provides access to some context information plus a way of presenting this information. `TOCView`, `IndexView`, 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.

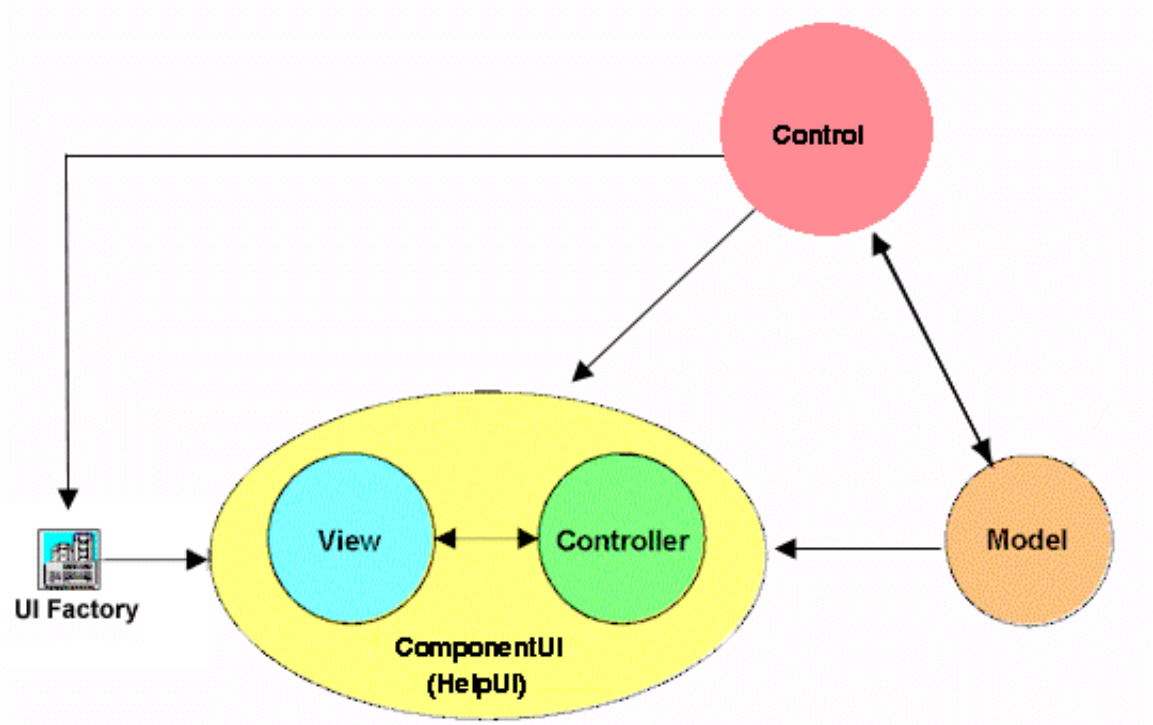
Swing components

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 `UIFactory`. When a new Swing component is created, it asks the current `UIFactory` 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.

Context Sensitive Help

JavaHelp supports a `Map` between identifiers and URLs. `FlatMap` and `TryMap` are two implementations; sofisticated 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`.

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.

JavaHelp™ 1.0 - Scenarios

Copyright 1998-1999 Sun Microsystems

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 and deployment	Scenarios that describe different ways that the JavaHelp system can be used to present help information. These scenarios also illustrate different methods for deploying the JavaHelp system classes and help data.
Search	Scenarios that describe different ways that full-text searches of JavaHelp system information can be implemented
Packaging	Scenarios that describe different ways that JavaHelp system data can be encapsulated and compressed using Java Archive (JAR) files
Merging HelpSets	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 reference implementation release available at http://java.sun.com/products/javahelp.

Invocation Mechanisms

These scenarios describe the different ways the JavaHelp system can be invoked.

Menus and Buttons

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 activation 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.

Tooltips

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

Although tooltip information could be included in the JavaHelp system data, it will usually be delivered as part of the application and will be co-located with the code.

Context-Sensitive Help

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 CSH.html for more details.

Helpers

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. This is not directly supported in the 1.0 release of the JavaHelp system.

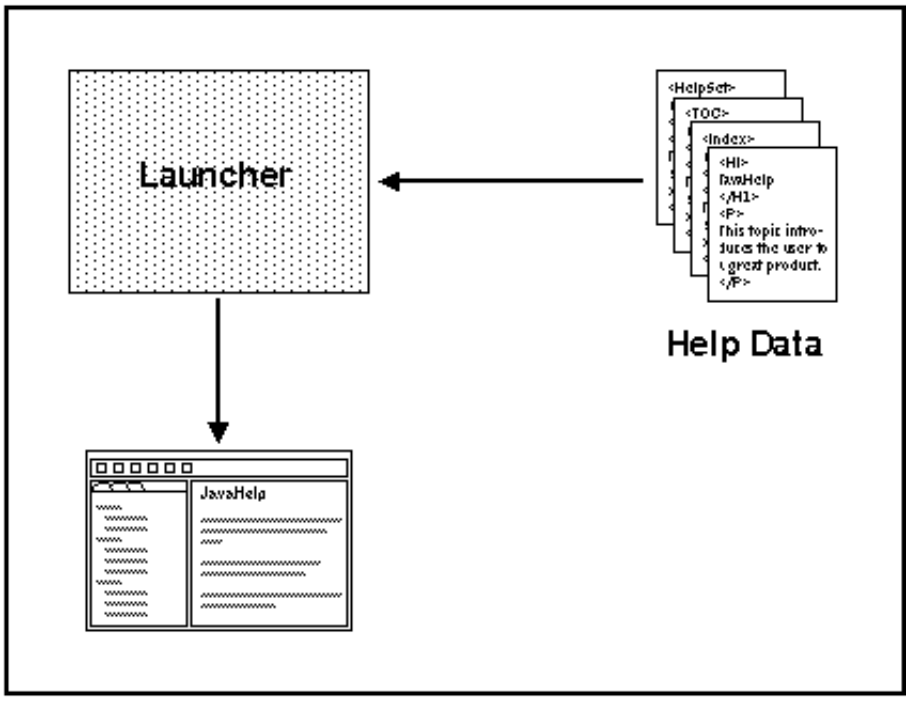
Presentation/Deployment Scenarios

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

Information Kiosk

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

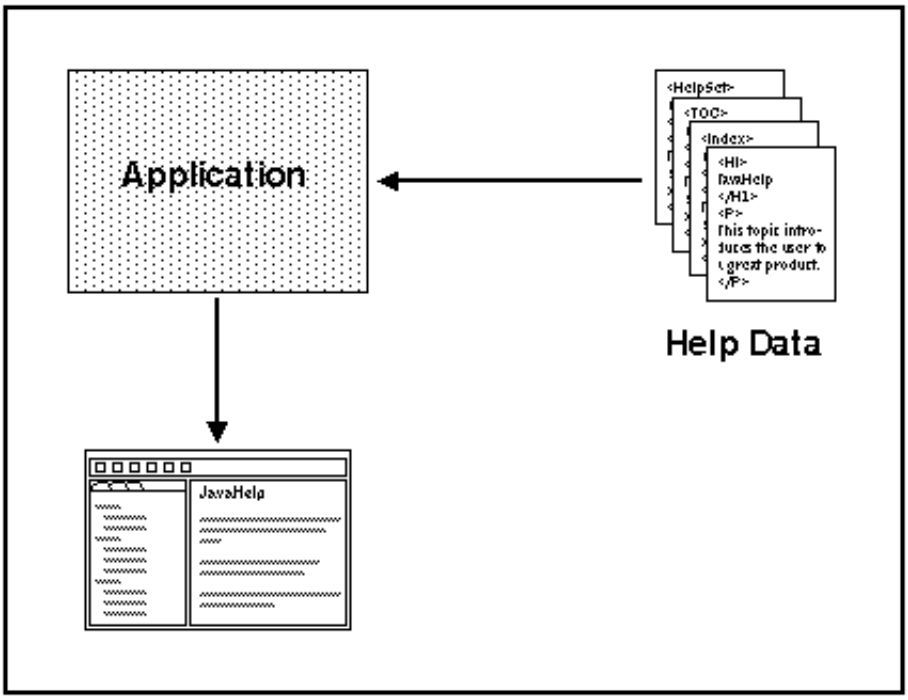
An example on the Solaris platform is AnswerBook -- a technology used to display all of Sun’s documentation online. All that is required is a help browser that can be launched to present and navigate through the information.



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).

Stand-Alone Application

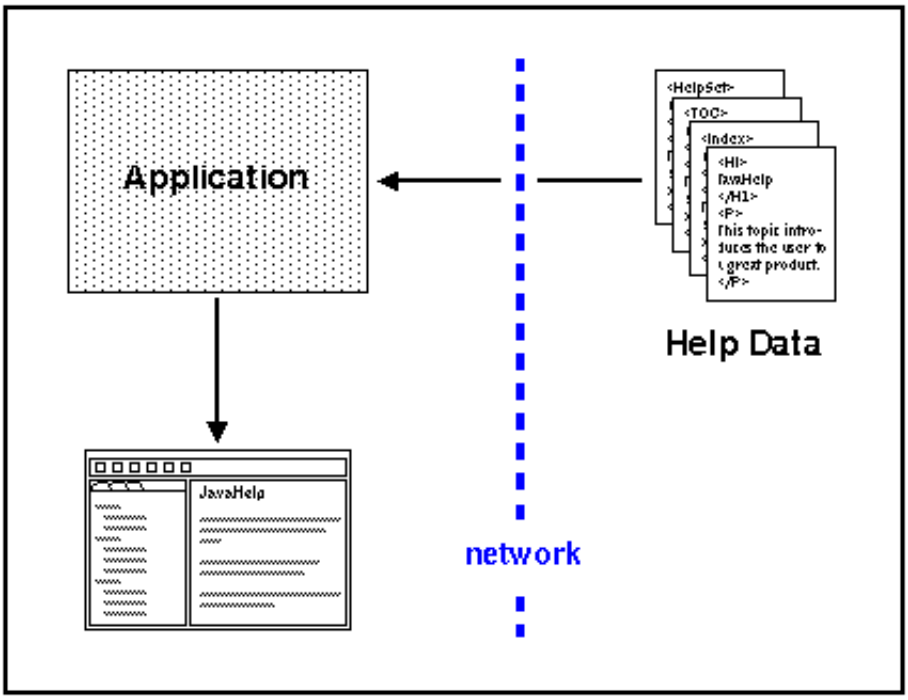
The simplest scenario is one in which the Java application runs locally and accesses help data installed on the same machine.



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.

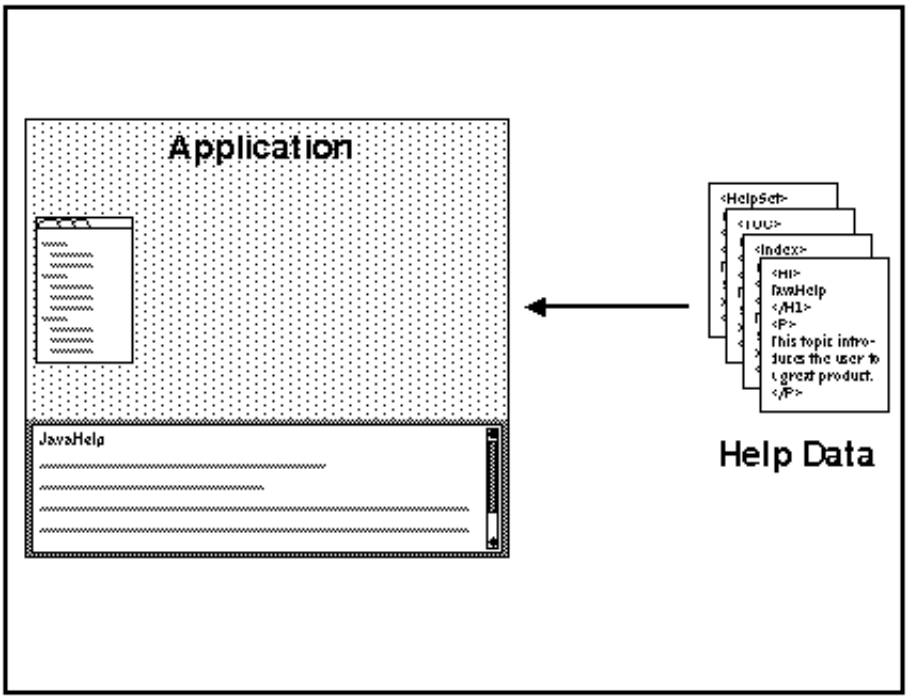
Network Application

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



Embedded Help

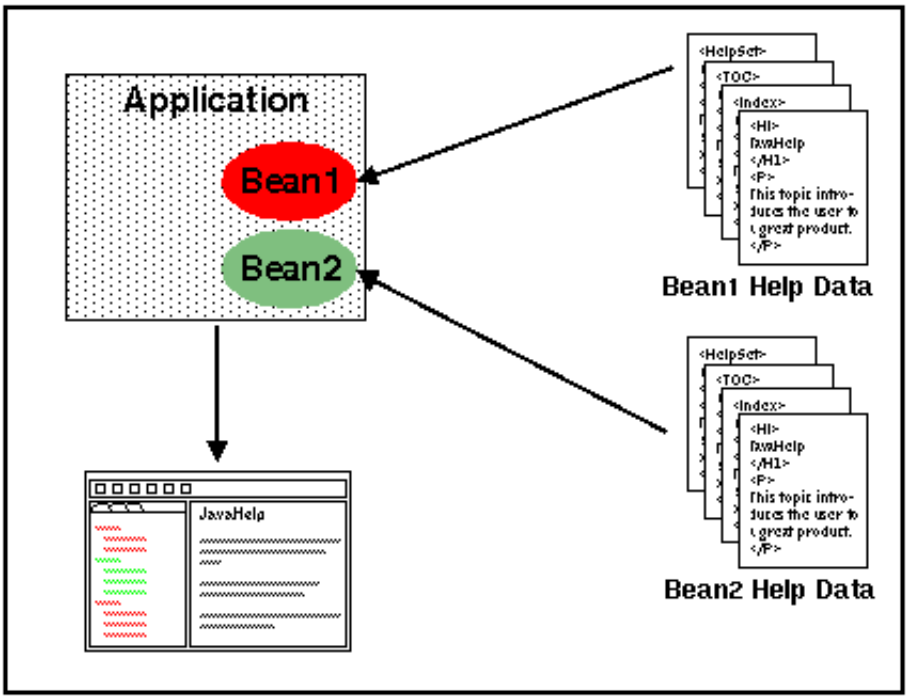
Information can also be presented embedded directly in application windows. The JFC components that implement the JavaHelp specification are embedded directly into the application frame. The application can create its own customized presentation, by using the JFC components from the reference implementation.



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.

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.

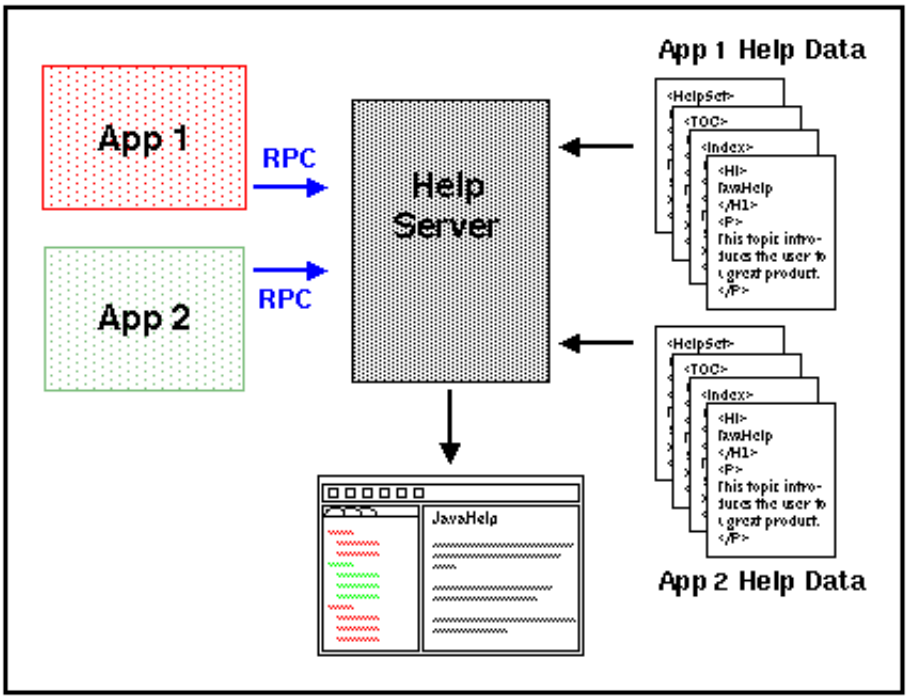


The help information can be merged in different ways. For instance, the table-of-contents, index, and full-text search information from each component may be displayed one after the other to create a new, unified help system.

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 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).



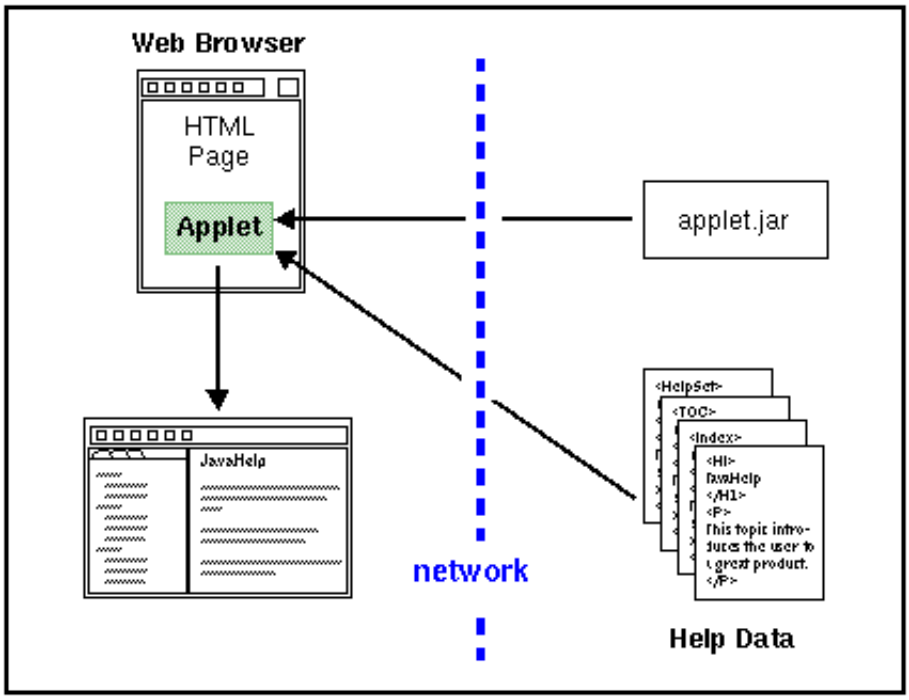
The help server model is useful if the application is not written in Java and does not include a JVM. It would also be useful for a suite of Java applications that can share a common help server.

Web Pages and Applets

The final 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()`.

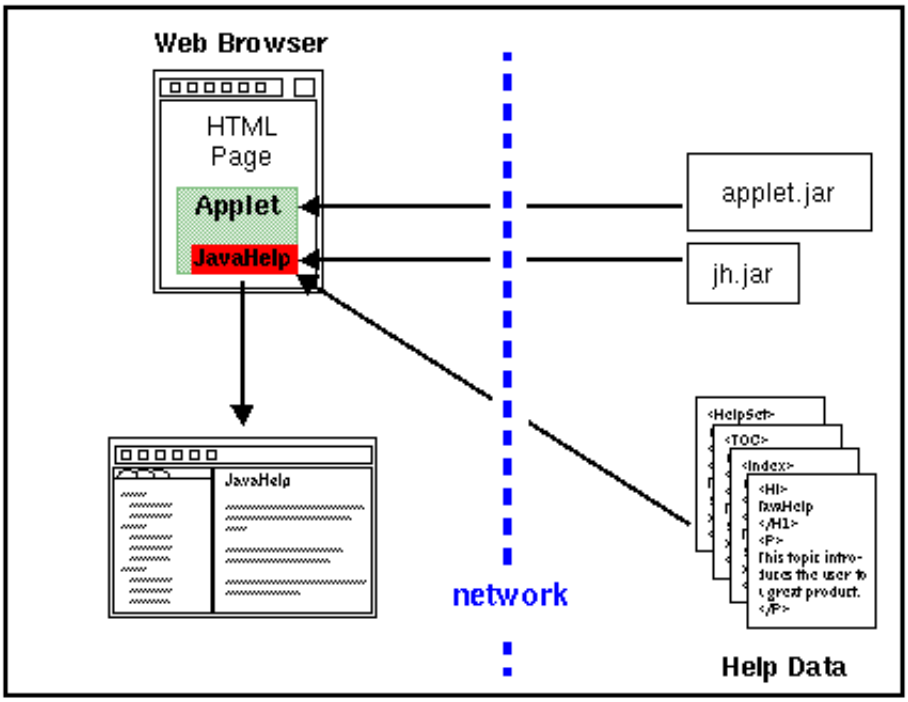
This scenario can have a number of variations. Here are a three:

- 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.

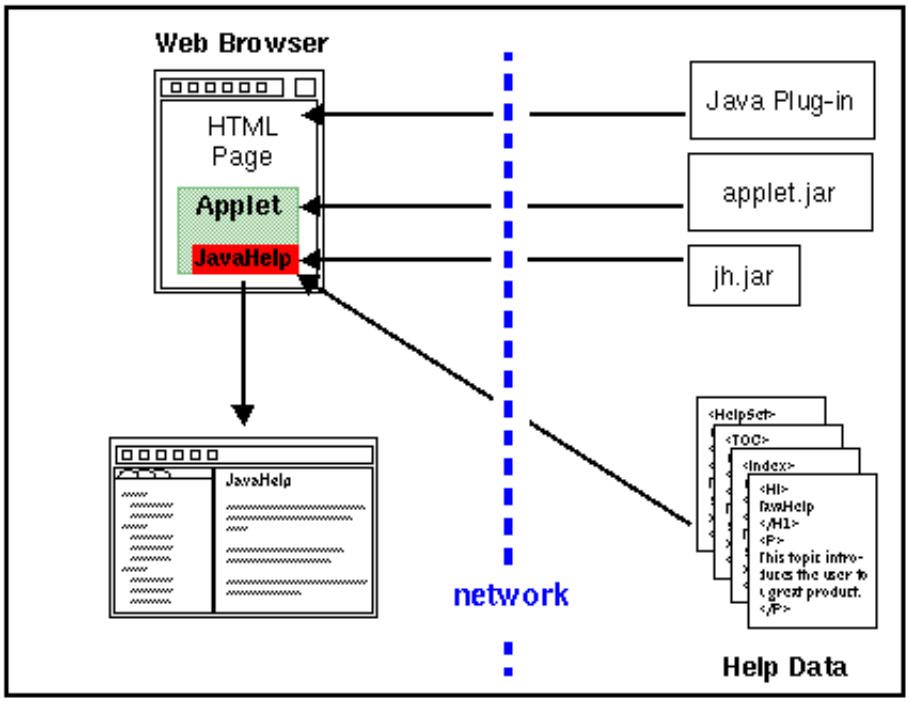
  This situation is depicted in the next picture where, for variety sake, we have changed the help presentation so the navigator is separate from the content.



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-conformant 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:



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

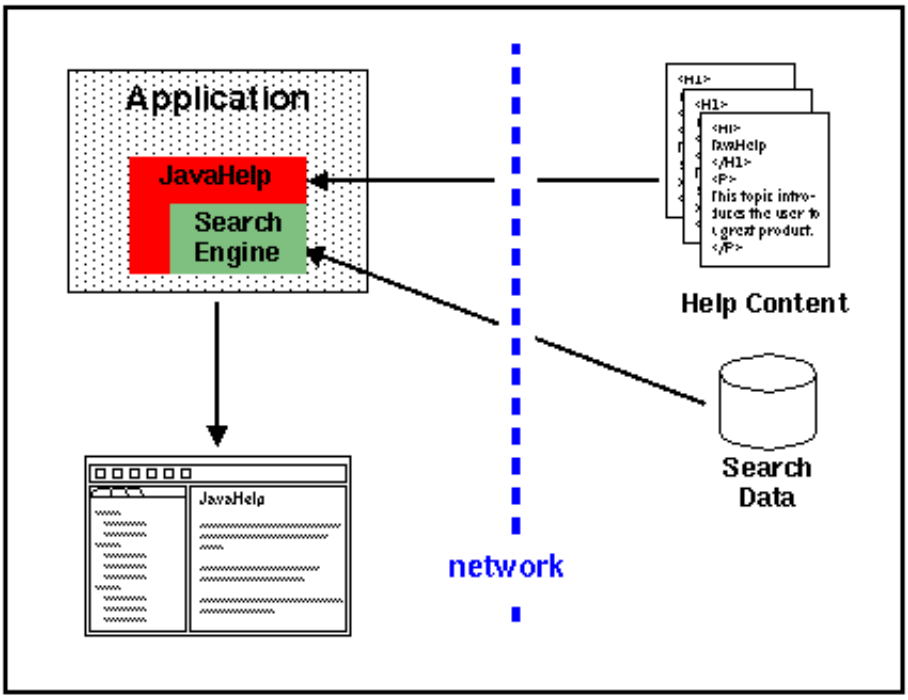
Search Scenarios

The JavaHelp system supports an extensible full-text search mechanism using the extension framework 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 search	The search database is downloaded from the server, then searched on the client
Server-side search	The search database and search engine are located on the server
Stand-alone search	The search database is included as part of the HelpSet and the search occurs in the application

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.

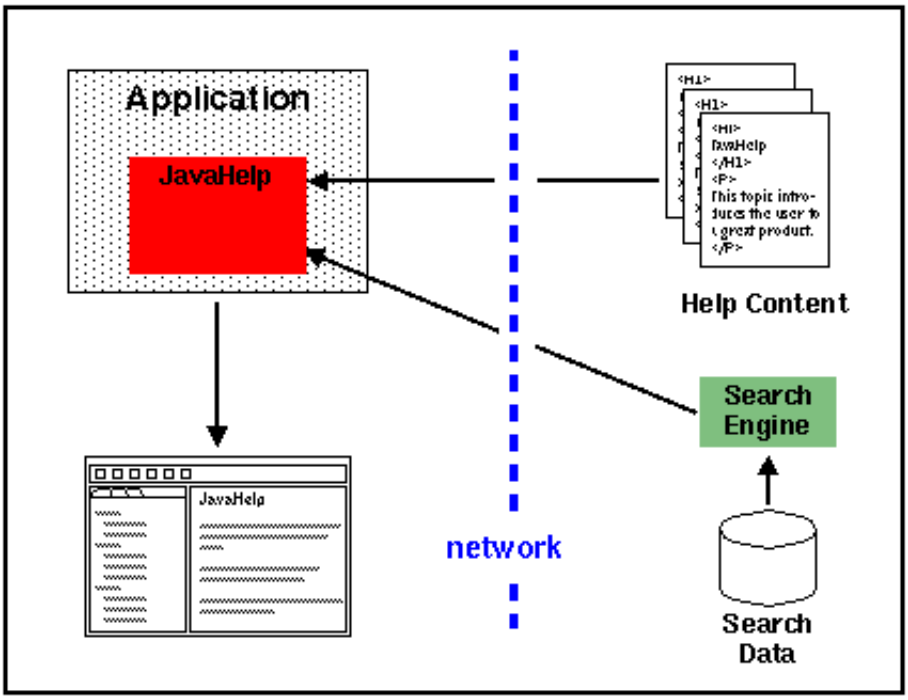


1. Search is initiated
2. Search engine loads database
3. Search engine searches database and delivers "hits" to application
4. User (or application) chooses a "hit"
5. Content is loaded and displayed

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.

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.

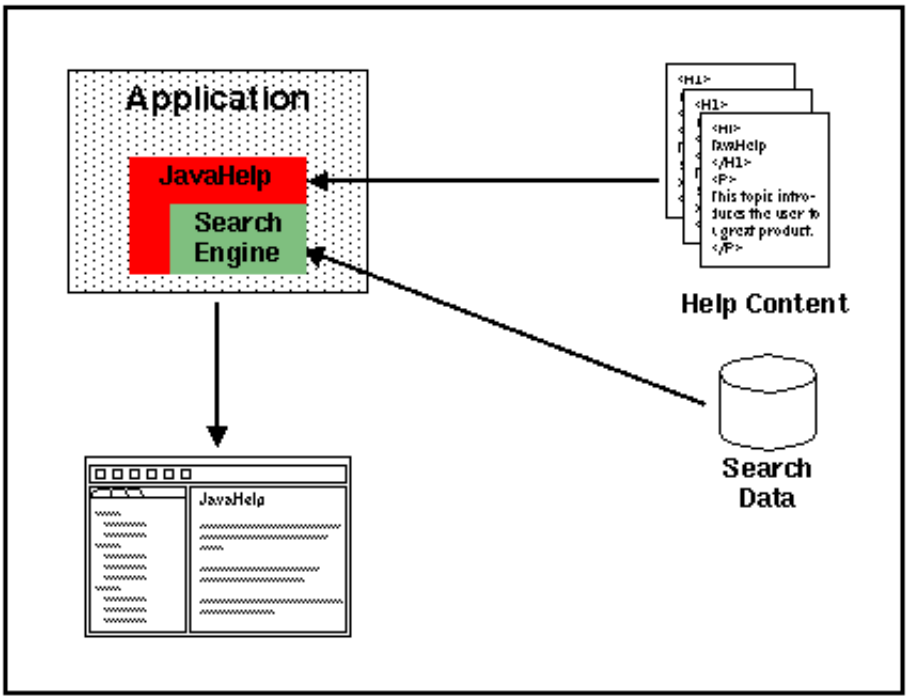


1. Search is initiated
2. JavaHelp requests search from server
3. Server-side search engine searches database and delivers "hits" to application
4. User (or application) chooses a "hit"
5. Content is loaded and displayed

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.

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.



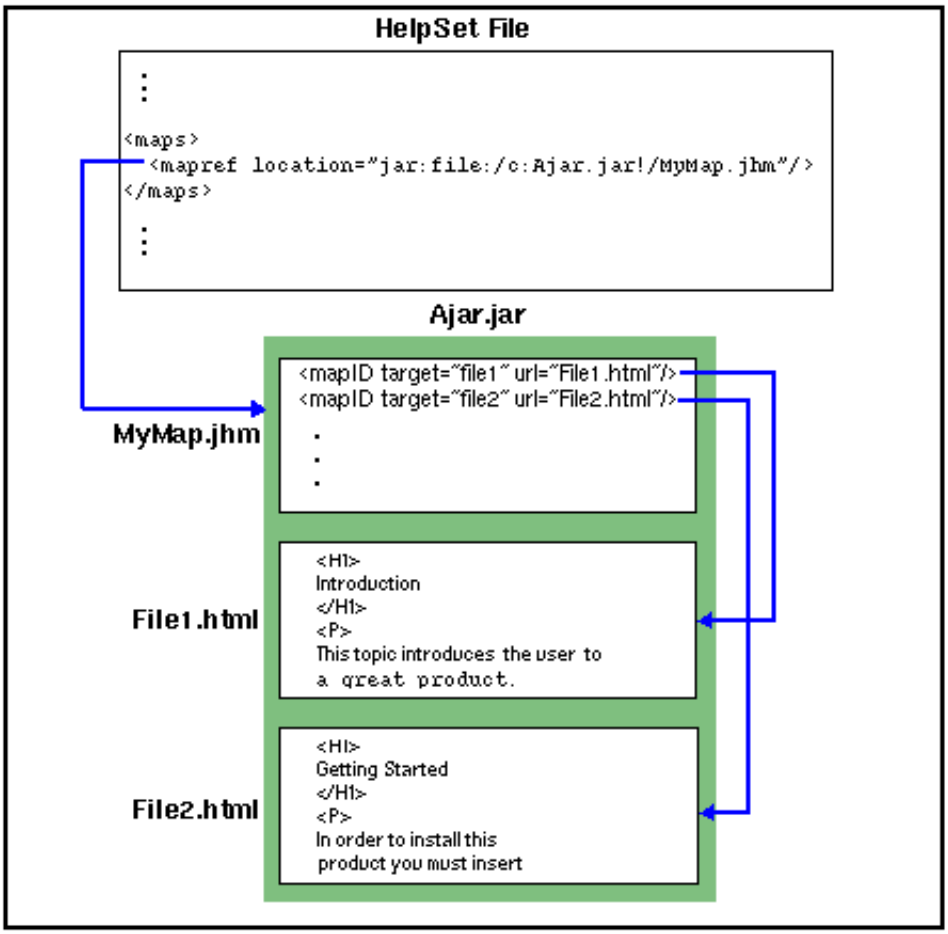
1. Search is initiated
2. Search engine loads database
3. Search engine searches database and delivers “hits” to application
4. User (or application) chooses a “hit”
5. Content is loaded and displayed

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

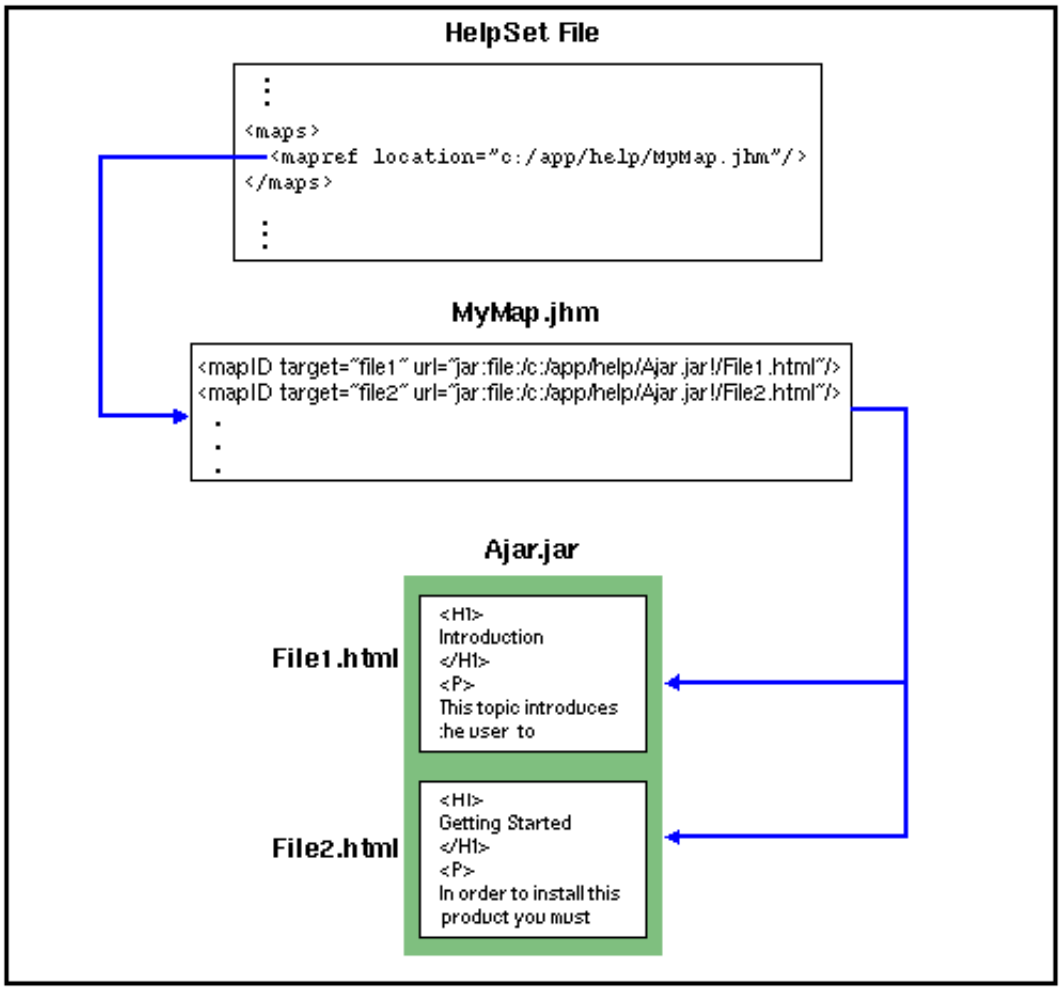
Packaging Scenarios

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

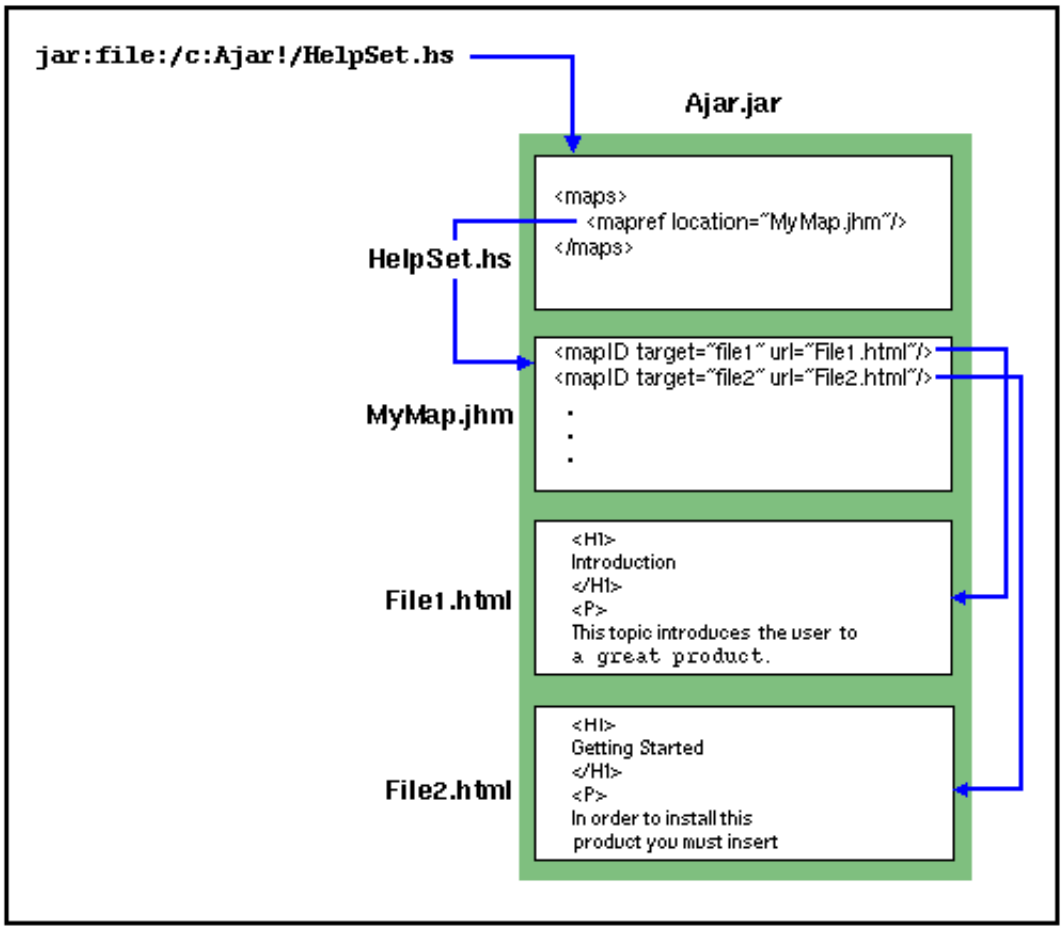
The first picture represents a project in which the map file is packaged together with most (all?) of the content files. The "!" syntax is used to specify the URLs relative to the JAR where the map is located. The HelpSet file is packaged outside of the JAR file, perhaps to simplify updates later on.



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.



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



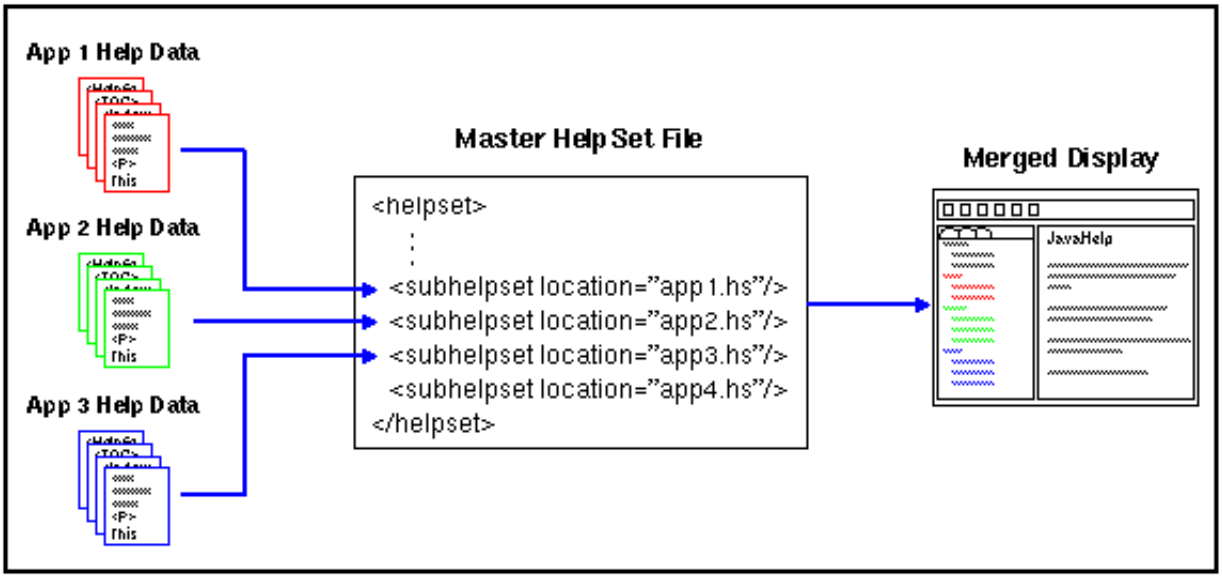
The advantage of this arrangement is that all the URLs are relative to the base URL of the HelpSet file, and that there is no need to mention the jar: protocol within any JavaHelp system file. This JAR, when placed in a CLASSPATH, permits a JDK1.1 application to refer to the HelpSet within the JAR file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

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.



For more information about merging see Merge in the specification or "Merging HelpSets" in the JavaHelp System User's Guide.

JavaHelp™ 1.0
JavaHelp System 1.0 Reference Implementation

Copyright 1998-1999 Sun Microsystems

JavaHelp System 1.0 Reference Implementation

Sun's reference implementation of the JavaHelp system implements the 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 AWT Java Components that 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`.

HelpBroker

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

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 Relaxation Searching.

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.

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.

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.

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.

JavaHelp™ 1.0
Send your comments to javahelp-comments@eng.sun.com
Last modified: Mon Apr 12 19:17:03 MDT 1999

JavaHelp™ 1.0 - Java Components

Copyright 1998-1999 Sun Microsystems

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

---

JavaHelp™ 1.0
Send your comments to javahelp-comments@eng.sun.com
Last modified: Mon Apr 12 19:08:36 MDT 1999

JavaHelp™ 1.0 - Relaxation Searching

Copyright 1998-1999 Sun Microsystems

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.

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.

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.

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 nominalization). 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.

JavaHelp™ 1.0
Proposed jar: Specification

Copyright 1998-1999 Sun Microsystems

NOTE: the main body of this document is taken verbatim from the original 1.2 proposal.

JAR URL Syntax Proposal

Summary

The following is a proposed URL syntax for representing JAR file entries. This new URL syntax can be used to refer to resources stored in both local and remote JAR files, and also replaces the need for the systemresource protocol handler.

Absolute URLs

An absolute JAR URL has the following syntax:

```
abs-jar-url = "jar:" jar-file-url "!" [ sub-jar-path ] [ entry-path ]
jar-file-url = abs-url ( absolute URL of JAR file )
sub-jar-path = entry-path "!" [ sub-jar-path ]
entry-path = abs-path ( absolute path of JAR file entry )
```

The optional component ’sub-jar-path’ can be used to refer to JAR files embedded within JAR files. If ’entry-path’ is omitted then it defaults to the root path entry ’/’.

Examples,

```
jar:http://foo/foo.jar!/foo/bar/gif
jar:http://foo/foo.jar!/baz.jar!/foo.gif
```

The seconds example refers to a JAR file ’baz.jar’ embedded within the JAR file at URL http://foo/foo.jar.

Since the ’!’ character has special meaning within a JAR URL then it must be encoded if appearing as the last character in a JAR entry path or the embedded JAR file URL.

A JAR URL without any ’!’ character is equivalent to the URL obtained by removing ’jar’ scheme. For example,

```
jar:http://foo/foo.html -> http://foo.foo.html
```

This makes it possible for a relative URL to "jump out" of a JAR URL if necessary.

Relative URLs

When resolving relative URLs against an absolute JAR URL as the base URL, similar rules are employed as with file URLs. The one exception is if the character ’!’ appears alone as a component of the relative URL then it will cause every component following the last ’!’ to be removed from the absolute URL. This makes it possible to immediately jump to the root of the last embedded JAR file.

For example, if the base URL is ’jar:http://foo/foo.jar!/foo/bar.html’ then the following relative URLs as resolve as follows:

```
baz.html        -> jar:http://foo/foo.jar!/foo/baz.html
!/../bar.jar!/  -> jar:http://foo/bar.jar!/
!/../foo.gif    -> jar:http://foo/foo.gif -> http://foo/foo.gif
/bozo.html      -> jar:http://foo/bozo.html -> http://foo/bozo.html
```

JDK1.2 support JAX files

With JDK1.2, the invocation of kiosks can be made even simpler through the use of the JAR auto-invocation mechanism. The underlying platform’s process start-up mechanism distinguishes a class file with a `main()` entry in the JAR file’s MANIFEST. Then the platform’s normal mechanism (double click, explicit launch, magic number) is used to launch the file. A JAX file is a JAR file intented to be used in this fashion with a different extension; this is useful with the standard win32 "launcher by extension" mechanism.

JavaHelp™ 1.0 - Copyright

Sun Microsystems, Inc.

Copyright 1998-1999 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, nontransferable, worldwide, limited license (without the right to sublicense) under SUN’s intellectual property rights that are essential to practice the JavaHelp 1.0 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.

Last modified: Mon Apr 12 16:20:07 MDT 1999