Technote 1141

Extending and Controlling Sherlock

Document 1.2

By John Montbriand
Apple Worldwide Developer Technical Support

CONTENTS

Mac OS 8.5 includes several enhanced searching capabilities, known collectively as **Sherlock**. Previously, the Mac OS Find application allowed users to search mounted disk volumes for files based on information such as name, modification date, and file type. Sherlock retains this functionality, but also extends the user's search options to include both the content of files and the Internet.

Sherlock 2 adds a number of new features to the array of search options presented to the user. To accommodate those new features, some additions have been made to the Internet Search Plug-in language, new applescript commands have been added. Where appropriate, these new features are described in this document.

Find by Content library information formerly found in this note has been moved to Technote [TN1180, "Sherlock's Find By Content Library."]

Overview

To perform an Internet search, the Sherlock application sends query information to one or more Internet search sites. The information returned by the search sites is interpreted by the Sherlock application and then displayed for perusal. As each Internet search site has its own particular format for query and response information, the Sherlock application uses plug-ins that describe data formats expected and provided by individual Internet search sites for formatting queries and parsing response data. Internet search site providers interested in building their own Internet search site plug-ins will find directions for doing so in the Internet Search Plug-ins section.

AppleScript commands for accessing the new content-based search and Internet search facilities provided by the Sherlock application are available. These include commands for searching by content, a command for indexing volumes, and commands for performing Internet searches. These commands are discussed in greater detail in the AppleScript Support section.

The Sherlock application, when asked to open a file that was found by way of a content-oriented search, attaches information about the search and why the file was selected to the 'odoc' Apple event it passes to the Finder. The Finder passes this information along to applications as a property associated with the `'odoc'` Apple event. Applications can access this information and use it to perform further search and display actions when it is found in the `'odoc'` event. More information can be found in the kAEOpenDocuments section.

Find By Content is a new system-level facility implemented as a Code Fragment Manager library. The Sherlock application is a client of Find By Content and utilizes its search facilities for performing content-based searches. Developers interested in using the Find By Content services from within their applications may do so by linking against the PowerPC Code Fragment Manager library named "Find By Content" (without the quotes). Routine descriptions and examples are provided in the Find By Content section below.

Internet Search Plug-ins

The "Search Internet" feature in the Sherlock application allows users to perform Internet searches using one or more Internet search engines. The Sherlock application itself contains no information about the exact data formats expected or generated by individual Internet search engines; when accessing any particular Internet search site, the Sherlock application uses a search plug-in file that describes the data formats both expected by the site for queries and produced by the site in its responses to queries. Internet Search Interface Language (ISIL) is the language used in search plug-in files so that Internet search site administrators may provide their own search plug-in files.

ASCII text describing the search site is contained in a search plug-in's data fork. The resource fork may be used for custom icons, Finder strings, et cetera. Search plug-in files have the creator code `'fndf'` and the file type `'issp'` and will be only recognized by the Sherlock application when they reside in the "Internet Search Sites" Folder (FindFolder type = `'issf'`). When dropped onto the System Folder's (closed) icon, files of type `'issp'` are autorouted to the "Internet Search Sites" folder.

ISIL is modeled closely after the HTML it is used to describe, so HTML authors familiar with the syntax should have little or no trouble creating their own search plug-in files. An exact specification of the language can be found in the Internet Search Interface Language BNF section, and the sections that follow discuss the language in greater detail.

To create a search plug-in file, you will need a text editor program—Simple Text will do—and a utility that will allow you to change the plug-in's file type. The basic steps for editing a search plug-in file are:

1. Open or create and then edit the file using your text editor program.
2. Save any changes you make and close the file.
3. Change the file type of the file from `'TEXT'` to `'issp'.`
4. Test your file (now a Sherlock plug-in) using the Sherlock application.
   If satisfied, you're done: stop.
5. Change the file type of the search plug-in from `'issp'` to `'TEXT'.`
6. Go to the first step in this list.

If your text editor edits any file regardless of type and does not change the types of the files it edits, you can skip steps 3 and 5.

The Sherlock application scans the "Internet Search Sites" only once when it is starting up. You should restart the Sherlock application each time you would like to test your search site file.

Back to top

Search Plug-in Files

Search plug-in files contain ASCII text formatted similarly to the HTML text used to define web pages. Accordingly, terminology used to describe HTML is used in this document's description of ISIL syntax. Information describing an Internet search site is contained in a block labeled with the SEARCH tag. This block is used to describe how the Sherlock application sends queries to an Internet search site, and it includes information such as the site's URL, the HTTP command used to send a query, and query parameters. Listing 1 illustrates the typical layout for a SEARCH block.

```
<SEARCH
    name = "<search engine name>"
    method = ["get" | "post"]
    action = "<url to address>"
    [update = "<url containing update file>"]
    [updateCheckDays = "<days between update pings>"]
    [description = "<human-readable-description">]>

....

<INPUT
    name = "<input name>"
    value = "<value>"
    [mode = "results"]>
<INPUT
    name = "<input name>"
    value = "<value>"
    [mode = "browser"] >

....

<INPUT
    name = "<input name>"
    user>

....

<INTERPRET

    [relevanceStart = "<text>"]
    [relevanceEnd = "<text>"]

    [resultListStart = "<text>"]
    [resultListEnd = "<text>"]
    [resultItemStart = "<text>"]
    [resultItemEnd = "<text>"]
    [skipLocal=true]

    [charset = "<text>"]
    [resultEncoding = <integer>]
    [resultTranslationEncoding = <integer>]
    [resultTranslationFont = "<text>"]>

....

</SEARCH>
```

Listing 1. Typical layout for a SEARCH block in a search plug-in file.

Search blocks begin with the <SEARCH ....> tag (containing a number of attributes, as described in Table 1) and end with a </SEARCH> tag. Within a typical search block describing an Internet search site, there will be one or more INPUT tags and an INTERPRET tag. The SEARCH block attributes describe the search site, how it is to be accessed, and where updates to the search plug-in file can be found.

Table 1. `SEARCH` block attributes.

Attribute Name	**Description**
`name`	This is a human-readable name for the search plug-in.
`method`	The `method` attribute specifies the type of HTTP command that should be used for communications with the HTTP server. Currently, either `"GET"` or `"POST"` can be specified as the communications method.
`action`	Specifies the full URL to access the search server. Any relative links in the result list will be localized using this URL.
`update`	This is an optional attribute specifying where the most recent version of the search plug-in file is kept. If provided, the Sherlock application will periodically check this URL for changes. If the file at this URL is found to be more recent than the one that is currently installed, the Sherlock application will prompt the user to download the new file and automatically install it. The file located at this URL should be in BinHex format (but not otherwise compressed or encoded).
`dateCheckDays`	This is an optional attribute specifying the number of days between times when the `update` URL is checked for more recent versions of the search plug-in file. If this attribute is not present, the default value of 30 days is used.
`description`	This is an optional attribute containing text describing the search engine, its capabilities, and the content type of the search results. This text may be used for display in user interface facilities.
Additional `SEARCH` block attributes for Sherlock 2	
`routeType`	Specifies which Channel the internet search site plug-in belongs in. The System will route internet search site plug-ins to the "Internet Search Sites" folder of the System Folder. Sherlock 2 will move plug-ins from the "Internet Search Sites" folder to the appropriate Channel folder. If a plug-in does not have a routeType attribute, then it will be move the "My Channel" Channel. If a plug-in has a routeType attribute, then it will be moved to the appropriate channel. Route types for the default Channels are as follows:

routeType Specification	**Channel**
routeType="internet"	Internet
routeType="people"	People
routeType="apple"	Apple
routeType="reference"	Reference

routeType="news"	News
routeType="shopping"	Shopping

The `INPUT` tags are used to construct the data field used in the GET or POST command sent to the HTTP server. The data field is constructed using the HTTP syntax and the `method` field determines the method that is used to query the server. A search block may contain one or more `INPUT` tags, but only one of the `INPUT` tags can be a `USER INPUT` tag.

`INPUT` tags may specify an optional mode attribute. The Sherlock application will send two types of queries: one when it is retrieving results and another when it sends a query URL to a browser. `INPUT` tags specifying the "results" mode (the default) are used by the Sherlock application when it sends queries to search sites that will be displayed in the list of search results in the Sherlock application's window. `INPUT` tags specifying "browser" will be included in query URLs sent to browser applications for display. For example, the following two `INPUT` tags may be present in a search plug-in file:

```
<input name="sv" value="AP" mode = "results">
<input name="sv" value="IS" mode = "browser">
```

Here, &sv=AP will be sent to the server when the Sherlock application will be used to display the results, and &sv=IS will be sent to the server when a web browser will be used to display the results.

The `INTERPRET` tag describes the format of the information returned from search queries sent to the site. This information allows the Sherlock application to extract individual search results from a query and format them into a list. Table 2 describes the various attributes that may be specified for an `INTERPRET` tag. Each attribute specified in the `INTERPRET` tag specifies a text pattern occurring in the result page delimiting some specific part of the results. When available, the Sherlock application will use these text patterns to extract search result information from the result pages returned by Internet search sites and build lists of items for display.

Table 2. `INTERPRET` tag attributes.

Attribute Name	**Description**
`resultListStart`	Specifies the text pattern present at the beginning of the list of search results in the result page returned by the server. If `resultListStart` is not specified, then the Sherlock application will assume the result list begins at the top of the result page.
`resultListEnd`	Specifies the text pattern present at the end of the list of search results in the result page returned by the server. If `resultListEnd` is not specified, then the Sherlock application will assume the result list ends at the bottom of the result page. The `resultListStart` and `resultListEnd` attributes are used to define text patterns delimiting the list of results.
`resultItemStart`	Specifies a text pattern present at the beginning of each individual item in the list of results. When the text specified is matched in the result page, only links immediately following the text pattern will be included in the list of results displayed for the user.
`resultItemEnd`	Specifies a text pattern present at the end of the text used to describe an item in the list of results. Text between a result's link and this text pattern will be presented in the details pane.

	The `resultItemStart` and `resultItemEnd` attributes are used to define text patterns delimiting individual items in the list of results returned by the server.
`relevanceStart`	Specifies a text pattern marking the beginning of the relevance information provided for each item in the list of results. When present, the first numeric text found after the pattern will be interpreted as the relevance of the item. Note: the numbers used to represent relevance scores should be between 0 and 100.
`relevanceEnd`	Specifies a text pattern marking the end of the relevance information. The search for relevance information will not proceed beyond this text pattern. The text patterns defined in the `relevanceStart` and `relevanceEnd` attributes are used to delimit the relevance score for each individual search result. Note: the numbers used to represent relevance scores should be between 0 and 100.
skipLocal	`skipLocal` is a boolean attribute. If `skipLocal` is true, then the Sherlock application will ignore links that refer to the same host as specified in the `ACTION` attribute in the `SEARCH` tag.
`charset`	The expected encoding of the HTML results. This attribute may be set to any value appropriate for the `charset` HTML meta tag.
`resultEncoding`	The encoding that the HTML results are in. This may be any integer constant defined in `<TextCommon.h>`.
`resultTranslationEncoding`	The encoding that the HTML results should be translated to. This may be any integer constant defined in `<TextCommon.h>`.
`resultTranslationFont`	the preferred font for the translated text

Additional INTERPRET tag attributes for Sherlock 2

`priceStart`	Specifies a text pattern marking the beginning of the price information provided for each item in the list of results. When present, the first numeric text found after the pattern will be interpreted as the price of the item. This attribute is only supported when the plug-in is in a "Shopping" channel.
`priceEnd`	Specifies a text pattern marking the end of the price information. The search for price information will not proceed beyond this text pattern. The text patterns defined in the priceStart and priceEnd attributes are used to extract the price for each individual search result.
`availStart`	Specifies a text pattern marking the beginning of the availability information provided for each item in the list of results. When present, the text found after the pattern will be interpreted as the price of the item. This attribute is only supported when the plug-in is in a "Shopping" channel.
`availEnd`	Specifies a text pattern marking the end of the availability information. The search for availability information will not proceed beyond this text pattern. The text patterns defined in the availStart and availEnd attributes are used to extract the availability for each individual search result.
`dateStart`	Specifies a text pattern marking the beginning of the date information provided for each item in the list of results. When

	present, the text found after the pattern will be interpreted as the date of the item. This attribute is only supported when the plug-in is in a "News" channel.
`dateEnd`	Specifies a text pattern marking the end of the date information. The search for date information will not proceed beyond this text pattern. The text patterns defined in the dateStart and dateEnd attributes are used to extract the date for each individual search result.
`nameStart`	Specifies a text pattern marking the beginning of the name information provided for each item in the list of results. When present, the text found after the pattern will be interpreted as the name of the item. By default, the information of that appears in the Name column for each result item is the first html anchor that appears in the text between the delimited by the resultItemStart and resultItemEnd attributes. To have the information from another anchor appear in the name column, use the nameStart and nameEnd attributes.
`nameEnd`	Specifies a text pattern marking the end of the name information. The search for date information will not proceed beyond this text pattern. The text patterns defined in the nameStart and nameEnd attributes are used to extract the name for each individual search result.
`langauge`	ISO 639 language code of the result page.*
`country`	ISO 3166 country code of the result page.*

*An internet search source plug-in can specify `language` and `country` codes in the interpret portion of the a search source. This information helps Sherlock 2 determine region information. For Sherlock 2 this information is only used to help determine the price column of a Shopping Channels search results. In Sherlock 1 this information is ignored.

The attributes `charset`, `resultEncoding`, `resultTranslationEncoding`, and `resultTranslationFont` are for interpreting results returned with different character encodings. If the result page contains the HTML meta tag "charset," then the Sherlock application will use the Text Encoding Converter to translate the document into a Macintosh encoding.

It is possible, though, that the Sherlock application will not be able to recognize a text encoding by name. For these cases, search plug-in creators can explicitly specify the character encoding that will be used in responses to queries by using the `resultEncoding` attribute. The value specified for the `resultEncoding` attribute can be any integer constant defined in the file `<TextCommon.h>`. Similarly, `resultTranslationEncoding` is used to specify the text encoding that the document should be translated into before processing continues. The value used for this attribute is also an integer constant from `<TextCommon.h>`.

For example, if a result page returned from a search site was encoded using the "euc-jp" character set (in `<TextCommon.h>` "euc-jp" is defined as `kTextEncodingEUC_JP` = 2336) and we would prefer that it be translated to Mac Japanese (defined as `kTextEncodingMacJapanese` = 1 in `<TextCommon.h>`) and displayed using the "Osaka" font, then the following character translation values would be specified:

```
<interpret
resultEncoding = 2336
resultTranslationEncoding = 1
resultTranslationFont = "Osaka">
```

`INTERPRET` tags are optional, and all of the attributes within an `INTERPRET` tag are optional as well. If a `SEARCH` block does not contain an `INTERPRET` tag, then every link found in the result page will be treated as a result and the Sherlock application will present the entire list to the user as the results of her query

With Sherlock 2, a plug-in can support multiple `INTERPRET` tags in a search tag. Multiple `INTERPRET` tags can be used when a given site can return result pages in a number of different formats or results may be returned on the same page in a number of different sections. With older versions of Sherlock, only the first interpret tag will be used.

Back to top

An Example

In this example, it is assumed that the Internet search site that we are writing the search plug-in file for is located at the URL <http://clarus.apple.com>. (As of this writing, this site does not exist, although the following text is written as if the site does exist. If the site did exist, it would presumably enable visitors to search for information regarding Clarus the Dogcow. An explanation of how visitors other than dogcattle would make use of the search results is beyond the scope of this document and is left as an exercise for the reader.)

Step 1: Describe the site in the opening `SEARCH` tag.

Using your web browser, go to the search site and view the HTML source for the web page. Somewhere in the HTML, you should find a `FORM` tag as follows:

```
<form action="http://clarus.apple.com/Titles" method="get"
name="Search">
```

Or, it is possible that the action may be specified as a local string as follows:

```
<form action="/Titles" method="get" name="Search">
```

If the action is specified as a local string, then prefix it with the address in the `SEARCH` tag's action attribute. Using the information found here, we can construct the opening `SEARCH` tag for the search block:

```
<search
    name="Clarus"
    description = "The Clarus Search Site"
    action="http://clarus.apple.com/Titles/"
    method=get>
```

From the HTML source, we were able to determine that the action is `http://clarus.apple.com/Titles/` and the method appropriate for communicating with the site is `get`. The name of the site and the description are values we set ourselves.

Step 2: Define the INPUT tags.

There are two ways to determine what inputs are expected by an Internet search site. The first method is to manually perform a query and look at the URL that is sent to the server. The second is to pick through the HTML to discover the information.

The Query Method. Looking at the query information is the simplest method. For example, if we go to the search site in our web browser and type the query string "coffee" and start a search, then we may observe a URL that looks like this:

```
http://clarus.apple.com/Titles?qt=coffee&nh=10
```

From which, we can locate the inputs. The inputs come after the "?" and are separated by ampersand characters [&]. In this query, the inputs are as follows:

```
qt=coffee
nh=10
```

Using this information, we can construct the following two INPUT tags:

```
<input name="qt" user>
<input name="nh" value="10">
```

There may be some optional parameters available on a search site, so trying different options and queries may yield more useful information.

The HTML Method. If the inputs are not present in the URL then they must be determined by looking at the HTML source. Here, we look for the `INPUT` tags present in the search site's web page to determine what will be used to describe the inputs. For example, suppose the first few lines of the HTML for a search site were formatted as follows:

```
<form action="/Titles" method="get" name="Search">
<table width="100%" cellspacing=0 cellpadding=3 border=0>
<tr><td colspan=4>
Search</td>
<td align=center><a
href="/Help?pg=Help.HTML"><b>Tips</b></a>
</td></tr>
<tr><td colspan=5>
<input type="text" name="qt" value="" size="25" MAXLENGTH=255>
</td></tr>
<INPUT TYPE=hidden NAME="nh" VALUE="10">
</table>
</form>
```

Between the `<form>` and `</form>` tags, there are the two inputs relevant to accessing this search engine:

```
<input type="text" name="qt" value="" size="25"
MAXLENGTH=255>
<INPUT TYPE=hidden NAME="nh" VALUE="10">
```

Again, this information can be used to construct the following two `INPUT` tags:

```
<input name="qt" user>
<input name="nh" value="10">
```

Experimenting with these input parameters and writing different types of query URLs can provide useful information about their meaning and use. For instance, after writing several variations of the query URL, we discovered that `nh` specifies the number of hits that should be returned in a response to a query. Rather than 10 hits at a time, we would prefer to see 25 hits, so we change the inputs as follows:

```
<input name="qt" user>
<input name="nh" value="25">
```

Now that the inputs have been determined, there is enough information to put together a complete search plug-in file:

```
<search
    name="Clarus Test"
    description = "The Clarus Search Site"
    action="http://clarus.apple.com/Titles/"
    method=get>
<input name="qt" user>
<input name="nh" value="25">
</search>
```

However, in this form, although it will be possible for queries to be sent and results to be displayed, the lack of an `INTERPRET` tag means that the results may not be displayed correctly. To ensure that they are, an `INTERPRET` tag should be added.

Step 3: Describe the results in the `INTERPRET` tag.

Determining the text delimiters located in the responses returned by Internet search engines requires examination of the HTML source returned as the response to one or more queries. From this data, we can determine text patterns delimiting interesting parts of the response information. For example, suppose the following were returned as a response to a query:

```
<HTML>
<HEAD><TITLE>Sample Results</TITLE></HEAD>
<BODY>

<A HREF="http://www.apple.com">
<IMG SRC="http://www.apple.com/main/elements/apple.gif"
 ALT="Apple Computer"
</A>

<P>
<SMALL>90%</SMALL>
<A HREF="http://www.apple.com/hotnews/">Hot News</A>&nbsp;
Apple Hot News - http://www.apple.com/hotnews
<BR><A HREF="http://www.apple.com">Apple Computer</A>
</P>
<P>
<SMALL>85%</SMALL>
<A HREF="http://www.apple.com/products/">Apple Products</A>
&nbsp;
Apple - Products - http://www.apple.com/products
<BR><A HREF="http://www.apple.com">Apple Computer</A>
</P>
</BODY>
</HTML>
```

Listing 2. A sample HTML response to a query.

The List of results are delimited by the text patterns "</A>" and "</BODY>":

```
resultListStart="</A>"
resultListEnd="</BODY>"
```

Each item in the list of results is bracketed by the text patterns "<P>" and "</P>":

```
resultItemStart="<P>"
resultItemEnd="</P>"
```

And, the relevance score for each item is bracketed by the text patterns "<SMALL>" and "</SMALL>":

```
relevanceStart="<SMALL>"
relevanceEnd="</SMALL>"
```

Putting this all together, the complete search plug-in file would have the following contents:

```
<search
    name="Clarus Test"
    description = "The Clarus Search Site"
    action="http://clarus.apple.com/Titles/"
    method=get>
<input name="qt" user>
<input name="nh" value="25">
<interpret
    resultListStart="</A>"
    resultListEnd="</BODY>"
    resultItemStart="<P>"
    resultItemEnd="</P>"
    relevanceStart="<SMALL>"
    relevanceEnd="</SMALL>">
</search>
```

Back to top

Internet Search and XML Search Results

It is possible that a search engine may provide a separate machine-readable interface such as Extensible Markup Language (XML).

```
<searchResponse>
 <advertisement>
  <a href="http://www.advertiser.com">
  <img src="ad.gif">
  </a>
 </advertisement>

 <searchResults>
  <resultItem>
   <b><relevance>67%</relevance></b>
   <link><a
href="http://www.foo.com">Title</a></link><br/>
   <summary>Summary</summary>
  </resultItem>
 </searchResults>
</searchResponse>
```

Listing 3. A sample XML document.

At the time of this document's creation, the XML specification is still under development; however, using the current state of the standard, the Internet Search Interface can be easily configured to interpret XML result lists. For example, the `INTERPRET` tag shown below illustrates how a search plug-in could be set up to interpret the XML document shown in Listing 3.

```
<interpret
   resultListStart = "<searchResults>"
   resultListEnd = "</searchResults>"
   resultItemStart = "<resultItem>"
   resultItemEnd = "</resultItem>"
   relevanceStart = "<relevance>"
   relevanceEnd = "</relevance>">
```

Back to top

Tips for Search Site Administrators

Comment-style Delimiters

The Sherlock application uses information provided by search plug-in files to extract information from HTML results returned from Internet search sites. Specifically, information in search plug-in files is used to find delimiters in the response information for the search results. The question of the Sherlock application being able to find and display results consistently depends entirely on the search site remaining in sync with the formats specified in the search plug-in file. When the formats specified in the search plug-in file are based on anecdotal properties found in one or two search results files as in the example above, this sort of desynchronization can occur quite easily whenever small formatting changes are made in the result pages generated by a search site.

To avoid this problem, it is suggested that search site administrators include comments delimiting the interesting parts of response pages. By doing so, search plug-in files can be built to use the comment text as delimiters, and HTML formatting information included in result pages can be modified without risk of invalidating search plug-in files that have been built to access the search site. For example, the INTERPRET tags given below could be used to interpret the HTML response information shown in Listing 4.

```
resultListStart="<!-- RESULT LIST START -->"
resultListEnd="<!-- RESULT LIST END -->"
resultItemStart="<!-- RESULT ITEM START -->"
resultItemEnd="<!-- RESULT ITEM END -->"
relevanceStart="<!-- RELEVANCE START -->"
relevanceEnd="<!-- RELEVANCE END -->"
```

Using these text delimiters, the search provider can freely add additional formatting information to their response pages without being concerned about invalidating any search plug-in files currently in use. This approach is strongly recommended for all search site providers creating search plug-in files.

```
<HTML>
<HEAD><TITLE>Sample Results</TITLE></HEAD>
<BODY>

<!-- RESULT LIST START -->

<!-- RESULT ITEM START -->
<P>
<SMALL>
<!-- RELEVANCE START -->
90%
<!-- RELEVANCE END -->
</SMALL>
<A HREF="http://www.apple.com/hotnews/">Hot News</A>&nbsp;
Apple Hot News - http://www.apple.com/hotnews
<BR><A HREF="http://www.apple.com">Apple Computer</A>
</P>
<!-- RESULT ITEM END -->

<!-- RESULT ITEM START -->
<P>
<SMALL>
<!-- RELEVANCE START -->
85%
<!-- RELEVANCE END -->
</SMALL>
<A HREF="http://www.apple.com/products/">Apple Products</A>
&nbsp;
Apple - Products - http://www.apple.com/products
<BR><A HREF="http://www.apple.com">Apple Computer</A>
</P>
<!-- RESULT ITEM END -->

<!-- RESULT LIST END -->

</BODY>
</HTML>
```

Listing 4. A simple HTML response to a query that includes delimiting comments.

Result Lists

When interpreting search results, the Sherlock application identifies results by looking for HTML anchors containing hypertext jump attributes. At least one anchor including an hypertext jump (HREF attribute) should occur between the text patterns specified in `resultItemStart` and `resultItemEnd` or `resultItemStart`. The Sherlock application will attempt to interpret HTML results between these text patterns and expects to find at least one such anchor.

Back to top

Internet Search Language IBNF

IBNF (Italicized BNF) for Internet search plug-ins. *Italics* have been used to represent non-terminal nodes rather than the < > notation as the language being described uses those characters frequently. All letters and strings are case-insensitive and white space is ignored. Lines beginning with a # character are ignored and are treated as comments.

```
search-plugin ::= internet-search-plugin
search-plugin ::= LDAPPlugin

internet-search-plugin ::= search-header search-element-list
search-footer

search-header ::= <search search-param-list >
search-param-list ::= search-param-list search-param
search-param-list ::= search-param

search-param ::= name = string
search-param ::= method = string
search-param ::= action = string
search-param ::= update = string
search-param ::= updateCheckDays = string
search-param ::= description = string

#routeType is new for Sherlock 2
search-param ::= routeType = channel
channel ::= string
channel ::= internet
channel ::= apple
channel ::= reference
channel ::= news
channel ::= shopping

#note: multiple interpret tags are new for Sherlock 2
search-element-list ::= search-element-list search-element
search-element-list ::= search-element
search-element::= <input input-item-list >
search-element::= <interpret match-item-list >

input-item-list ::= input-item-list input-item
input-item-list ::= input-item
input-item ::= value = string
input-item ::= name = string
input-item ::= user

# The following LDAP Input Tag Extensions defined
for use
# in LDAP plug-ins can also be used in Sherlock 2 Internet
# search plugins:
input-item ::= user1
input-item ::= user2
input-item ::= user3
input-item ::= userN
input-item ::= prefix = string
input-item ::= suffix = string

match-item-list ::= match-item-list match-item
match-item-list ::= match-item

# Match item types for Sherlock 1:
match-item::= resultListStart = string
match-item::= resultListEnd = string
match-item::= resultItemStart = string
match-item::= resultItemEnd = string
```

```
match-item::= relevanceStart = string
match-item::= relevanceEnd = string
match-item::= skipLocal

# New match item types for Sherlock 2:
match-item::= priceStart = string
match-item::= priceEnd = string
match-item::= availStart = string
match-item::= availEnd = string
match-item::= dateStart = string
match-item::= dateEnd = string
match-item::= nameStart = string
match-item::= nameEnd = string

search-footer ::=  </search>

string ::= " letterlist "
string ::= ' letterlist '
string ::= nospaceletterlist
letterlist := letterlist letter
letterlist := letter
letterlist :=
nospaceletterlist := nospaceletterlist printingletter
nospaceletterlist := printingletter
```

Back to top

Making Plug-ins for the People Channel

Sherlock 2 includes a new 'People' channel that allows users to conduct searches based on personal names. When this channel is selected, Sherlock 2 uses LDAP (Lightweight Directory Access Protocol) to communicate with the server specified by the search plug-in. Developers interested in creating plug-ins that can be used for searches in the people channel can do so by using the extended LDAP search tags available for use with Sherlock 2. These new search tags are described in this section.

LDAP plug-ins are very similar in format to search plug-ins used in the other Sherlock 2 channels. The difference is they are being used to parse data received from a LDAP server instead of data received from an HTTP server. To compensate for this difference, these plug-ins use an extended version of the Internet Search Interface Language.

A detailed description of the format of LDAP Plug-in language is provided in IBNF later in this section. Readers may want to refer to this during the following discussion.

Back to top

The LDAP Search Header

The search header for LDAP Plug-in files always specifies an action that contains a LDAP URL, a method type of "ldap", and a route type of "people". For example, the following is a valid header for LDAP Plug-in:

```
    <search
        name="ldap.example.com"
        action="ldap://ldap.example.com/??one?"
        method="ldap"
        routeType="people">
```

Back to top

LDAP Input Tag Extensions

Input tags have been extended to allow selection of specific extraction of particular words from the string typed by the user. Input tags inside of Internet search plug-ins can use the 'user' flag to indicate that the input tag should use the data typed in the text entry field by the user. In LDAP Plug-ins the following flags are available for use in input tags:

- `user` - the entire text of the query.
- `user1` - the first word of the query.
- `user2` - the second word of the query.
- `user3` - the third word of the query.
- `userN` - the last word of the query.

In addition to these tags, it is possible to specify prefix and suffix strings that will be appended to inputs once they have been extracted from the string typed by the user and before they are sent to the LDAP server. To specify these strings add one of the following specifications to the input tag:

- `prefix = string` - text to prepend to the beginning of query item.
- `suffix = string` - text to append to the end of query item.

Back to top

LDAP Interpret Tag Extensions

Search results are not returned from a LDAP server in HTML format. Because of this, LDAP plug-ins use `ldapinterpret` tags instead of `interpret` tags for parsing returned data. These tags allow you to describe different kinds of information that will be displayed for the tag. The layout or a ldapinterpret tag is as follows:

```
<ldapInterpret
    name=<ldapAttribute>
    [prefix=<prefixString>]
    [suffix=<suffixString>]
    [prop=<propertyString>]
    [type=<typeString>]
    />
```

- `name` - `<ldapAttribute>` is the LDAP attribute to be displayed.
- `prefix` - optional `<prefixString>` is a prefix string that will be displayed in before the attribute if the attribute was found in the results.
- `suffix` - optional `<suffixString>` is a suffix string that will be displayed in after the attribute if the attribute was found in the results.

- `prop` - The column that this attribute relates to in Sherlock's result list. Following properties are supported with Sherlock 2: "name" (the Name column), "email" (the EMail column), and "telephonenumber" (the Telephone column).
- `type` - Type of result for Sherlock 2, the only type of result currently allowed is "mailto". This tells Sherlock to build a "mailto" URL that will be displayed in the description for the result.

Back to top

IBNF for LDAP Plug-ins

IBNF (Italicized BNF) for LAPD search plug-ins. *Italics* have been used to represent non-terminal nodes rather than the < > notation as the language being described uses those characters frequently. All letters and strings are case-insensitive and white space is ignored. Lines beginning with a # character are ignored and are treated as comments.

```
LDAPPlugin ::= LDAPHeader LDAPDefinitionList LDAPFooter

LDAPHeader ::= <search LDAPHeaderItemList >
LDAPHeaderItemList ::= LDAPHeaderItemList LDAPHeaderItem
LDAPHeaderItemList ::= LDAPHeaderItem
LDAPHeaderItem ::= name = string
LDAPHeaderItem ::= action = "LDAP_URL"
LDAPHeaderItem ::= method = "ldap"
LDAPHeaderItem ::= routeType = "people"

LDAP_URL ::= defined in rfc2255, "The
LDAP URL Format"

LDAPDefinitionList ::= LDAPDefinitionList LDAPDefinition
LDAPDefinitionList ::= LDAPDefinition
LDAPDefinition ::= <input name = string LDAPInputParamList
/>
LDAPDefinition ::= <ldapInterpret name = string
LDAPInterpretList />

LDAPInputParamList ::= LDAPInputParamList LDAPInputParam
LDAPInputParamList ::= LDAPInputParam
LDAPInputParam ::= LDAPQueryPart
LDAPInputParam ::= LDAPExtension
LDAPQueryPart ::= user
LDAPQueryPart ::= user1
LDAPQueryPart ::= user2
LDAPQueryPart ::= user3
LDAPQueryPart ::= userN
LDAPExtension ::= prefix = string
LDAPExtension ::= suffix = string

LDAPInterpretList ::= LDAPInterpretList LDAPInterpretItem
LDAPInterpretList ::= LDAPInterpretItem
LDAPInterpretItem ::= LDAPMatchPattern
LDAPInterpretItem ::= prop = LDAPProperty
LDAPInterpretItem ::= type = LDAPType

LDAPMatchPattern ::= prefix = string
LDAPMatchPattern ::= suffix = string
```

```
LDAPProperty ::= "name"
LDAPProperty ::= "email"
LDAPProperty ::= "telephonenumber"

LDAPType ::= "mailto"

LDAPFooter ::= </search>

string ::= " letterlist "
string ::= ' letterlist '
string ::= nospaceletterlist
letterlist := letterlist letter
letterlist := letter
letterlist :=
nospaceletterlist := nospaceletterlist printingletter
nospaceletterlist := printingletter
```

Back to top

AppleScript Support

The new search facilities provided by the Sherlock application can be accessed from AppleScript scripts. AppleScript scripts can ask the Sherlock application to perform an Internet search using one or more Internet Search Sites or search for files with specific content on local or remote volumes. Each of these commands returns the results of the search as a string that can be used elsewhere in your script. Optionally, AppleScript scripts can ask the Sherlock application to display the results of the search.

Searching the Internet

Internet based searches use the "search Internet" command. The "search Internet" command allows AppleScript scripts to specify the Internet search sites that will be used in the search along with query information. The query information can be provided as either a string or as a reference to a file containing the query information (but not both). Results of the search are returned as a string, and it is possible to specify that the Sherlock application display the results. Definition 1 includes the "search Internet" entry from the Sherlock application's AppleScript dictionary.

search Internet: Search the Internet

search Internet string—the Internet sites to search, optional

[**in channel** string] —the channel to search*
[**for** string]—the text to look for...
[**using** alias]—...or a saved Find file containing the query
[**display** boolean]—Specifies whether or not to display the result (default is without display)

Result: string—the URLs that match the query

*The new **in channel** parameter is only available in Sherlock 2.

Definition 1. The "search Internet" dictionary entry from the Sherlock application.

It is important to remember that the "for" and "using" parameters are mutually exclusive and cannot be used together in one command. Either the query information is provided as a string or it is provided in a file. If the display parameter is true, then the Sherlock application will display the results of the search.

The "using" parameter allows query information stored in a file to be used rather than a query string. To create such a file, use the "Save Search Criteria" command in the Sherlock application's File menu.

The direct object to this command is a list of Internet search site names. If the list of Internet search site names is not specified and the "for string" parameter is used, then the same sites that were used in the last Internet search will be used in the search. The list of Internet sites is ignored when the "using alias" parameter is specified.

Selecting Search Sites

Sherlock provides a AppleScript command allowing you to select the search sites that will be used in the next Internet search. With Sherlock 2, an additional parameter has been added to the select search sites command allowing you to select a set of search sites that will be used within a particular channel.

select search sites: Select the specified Internet search sites
    select search sites names...—a list of strings
        [**in channel** string]—in the specified channel*

    *The new **in channel** parameter is only available in Sherlock 2.

Definition 2. The "search" dictionary entry from the Sherlock application.

Searching Files

Two AppleScript commands are provided for access to the Find by Content facilities in the Sherlock application. The first command allows AppleScript scripts to perform searches based on contents of files and the second allows AppleScript scripts to create or update index files on particular volumes that are used by Find By Content. The AppleScript dictionary entry for the "search" command is shown in Definition 2 and the "index volumes" command is shown in Definition 3. The "search" command allows AppleScript scripts to perform searches based on file contents.

search: Search disks or servers
    search alias—the volumes or folders to search, optional
        [**for** string]—the text to look for...
        [**similar to** alias]—...or file(s) containing text for Find by Content...
        [**using** alias]—...or a saved Find file containing the query
        [**display** boolean]—(default is without display) Specifies whether or not to display the result

    Result: alias—the files that match the query

Definition 3. The "search" dictionary entry from the Sherlock application.

In the "search" command, the parameters "for," "similar to," and "using" are mutually exclusive parameters and may not be used together in the same command.

As in the Internet search command, the "using" parameter allows query information stored in a file to be used rather than a query string. To create such a file, use the "Save Search Criteria" command in the Sherlock application's File menu.

The direct object to the "search" command is a list of volumes or folders to search. If no list of volumes is provided and either the "search for" or the "search similar to" parameter is used, then the "search" command will search all local, indexed volumes. When the "using" parameter is specified, the list of volumes is ignored.

Indexing Volumes

Before the Find By Content facilities can be used to search a volume, the volume must contain an index. Index files are stored in an invisible folder called "TheFindByContentFolder" located in a volume's root directory and they contain necessary information for performing content-based searches. A volume cannot be searched by the Find By Content facilities unless it contains an index. AppleScript scripts can ask the Sherlock application to either update or create an index file for one or more volumes.

index volumes: Create or update the index(es) of the specified volume(s)
    index volumes alias—list of volumes to index

Definition 4. The "index volumes" dictionary entry from the Sherlock application.

Indexing Containers

Sherlock 2 adds a new AppleScript feature allowing callers to re-index particular folders or files without having to index an entire volume. This feature is not available with the original version of Sherlock. Scripts attempting to use this feature with older versions of Sherlock will fail.

index containers: Create or update the index(es) of the specified volume(s)/folder(s)/file(s)
    index containers alias—list of volume(s)/folder(s)/file(s) to index

Definition 5. The "index containers" dictionary entry from the Sherlock 2 application.

Search Channels

Sherlock 2 adds the concept of search channels. To allow script writers full access to this new facility, a new "channel" class has been added to Sherlock's AppleScript suite. Scripts can use this new class to find out what channels are available, get and set the current channel, and refer to channels in search commands. Here are some examples of commands that can be used with channels:

```
count channels
exists channel "Internet"
get channels
get name of channels
get all search sites of channel "Internet"

get current channel
set current channel to channel "Internet"
```

Back to top

The Optional kAEOpenDocuments Apple Event Parameter

To provide applications with information useful in selecting and displaying parts of documents in which users are most likely interested, when the user opens a file that was located by way of a content-based search from within one of the Sherlock application's windows, the Sherlock application will insert information about the search that led to the file into the `kAEOpenDocuments` (`'odoc'`) Apple event that is used to open the file. The Sherlock application opens files by sending `kAEOpenDocuments` Apple events to the Finder. The Finder, when receiving the `kAEOpenDocuments` Apple event, launches the application owning the document and passes the event to the application.

This type of `kAEOpenDocuments` Apple event contains an additional `keyAEPropData` (defined in `AERegistry.h`) parameter. Among the properties in the `keyAEPropData` parameter there is one identified using the keyword `'srwd'` that contains the original query string used to locate the file. The `'srwd'` property's data is formatted as a C-style string.

```
OSErr GetSearchWordsFromAppleEvent(AppleEvent* inAppleEvent,
                                      char* theText, long *ioLength)
{
    OSErr err;
    DescType outType;
    AERecord propData = {typeNull, NULL};

        /* set up our variables */
    if (ioLength == NULL || theText == NULL) return paramErr;

        /* get the property data from the Apple event */
    err = AEGetParamDesc(inAppleEvent,
                    keyAEPropData, typeAERecord, &propData);

        /* extract the search words information */
    if (err == noErr)
        err = AEGetKeyPtr(&propData, 'srwd', typeChar,
                    &outType, theText, *ioLength, ioLength);

        /* clean up and return */
    AEDisposeDesc(&propData);
    return err;
}
```

Listing 5. Retrieving the search words from and `'odoc'` Apple event.

The Example shown in Listing 5 illustrates how an application may extract the query information from an `kAEOpenDocuments` Apple event. Here, the routine attempts to retrieve the `keyAEPropData` parameter and then it attempts to extract the `'srwd'` information from the property data. If no problems occur and the `'srwd'` data is present, then the original query text will be returned in the buffer pointed to by theText, whose length must be passed in ioLength. On return, `*ioLength` will be set to the length of the string, and the function will return `noErr`.

Note: It is possible for `GetSearchWordsFromAppleEvent` to return `noErr`, but to have also returned only a portion of the query text. You should compare the size returned in `ioLength` to the original value passed in. If the value returned is larger than the original value, you should resize the buffer to the size returned, and then call `GetSearchWordsFromAppleEvent` again.

The presence of this additional parameter will not affect the behavior of existing applications built according to the guidelines set forth in the "Responding to Apple Events" chapter of *Inside Macintosh*: Interapplication Communication. However, developers may choose to take advantage of this new information when it is present in an Apple event as a clue pointing to the part of the document that the user would like to see first. (The presence of the `'srwd'` information in an `kAEOpenDocuments` Apple event implies that the user conducted a search by content and then selected and opened the document from within the list of files that were found in the search.) For example, an application may choose to highlight all occurrences of the words in the string, view the first occurrence of a word from the string, or open its find window with one or more of the query terms.

In some cases, however, it is possible that some or all of the words in the query string may not appear in the document being opened. In a normal search based on a query phrase, Find By Content will locate files that contain one or more of the words in the query. But, when a user selects one or more documents found in a previous search and requests "similar" documents, then it is possible that some of the documents found may not contain any of the words from the query string specified in the original search. Developers accessing the `'srwd'` property should plan for the possibility that some or all of the keys in the query string may not be present in the document being opened.

Back to top

Further References

- Technote TN1180, "Sherlock's Find By Content Library."
- Technote TN1181, "Sherlock's Find by Content Text Extractor Plug-ins."

Back to top

Downloadables

Acrobat version of this Note (98K).

Change History

- Originally written in September 1998 by John Montbriand.
- Revised in October 1999 by John Montbriand. The Find by Content library information formerly found in this note has been moved to Technote TN1180, "Sherlock's Find By Content Library."

Back to top

Acknowledgments

Thanks to David Casseres, Pete Gontier, Tim Holmes, Ingrid Kelly, Michael J. Kobb, Eric Koebler, Alice Li, Wayne Loofbourrow, and Karl Schlosser (Editor).

To contact us, please use the Contact Us page.
Updated: 01-November-1999