

SEARCH

The Search subsystem provides a framework for developing customized search applications. It supplies Freedom's user interface for searching and connects Freedom with one or more underlying search engines.

The Search subsystem integrates the following:

- **Search servers:** Search servers are dedicated machines operated as IIS web servers. Via HTTP, they accept search queries and return result sets. The Search subsystem can use multiple Search servers.
- **Search engines:** Search engines are software packages that run on Search servers. They closely cooperate with IIS, maintain document reference catalogs designed for quick searching, and match queries against their catalogs. Freedom's Search subsystem can use Microsoft's Site Server Search or Verity's K2 search engine. The Search subsystem allows only one search engine per Search server.
- **Search Application:** The Search Application supplies the Search subsystem's user interface, including user interface elements for building and modifying queries, submitting queries, and displaying result sets. The Search Application is an ASP application that runs on the Freedom server and is designed to accommodate extensive programmer customization. It consists of ASP pages, XSL, VBScript, and JavaScript.
- **Search Component:** The Search Component is a COM object that resides on the Search server and communicates with a search engine. It encapsulates multiple search engine APIs, exposing its own API.

As with metadata for all Freedom subsystems, Search subsystem metadata is stored in the Federated Component Directory (FCD). For example, individual Search Application files are registered as metadata objects, and the entire Search Application is registered as a single Freedom Application Object.

Note: Deployment of the Search subsystem is optional.

How the Search Subsystem Works

This chapter aims to reduce the time you spend learning details of the Search subsystem. The better you understand the framework provided, the easier you will find implementing customization requests.

The following list provides a general description of the events involved in a search. For a detailed description, including sequence diagrams and accompanying dialogs, refer to “Search Subsystem Sequences” (starting on page 9-23).

1. A user selects Search from Freedom’s Site Menu.
2. Freedom invokes `fs_search.asp` (residing on the Freedom server), which starts the Search Application.
3. The Search Application generates HTML used to render its user interface and sends the HTML to the user’s browser. Figure 9.1 shows an example Search Application using its default user interface.

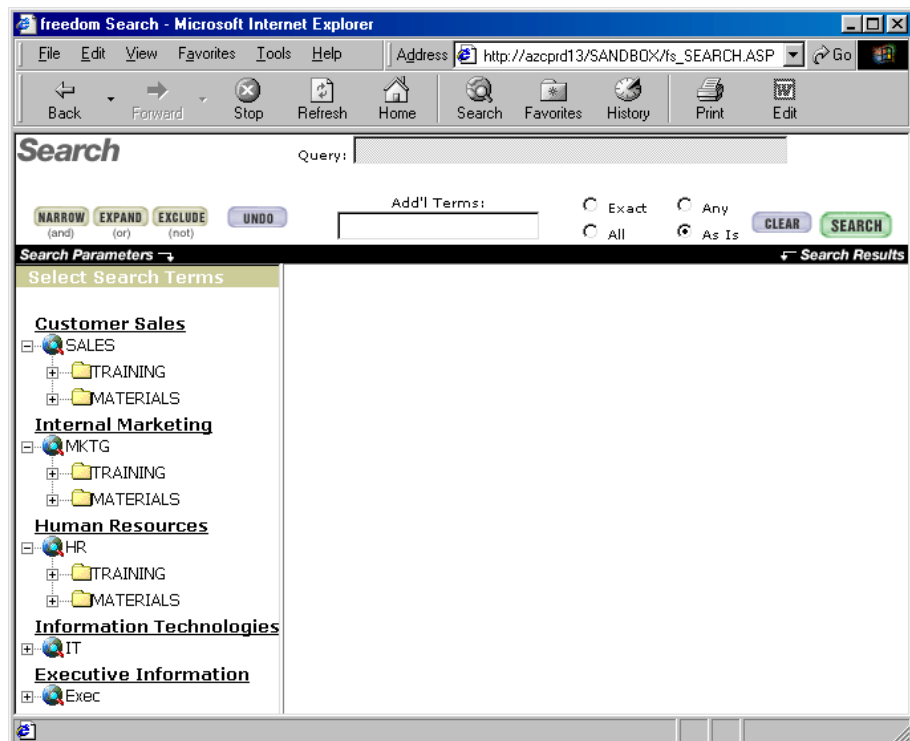


Figure 9.1 Default Search Application User Interface

3. The user creates a search query using one of the following techniques:
 - Typing a query string in the Add'l Terms field.
 - Selecting nodes and folders of a displayed Knowledge Map. A Knowledge Map is tree-like structure whose nodes and folders

“contain” text used to build search queries. As the user selects nodes and folders, the Search Application constructs the search query. In Figure 9.1, the left frame contains a Knowledge Map.

- Building the query string with a combination of manually typed text and text provided by selecting Knowledge Map nodes and folders.
4. The user clicks the Search Application’s Search button.
 5. Via an HTTP request to IIS running on a Search server, the Search Application instantiates a Search Component object on the Search server and submits the search query to it.
 6. The Search Component processes the query, and then submits the processed query to the search engine running on the Search server.
 7. Using prepared search catalogs, the search engine processes the query and delivers a result set to the Search Component.
 8. Via an ASP page on the Search server, the Search Component sends the result set to the Search Application.
 9. The Search Application creates HTML to display the result set as a list of hyperlinked document titles and sends the HTML to the user’s browser for rendering.
 10. The user clicks on a document title listed in the result set.
 11. The user’s browser issues a request for the document. The target document must reside on a repository machine satisfying the document repository requirements of the search engine. In most cases, the repository machine provides file services or runs a web server.
 12. The user’s browser opens an external window to process the repository machine’s response and display the document.

Search Application

As an ASP application running on the Freedom server, the Search Application generates HTML used to render the Search subsystem user interface and submit queries to the Search server. The Search Application then translates returned result sets from XML into HTML and sends result set HTML to the user’s browser for rendering.

You are encouraged to customize the Search Application by modifying its code or adding additional code.

Note: If you add or delete Search Application files, Freedom considers the customized Search Application a new Application Object. Using the Freedom Administration Client, you must add or delete metadata as necessary and create the new Application Object in the FCD. Refer to the *Freedom System Administrator’s Guide* for more information.

Search Application Files

In cooperation with several supporting files, four main ASP pages generate the Search Application user interface:

- **fs_search.asp**: provides three HTML frames. Each frame groups related user interface elements.
- **fs_nav.asp**: creates query modification controls and a blank field in which the user can enter search query text. HTML generated by `fs_nav.asp` is rendered in `fs_search.asp`'s top frame, referred to as the Nav frame.
- **fs_explorer.asp**: displays Knowledge Maps. HTML generated by `fs_explorer.asp` is rendered in `fs_search.asp`'s left frame, referred to as the Explorer frame.
- **fs_query.asp**: displays result sets as lists of hyperlinked document titles. HTML generated by `fs_query.asp` is rendered in `fs_search.asp`'s right frame, referred to as the Query frame.

Supporting files are described as follows:

- **fs_ssinfo.asp**: interfaces with Freedom's Digital Business Identity (DBI) subsystem to acquire user profile information required by `fs_search.asp` and `fs_nav.asp`.
- **fs_kmapinfo.asp**: interfaces with the FCD to acquire Knowledge Maps and pass them to `fs_explorer.asp`.
- **fs_resultset.xsl**: supplies XSL used by `fs_query.asp` to translate result set XML into HTML.

Customization you make to the Search Application typically affects `fs_nav.asp`, `fs_query.asp`, or the supporting files listed above. Most files that make up the Search Application are stored on the Freedom server in the following directory:

```
\\<Freedom directory path>\Apps\Search
```

`fs_query.asp` and `fs_resultset.xsl` are stored on the Search server, typically in the following directory:

```
C:\FreedomSearch
```

fs_search.asp

`fs_search.asp` does little more than create the frameset used to contain HTML generated by `fs_nav.asp`, `fs_explorer.asp`, and `fs_query.asp`. Except to change the frame layout, it is unlikely that you will need to modify `fs_search.asp`.

fs_nav.asp

`fs_nav.asp` generates user interface elements contained in the Nav frame. Nav frame user interface elements allow the user to modify search queries and execute a search. `fs_nav.asp` contains hidden form fields that participate in query submission. Corresponding hidden form fields exist in `fs_query.asp`, where they accept returned result sets.

Note: If you modify hidden form fields in `fs_nav.asp`, mirror the changes in `fs_query.asp`.

Figure 9.2 shows an example rendering of `fs_nav.asp` using its defaults.

Figure 9.2 User Interface Elements in the Nav Frame (`fs_nav.asp`)

User interface elements in the Nav frame are described as follows:

- **Query field:** contains the actual query string used in a search. As a user selects Knowledge Map nodes and folders, `fs_nav.asp` creates the query string in the Query field. This field is not editable by the user.
- **NARROW, EXPAND, and EXCLUDE buttons:** create relationships between terms in the query string using the boolean logic operators AND, OR, and NOT.
- **UNDO button:** changes the text in the Query field to the text it contained during the previous search. UNDO ignores changes made *since* the last search. By default, the UNDO button provides ten levels of undo.
- **Add'l Terms field:** provides an area in which the user can manually enter text to include additional terms in the query string. Upon submission, `fs_nav.asp` appends text from the Add'l Terms field to the query string in the Query field.
- **Exact, All, Any, and As Is radio buttons:** create boolean logic relationships between manually entered text in the Add'l Terms field and the query string built in the Query field using Knowledge Maps. The relationships can be the following:
 - All—boolean AND.
 - Exact—boolean AND, but restricts hits to those containing an exact match of the text in the Add'l Terms field.
 - Any—boolean OR.
 - As Is—No boolean operator is inserted. The relationship between the two strings is determined by the text in the Add'l Terms field. This is the default.
- **Clear button:** clears both the Query and Add'l Terms fields.
- **Search button:** submits the search query.

Table 9.1 describes all `fs_nav.asp` methods.

Method	Description
<code>clean()</code>	Clears both search string input fields, the search input box, the search query stack, and any results displayed in the Query frame.
<code>go()</code>	Submits the search request.
<code>MM_callJS(jsStr)</code>	Evaluates the JavaScript containing the <code>Undo()</code> method.
<code>nodeAction(string)</code>	Updates the Query field with appropriate boolean logic characters for Expand, Narrow, or Exclude.
<code>populate()</code>	Upon loading of the Nav frame, updates the hidden fields containing ProgID, Engine Type, Catalogs List, and Fields List.
<code>pushToStack()</code>	Stores executed queries on the query stack.
<code>showHoverText()</code>	Same as the <code>updateMouseover()</code> method, except that it displays the text when the user clicks on the Query field.
<code>undo()</code>	Removes the last executed query from the query stack and displays its previous state.
<code>updateMouseover()</code>	Displays an English version of the query when users mouseover the Query field.

Table 9.1 `fs_nav.asp` Method Descriptions

fs_explorer.asp

`fs_explorer.asp` uses a complex process to display Knowledge Maps. It is unlikely that you will need to modify `fs_explorer.asp`, but you still need to understand how `fs_explorer.asp` works.

`fs_explorer.asp` requests Knowledge Maps from `fs_kmapinfo.asp`, which then requests Knowledge Map metadata from the FCD. The FCD returns the metadata as an XML string. `fs_kmapinfo.asp` passes the XML string to `fs_explorer.asp`, which then generates HTML for displaying the Knowledge Maps using JavaScript and an Outline Control, both provided by Freedom Center's Abstract Windowing Toolkit.

Figure 9.3 shows a Knowledge Map rendered in the Explorer frame.

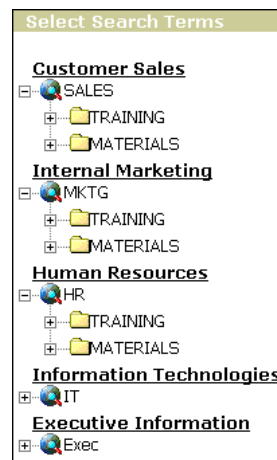


Figure 9.3 Knowledge Map in the Explorer Frame (fs_explorer.asp)

If you retain default Knowledge Map functionality, ensure that the FCD contains appropriate Knowledge Maps. By default, `fs_explorer.asp` displays all Knowledge Maps, including all nodes and folders.

Note: To restrict user access to entire Knowledge Maps or individual Knowledge Map nodes and folders, implement metadata filters in `fs_kmapinfo.asp`.

fs_query.asp

Using the XSL in `fs_resultset.xsl`, `fs_query.asp` translates the result set XML string returned by the Search Component into HTML. The HTML `fs_query.asp` generates is rendered in the Query frame as a table containing a list of hyperlinked document titles. `fs_query.asp` lists five titles per page in a four-column table and displays a navigation bar above the table. `fs_query.asp` uses multiple Microsoft XML DOM objects to translate result set XML into HTML. For details on how `fs_query.asp` works, refer to “Search Sequence” on page 9-26.

Figure 9.4 shows a displayed result set in the Query frame, using `fs_query.asp` defaults.

Documents 1 to 5 of 8 Next			
rank	score	type	title
1	77%		Restoring the Console
2	77%		Microsoft Management Console Basics
3	77%		Introduction to Microsoft Management Console
4	77%		About Microsoft Management Console
5	77%		About Microsoft Management Console

Figure 9.4 Result Set in the Query Frame (fs_query.asp)

Table columns in the result set are described as follows:

- **Rank:** an integer representing the document's position within a sorted result set.
- **Score:** a percentage indicating the relevance of a document to the query. Higher scores more closely satisfy the query. `fs_query.asp` sorts document links on this field (by default, in descending order).
- **Type:** the document type (HTML, Word, etc.), represented by an icon. Table 9.2 lists default associations between document types and icons.





Document Type	Icon	Icon Filename
Dataset		fs_dataset.bmp
HTML (default)		fs_document.bmp
Presentation		fs_presentation.bmp
Spreadsheet		fs_spreadsheet.bmp

Table 9.2 Default Associations Between Document Types and Icons

You can create new associations between document types and icons, or define criteria by which associations are made, by editing `fs_resultset.xml`. For example, you might create the following association:

PDF (Adobe Acrobat)



fs_pdf.bmp

- **Title:** The document title as a hyperlink. If a document type supports the Title tag (includes Microsoft Office document types, HTML, and others), `fs_query.asp` lists the document's title; otherwise, `fs_query.asp` lists the document's filename.

Ensure that `fs_query.asp` and any customization files it requires reside on the Search server, typically in the following directory:

C:\FreedomSearch

You can modify result set display in the following ways:

- Change text formatting by modifying `fs_resultset.xml`.
- Change the size or position of the entire Query frame by modifying `fs_search.asp`.
- Include custom columns and headings, or change the default columns, their positions, or their headings. Refer to “Search Catalog Custom Fields” on page 9-31 and “Modify the Display of Result Set HTML” on page 9-35 for more information.

You may need to modify code contained in `fs_query.asp`. However, it is more likely that you will need to modify the XSL contained in `fs_resultset.xml`. Refer to “`fs_resultset.xml`” on page 9-12 for a discussion of the `fs_resultset.xml` file. Simple example procedures for modifying the formatting of displayed result sets are provided in “Modify the Display of Result Set HTML” on page 9-35.

`fs_query.asp` contains a significant amount of code in addition to the code in the methods described here. If you extensively modify the Search Application, you may need to reorganize the code in `fs_query.asp` into additional methods.

Table 9.3 describes `fs_query.asp` methods you may need to modify.

Method	Description
<code>BrowserSupportsXML()</code>	Returns TRUE if the client browser can process XML; otherwise, returns FALSE.
<code>Dolnit()</code>	If the browser can process XML, the <code>Dolnit()</code> method applies the <code>fs_resultset.xml</code> file to the XML result set to generate HTML.

Table 9.3 fs_query.asp Method Descriptions

fs_ssinfo.asp

`fs_ssinfo.asp` acquires user profile information from the DBI subsystem to determine essential details regarding the Search server assigned to a user. `fs_ssinfo.asp`'s `loadSearchServerInfo()` method instantiates a `FreedomDBI.User` object, which then calls FCD services to retrieve Search server metadata. The DBI subsystem returns Search server metadata to `fs_ssinfo.asp` as an XML string. `fs_ssinfo.asp` converts the XML string into an XML object. Most `fs_ssinfo.asp` methods traverse the XML object and extract element values representing catalog and field information. `fs_ssinfo.asp` stores Search server information for later retrieval by `fs_search.asp` and `fs_nav.asp`, and finally, passes the original XML string to `fs_search.asp`.

Table 9.4 describes all `fs_ssinfo.asp` methods.

Method	Description
<code>loadSearchServerInfo()</code>	<ul style="list-style-type: none"> • Calls <code>Init()</code> to properly set up a <code>FreedomDBI.User</code> object. • Calls the <code>getSearchServerInfoXML()</code> method off the <code>DBI</code> object and loads Search server info as an XML object (<code>obj_Root</code>). • Traverses the <code>obj_Root</code> object and extracts base variables. • Returns <code>TRUE</code> if successful; otherwise, <code>FALSE</code>.
<code>getAllFieldAliases()</code>	Returns a comma-delimited string of all field aliases in all search catalogs associated with a specific Search server (duplicates removed).
<code>getAllFieldAttributeList(str_Attribute)</code>	Returns a comma-delimited string of all field attributes matching the passed parameter (with duplicates removed). The returned value is not specific to a single search catalog.
<code>getAllFieldCount()</code>	Returns the count of all fields listed in all search catalogs associated with a specific Search server (duplicates removed).
<code>getAllFieldDescs()</code>	Returns a comma-delimited string of all field descriptions in all search catalogs associated with a specific Search server (duplicates removed).
<code>getAllFieldIds()</code>	Returns a comma-delimited string of all field IDs in all search catalogs associated with a specific Search server (duplicates removed).
<code>getAllFieldNames()</code>	Returns a comma-delimited string of all field names in all search catalogs associated with a specific Search server (duplicates removed).
<code>getCatalogAliases()</code>	Returns a comma-delimited string of all catalog aliases.
<code>getCatalogAttributeList(str_Attribute)</code>	Searches for catalog nodes with attributes matching the parameter and returns a comma-delimited string of values for all matching attributes.
<code>getCatalogCount()</code>	Returns a single integer value representing the total number of catalog nodes.
<code>getCatalogDescs()</code>	Returns a comma-delimited string of all catalog descriptions.
<code>getCatalogIds()</code>	Returns a comma-delimited string of all catalog IDs.
<code>getCatalogNames()</code>	Returns a comma-delimited string of all catalog names.

Table 9.4 fs_ssinfo.asp Method Descriptions

Method	Description
getFieldAttributeList (str_CatalogName, str_Attribute)	Returns a comma-delimited string of values matching the field attribute within the named catalog.
getGrammarDisplay(str_GrammarClass, strFunctionName)	Returns a string containing the grammar display value for nodes matching the passed grammar class and function name.
getGrammarFunctions (str_GrammarClass)	Returns a variant array of grammar functions used by a specific Search server. Each entry in the array is itself an array of 3 elements. The array format by position is as follows: <ul style="list-style-type: none"> • 0—FunctionName • 1—Display • 2—Syntax
getGrammarSyntax(str_GrammarClass, strFunctionName)	Returns a string containing the grammar syntax value for nodes matching the passed grammar class and function name.
setAttributes()	Extracts attribute values from the Search server element and initializes public variables with the extracted values.
setCapabilities()	Extracts attribute values from the CAPABILITIES node element and initializes public variables with the extracted values.
setURL()	Extracts the node value from the URL element and initializes a public variable with the extracted value.

Table 9.4 fs_ssinfo.asp Method Descriptions

fs_kmapinfo.asp

fs_kmapinfo.asp requests Knowledge Map metadata from the FCD. The FCD returns the metadata as an XML string, which fs_kmapinfo.asp passes to fs_explorer.asp.

Note: If you process Knowledge Map metadata, process it in fs_kmapinfo.asp before it passes the XML string to fs_explorer.asp.

Table 9.5 describes all `fs_kmapinfo.asp` methods.

Method	Description
<code>getKmapAttribList(str_Attribute)</code>	Returns a variant array of strings. Each string in the array maps to a Knowledge Map XML attribute value denoted by the input parameter.
<code>getKmapNames()</code>	Returns a variant array of strings. Each string in the array maps to a Knowledge Map name referenced in the XML returned by the FCD's <code>getAllKmapNamesXML()</code> method.
<code>getKmapXML(str_Name)</code>	Calls the FCD's <code>getKmapInfoXML()</code> method and passes back the XML string it returns.

Table 9.5 `fs_kmapinfo.asp` Method Descriptions

fs_resultset.xsl

`fs_query.asp` uses `fs_resultset.xsl` to convert result set XML into HTML. The default Freedom installation places `fs_resultset.xsl` in the following directory on the Freedom server:

```
\\<Freedom directory path>\Apps\Search
```

The default Freedom installation places a copy of `fs_resultset.xsl` on the Search server, typically in the following directory:

```
C:\FreedomSearch
```

Edit the copy of `fs_resultset.xsl` on the Search server. `fs_query.asp` looks for it there. Keep the `fs_resultset.xsl` copy on the Freedom server as a backup.

The default `fs_resultset.xsl` file provides examples of how to:

- Use XSL pattern syntax.
- Translate result set XML into HTML.
- Extract data values from the XML result set using the following XSL elements:
 - `<xsl:template>`
 - `<xsl:value-of>`
 - `<xsl:for-each>`
 - `<xsl:attribute>`
 - `<xsl:choose>`
 - `<xsl:when>`
 - `<xsl:otherwise>`

Table 9.6 shows the XSL code from `fs_resultset.xsl` used to produce HTML for rendering the Query frame navigation bar. The code extracts values from the XML result set's `<RESULTS>` root element using the XSL element `<xsl:value-of>`. See “`<RESULTS>`” on page 9-18 for information on the XML `<RESULTS>` element.

```
<TABLE WIDTH="100%" STYLE="font-size:10pt;">
<TR>
<TD ALIGN="LEFT"><A ID="idPrev"
  HREF="JavaScript:PrevPage()"> PREV page </A></TD>
<TD ALIGN="CENTER">Documents <B><xsl:value-of
  select="/RESULTS/@start"/></B> to <B><xsl:value-of
  select="/RESULTS/@end"/></B> of <B><xsl:value-of
  select="/RESULTS/@found"/></B> results.</TD>
<TD ALIGN="RIGHT">
  <A ID="idNext" HREF="JavaScript:NextPage()">
    NEXT page</A>
</TD>
</TR>
</TABLE>
```

Table 9.6 Example `fs_resultset.xsl` Code

Knowledge Maps

Knowledge Maps are tree-like, content-based data structures that allow users to navigate searchable content and build search queries. Knowledge Maps organize searchable content in a way that establishes the context of a search.

A user clicks on a Knowledge Map's nodes to collapse or expand its branches, selects one or more Knowledge Map folders, and then clicks the Search button. As the user selects Knowledge Map nodes and folders, the Search Application uses Knowledge Map metadata to build the search query in the Query field. The user cannot manually edit text displayed in the Query field.

Note: If you remove Knowledge Map functionality, remove appropriate dependencies from `fs_search.asp`, `fs_explorer.asp`, `fs_kmapinfo.asp`, `fs_nav.asp`, and customization files.

Knowledge Map Creation

Freedom does not supply out-of-the-box Knowledge Maps. You are expected to create or import Knowledge Maps using the Freedom Administration Client. Refer to the *Freedom System Administrator's Guide* for instructions on how to create or import Knowledge Maps.

Knowledge Map Metadata

The FCD stores Knowledge Map metadata as Root metadata objects and Category metadata objects. Root metadata objects contain Category metadata objects, and each Category metadata object can contain query text or subsequent Category metadata objects.

Names of Knowledge Map Root metadata objects become titles at the highest level of logical document grouping. In the Knowledge Map rendered in Figure 9.1 on page 9-2, the Search Application displays Root metadata objects as “Customer Sales”, “Internal Marketing”, etc.

Names of Knowledge Map Category metadata objects become text representing the nodes and folders of the Knowledge Map. Category metadata objects are shown in Figure 9.1 on page 9-2 as “SALES”, “MKTG”, “HR”, “TRAINING”, “MATERIALS”, “IT”, and “EXEC”. If a Category metadata object contains one or more subsequent Category metadata objects whose names are not yet displayed, the Search Application displays the node with a plus sign to its left. If all nodes and folders of a containing node are displayed, the Search Application displays the node with a minus sign to its left.

The Search Application builds search queries using the string contained in the Command property of selected Root and Category metadata objects.

Table 9.7 describes all Knowledge Map Root metadata object properties.

Property	Description	Required	Data Type
Command	Search query term representing this Knowledge Map Root metadata object. This string becomes part of the search query when a user selects an interface element representing a Knowledge Map Root metadata object.	Yes	String
CreateDate	Creation date of this Knowledge Map Root metadata object. Not editable. This property is automatically generated by the FCD.	No	String
Description	Use and origin of this Knowledge Map Root metadata object.	No	String

Table 9.7 Knowledge Map Root Metadata Object Property Descriptions

Table 9.8 describes all Knowledge Map Category metadata object properties.

Property	Description	Required	Data Type
Command	Search query term representing this Knowledge Map Category metadata object. This string becomes part of the search query when a user selects an interface element representing a Knowledge Map Category metadata object.	Yes	String
CreateDate	Creation date of this Knowledge Map Category metadata object. Not editable. This property is automatically generated by the FCD.	No	String
Description	Use and origin of this Knowledge Map Category metadata object.	No	String

Table 9.8 Knowledge Map Category Metadata Object Property Descriptions

Note: The FCD does not restrict user access to Knowledge Map Root or Knowledge Map Category metadata objects. By default, the Search Application displays all Knowledge Maps in the FCD, including all nodes and folders. To restrict access based on user authorizations, implement filters in `fs_kmapinfo.asp`. Refer to “Modify Knowledge Map Metadata” on page 9-34 for a suggested procedure.

Search Component

The Search Component is a COM object implemented as an MTS server package named `FreedomSearch` (`FreedomSearch.dll`). It provides a single interface, `FreedomSearch.Search`. The Search Component links the Search Application (on the Freedom server) with a search engine (on a Search server).

Note: `FreedomSearch.dll` must reside on each Search server, typically in the following directory:

`C:\FreedomSearch`

The following summarizes the Search Component:

- Server-side COM component that interfaces with APIs of multiple search engines.
- Encapsulates proprietary search engine grammar and syntax.
- Masks search engine variations, allowing the Search Application to function independent of any specific search engine.
- Facilitates Search subsystem scaling with multiple Search servers.

The Search Component (`FreedomSearch.dll`) has no metadata associated with it. Subsequently, it is not registered in the FCD.

The Search Component contains only two methods you need to be concerned with:

- **search()**: performs initialization routines in cooperation with the underlying search engine.
- **getXMLResultSet()**: returns the next page of the XML result set. The `intPAGE_SIZE` constant in `fs_query.asp` determines page size (the number of document links listed in a result set page).

When a user initiates a query, `fs_query.asp` first calls the Search Component's `search()` method, and then calls its `getXMLResultSet()` method. The Search Component responds by returning an XML string representing the first page of the result set. `fs_query.asp` later calls the `getXMLResultSet()` method each time the user requests the next page of a result set.

For descriptions of all public methods in the Search Component, refer to the *Freedom API Reference Guide*.

Example Result Set XML

If you modify `fs_query.asp` or significantly change other parts of the default Search Application, you may need to be familiar with the XML returned by the Search Component's `getXMLResultSet()` method.

Table 9.9 shows a formatted example XML result set page.

```
<?xml version="1.0"?>
<RESULTS query="&apos;Animals&apos;" start="6" end="10" found="14" searched="555"
  pagesize="5" maxdocs="200" sortfield="score" sortorder="false" >
  <HIT>
    <RANK sortable="false" alias="rank">6</RANK>
    <SCORE sortable="true" alias="score">99</SCORE>
    <TITLE sortable="true" alias="title">All About Dogs</TITLE>
    <URL sortable="false" alias="url">http://www.animals.com/dogs.htm</URL>
    <TYPE sortable="false" alias="type">htm</TYPE>
    <SUMMARY sortable="false" alias="summary">This web page is about dogs.
  </SUMMARY>
  <FIELDS>
    <FIELD sortable="true" alias="author">Dog Expert</FIELD>
    <FIELD sortable="true" alias="noise">woof</FIELD>
    <FIELD sortable="true" alias="eats">shoes</FIELD>
  </FIELDS>
</HIT>
<HIT>
  <RANK sortable="false" alias="rank">7</RANK>
  <SCORE sortable="true" alias="score">97</SCORE>
```

```

<TITLE sortable="true" alias="title">All About Cats</TITLE>
<URL sortable="false" alias="url">http://www.animals.com/cats.xls</URL>
<TYPE sortable="false" alias="type">xls</TYPE>
<SUMMARY sortable="false" alias="summary">This spreadsheet lists various
  expenses related to cats.</SUMMARY>
<FIELDS>
  <FIELD sortable="true" alias="author">Cat Expert</FIELD>
  <FIELD sortable="true" alias="noise">meow</FIELD>
  <FIELD sortable="true" alias="eats">mice</FIELD>
</FIELDS>
</HIT>
<HIT>
  <RANK sortable="false" alias="rank">8</RANK>
  <SCORE sortable="true" alias="score">92</SCORE>
  <TITLE sortable="true" alias="title">All About Birds</TITLE>
  <URL sortable="false" alias="url">http://www.animals.com/birds.doc</URL>
  <TYPE sortable="false" alias="type">doc</TYPE>
  <SUMMARY sortable="false" alias="summary">This Word document is about birds.
</SUMMARY>
<FIELDS>
  <FIELD sortable="true" alias="author">Bird Expert</FIELD>
  <FIELD sortable="true" alias="noise">chirp</FIELD>
  <FIELD sortable="true" alias="eats">seed</FIELD>
</FIELDS>
</HIT>
<HIT>
  <RANK sortable="false" alias="rank">9</RANK>
  <SCORE sortable="true" alias="score">88</SCORE>
  <TITLE sortable="true" alias="title">All About Hamsters</TITLE>
  <URL sortable="false" alias="url">http://www.animals.com/hamsters.ppt</URL>
  <TYPE sortable="false" alias="type">ppt</TYPE>
  <SUMMARY sortable="false" alias="summary">This PowerPoint presentation is
    about hamsters.
</SUMMARY>
<FIELDS>
  <FIELD sortable="true" alias="author">Hamster Expert</FIELD>
  <FIELD sortable="true" alias="noise">squeak</FIELD>
  <FIELD sortable="true" alias="eats">carrots</FIELD>
</FIELDS>
</HIT>
<HIT>
  <RANK sortable="false" alias="rank">10</RANK>
  <SCORE sortable="true" alias="score">81</SCORE>
  <TITLE sortable="true" alias="title">All About Iguanas</TITLE>
  <URL sortable="false" alias="url">http://www.animals.com/iguanas.pdf</URL>
  <TYPE sortable="false" alias="type">pdf</TYPE>
  <SUMMARY sortable="false" alias="summary">This Adobe document is about iguanas.
</SUMMARY>
<FIELDS>
  <FIELD sortable="true" alias="author">Iguana Expert</FIELD>
  <FIELD sortable="true" alias="noise">hiss</FIELD>
  <FIELD sortable="true" alias="eats">bugs</FIELD>
</FIELDS>
</HIT>
</RESULTS>

```

Table 9.9 Example XML Result Set (one page)

Result Set XML Elements and Attributes

The following sections describe elements and attributes of the XML string returned by the Search Component's `getXMLResultSet()` method.

<RESULTS>

The <RESULTS> element is the XML root element. It represents one page of the result set. All <RESULTS> element attributes are always present within the <RESULTS> element; however, they may be empty.

Table 9.10 describes <RESULTS> element attributes.

<RESULTS> Attribute	Description
end	Ordinal number associated with the last document on a page of the result set. Consider a result set that contains 100 documents and uses a page size of 5. The end attribute for the first page is 5; the second is 10; the third is 15; etc.
found	Number of documents present in the result set. This attribute is always empty ("") if the AllowHitCount property is set to FALSE.
maxdocs	Maximum number of documents allowed in the entire result set. Default = 200.
pagesize	Maximum number of documents in a single page of the result set. Default = 5.
query	Query string passed to the search engine. This query string uses the syntax of the native search engine.
searched	Number of documents searched as a result of the query. This attribute is valid only for the Verity K2 search engine. It is always empty for the Site Server Search search engine.
sortfield	Alias name of the primary sort field.
sortorder	Sort order of the primary sort field (ascending, descending).
start	Ordinal number associated with the first document on this page of the result set. Default = 1. Consider a result set that contains 100 documents and uses a page size of five. The start attribute for the first page is 1; the second is 6; the third is 11; etc.

Table 9.10 <RESULTS> Element Attribute Descriptions

The <RESULTS> element always contains at least one <HIT> child element.

<HIT>—a document listed in the result set.

<HIT>

Each document found in a search constitutes one <HIT> element. The <HIT> element contains no attributes; however, it contains child elements. Each child element represents a column in the displayed result set table.

The <HIT> element always contains one instance of each child element described in Table 9.11.

<HIT> Child Element	Description
<RANK>	Ordinal number of the document within the result set.
<SCORE>	Relevance of the document with respect to the query.
<SUMMARY>	Short description of the document.
<TITLE>	Document title.
<TYPE>	Extension of the file containing the document.
<URL>	Document URL.

Table 9.11 <HIT> Child Element Descriptions

The <HIT> element always contains zero or one instance of the <FIELDS> child element.

<FIELDS>—Custom fields associated with the document.

<RANK>

The <RANK> element contains an integer value representing the order of a document's listing within the result set. All <RANK> element attributes are always present within the <RANK> element.

Table 9.12 describes all <RANK> element attributes.

<RANK> Attribute	Description
alias	Represents the internal alias name of the field. This attribute always contains the string "RANK".
sortable	Represents whether the field is sortable. This attribute always contains the string "FALSE".

Table 9.12 <RANK> Element Attribute Descriptions

The <RANK> element contains no child elements.

<SCORE>

The <SCORE> element contains an integer value, 0 to 100, meant to be a percentage value representing a document's relevance to the query. All <SCORE> element attributes are always present within the <SCORE> element.

Table 9.13 describes all <SCORE> element attributes.

<SCORE> Attribute	Description
alias	Represents the internal alias name of the field. This attribute always contains the string "SCORE".
sortable	Represents whether the field is sortable. This attribute always contains the string "TRUE".

Table 9.13 <SCORE> Element Attribute Descriptions

The <SCORE> element contains no child elements.

<TITLE>

The <TITLE> element contains a string value representing the document's title. All <TITLE> element attributes are always present within the <TITLE> element.

Table 9.14 describes all <TITLE> element attributes.

<TITLE> Attribute	Description
alias	Represents the internal alias name of the field. This attribute always contains the string "TITLE".
sortable	Represents whether the field is sortable. This attribute always contains the string "TRUE".

Table 9.14 <TITLE> Element Attribute Descriptions

The <TITLE> element contains no child elements.

<URL>

The <URL> element contains a string value representing the document's URL. All <URL> element attributes are always present within the <URL> element.

Table 9.15 describes all <URL> element attributes.

<URL> Attribute	Description
alias	Represents the internal alias name of the field. This attribute always contains the string "URL".
sortable	Represents whether the field is sortable. This attribute always contains the string "FALSE".

Table 9.15 <URL> Element Attribute Descriptions

The <URL> element contains no child elements.

<TYPE>

The <TYPE> element contains a string value representing the document type. The Search Component extracts the file extension from a URL or file system pointer and copies it into the <TYPE> element. All <TYPE> element attributes are always present within the <TYPE> element.

Table 9.16 describes all <TYPE> element attributes.

<TYPE> Attribute	Description
alias	Represents the internal alias name of the field. This attribute always contains the string "TYPE".
sortable	Represents whether the field is sortable. This attribute always contains the string "FALSE".

Table 9.16 <TYPE> Element Attribute Descriptions

The <TYPE> element contains no child elements.

<SUMMARY>

The <SUMMARY> element contains a string value representing a short document description. All <SUMMARY> element attributes are always present within the <SUMMARY> element.

Table 9.17 describes all <SUMMARY> element attributes.

<SUMMARY> Attribute	Description
alias	Represents the internal alias name of the field. This attribute always contains the string "SUMMARY".
sortable	Represents whether the field is sortable. This attribute always contains the string "FALSE".

Table 9.17 <SUMMARY> Element Attribute Descriptions

The <SUMMARY> element contains no child elements.

<FIELDS>

The <FIELDS> element represents a grouping of custom fields associated with a document. The <FIELDS> element contains no attributes.

The <FIELDS> element always contains zero or more instances of the <FIELD> child element.

<FIELD>—represents a custom field associated with the document.

<FIELD>

Each <FIELD> child element within a <FIELDS> element contains a string value representing the name of a custom field. All <FIELD> element attributes are always present within the <FIELD> element.

Table 9.18 describes all <FIELD> element attributes.

<FIELD> Attribute	Description
alias	Represents the internal alias name of the custom field. This attribute is never empty.
sortable	Represents whether the field is sortable. This attribute always contains the string "TRUE".

Table 9.18 <FIELD> Attribute Descriptions

The <FIELD> element contains no child elements.

Search Subsystem Sequences

The following Search subsystem sequences expose most relevant details by tracking default behavior along two paths:

- Start-up
- Search

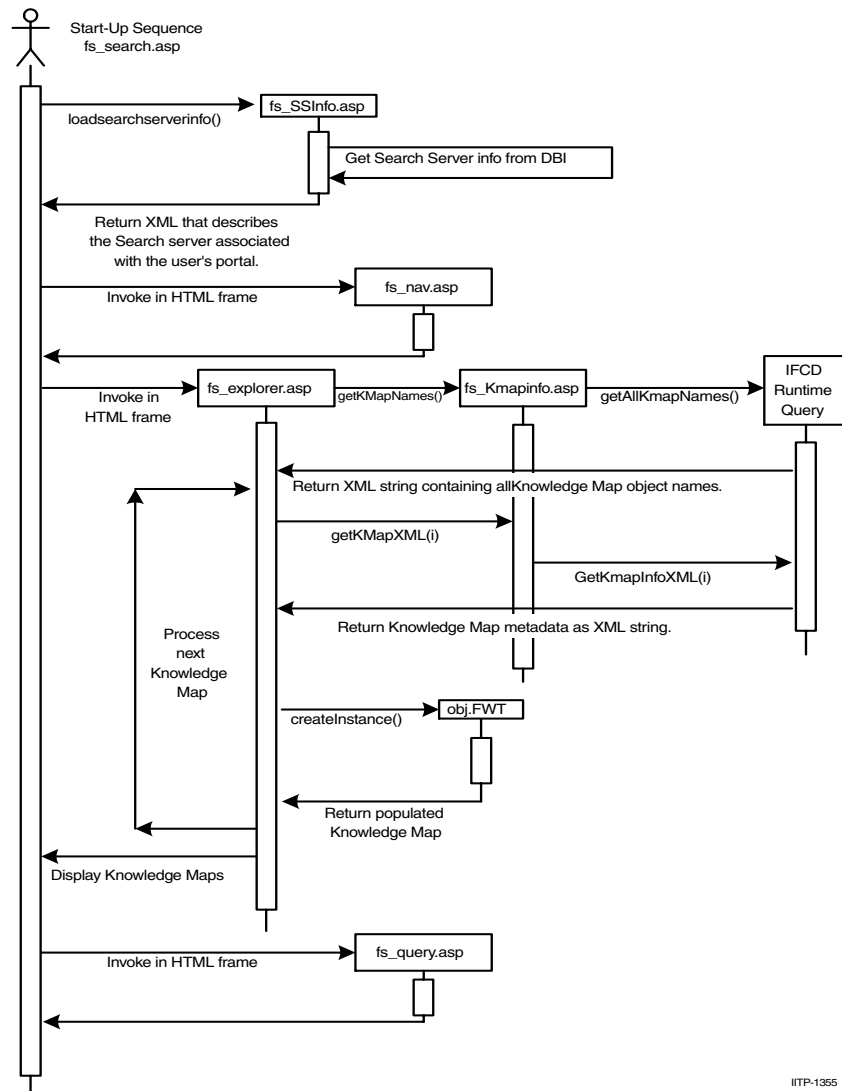
Start-up Sequence

When a user clicks the Search button in the Freedom Site Menu, the following occurs:

1. `fs_search.asp` calls `fs_ssinfo.asp`'s `loadSearchServerInfo()` method.
2. `fs_ssinfo.asp`'s `loadSearchServerInfo()` method instantiates a `FreedomDBI.User` object, initializes its variables, and calls the `getSearchServerInfoXML()` method off the object.
3. The `FreedomDBI.User` object's `getSearchServerInfoXML()` method retrieves the user's Search server information from the FCD and returns it as an XML string to `fs_ssinfo.asp`'s `loadSearchServerInfo()` method.
4. `fs_ssinfo.asp` converts the XML string to an XML object, parses the XML object, extracts metadata field values describing the Search server assigned to the user, converts the metadata field values to XML strings, stores the field values in local variables for later retrieval, and passes the original XML string to `fs_search.asp`.
5. `fs_search.asp` creates an XMLDOM object (`oXML`), loads it with metadata field values describing the Search server associated with the user, and uses the `SSurl` variable (Search server URL) to initialize the `SRC` attribute of `fs_query.asp` and point it the appropriate Search server.
6. `fs_search.asp` invokes `fs_query.asp`. The initial call to `fs_query.asp` returns an empty result set and a welcome banner for rendering in the Query frame.
7. `fs_search.asp` invokes `fs_nav.asp` in the Nav frame.
8. `fs_nav.asp` calls `fs_ssinfo.asp`'s `getGrammarSyntax()` and `getGrammarDisplay()` methods to get the appropriate syntax for each boolean operator supported by the Search server.
9. `fs_nav.asp` retrieves a list of search catalogs and values for the URL, SearchEngine Type, and ProgID, fields from `fs_ssinfo.asp`. `fs_nav.asp` stores the list and field values into their respective form elements.
10. `fs_nav.asp` creates the query modification controls (Exact, Any, All, As Is) and node action controls (Narrow, Expand, Exclude, and Undo) in the Nav frame.
11. `fs_search.asp` invokes `fs_explorer.asp` and passes it the extracted Search server URL.

12. `fs_explorer.asp` calls `fs_kmapinfo.asp`'s `getKMapNames()` method.
13. `fs_kmapinfo.asp`'s `getKMapNames()` method instantiates a `FreedomFCD.FCDRuntime` object, initializes its variables, and calls the object's `getAllKmapNamesXML()` method.
14. The `FreedomFCD.FCDRuntime` object's `getAllKmapNamesXML()` method returns an XML string containing all Knowledge Map names to `fs_kmapinfo.asp`'s `getKMapNames()` method.
15. The `fs_kmapinfo.asp`'s `getKMapNames()` method passes the XML string containing Knowledge Map names to `fs_explorer.asp`.
16. `fs_explorer.asp` calls `fs_kmapinfo.asp`'s `getKMapXML()` method, passing in as an actual parameter a string containing a single Knowledge Map name.
17. `fs_kmapinfo.asp`'s `getKMapXML()` method calls the `FreedomFCD.FCDRuntime` object's `getKmapInfoXML()` method, passing in the string containing the Knowledge Map name as an actual parameter.
18. The `FreedomFCD.FCDRuntime` object's `getKmapInfoXML()` method returns to `fs_kmapinfo.asp`'s `getKMapXML()` method an XML string containing metadata for one Knowledge Map (the current Knowledge Map).
19. `fs_kmapinfo.asp`'s `getKMapXML()` method returns to `fs_explorer.asp` the XML string containing metadata for the current Knowledge Map.
20. `fs_explorer.asp` instantiates a `Freedom Windowing Toolkit` object (`oFWT`), initializes its variables, and calls `getKMAP()` off the object while passing in as an actual parameter the XML string containing the current Knowledge Map metadata.
21. The `oFWT` object's `getKMAP()` method creates HTML to be further processed at a later step and passes that HTML to `fs_explorer.asp`.
22. `fs_explorer.asp` repeats steps 16 through 21 for each Knowledge Map name returned by its call in step 15 to `fs_kmapinfo.asp`'s `getKMapNames()` method.
23. In response to the invocation of `fs_explorer.asp` by `fs_search.asp` (step 12), `fs_explorer.asp` returns HTML representing all Knowledge Maps, along with supporting JavaScript used for rendering them in the Explorer frame. `fs_explorer.asp` obtains the JavaScript from the `oFWT` object. When the Knowledge Maps are rendered in the user's browser, the supporting JavaScript instantiates a `Freedom Center Outline Control` object on the client and uses it to structure the displayed Knowledge Maps.
24. `fs_search.asp` invokes `fs_query.asp` in the Query frame.

Figure 9.5 shows the Search subsystem's start-up sequence.



ITP-1355

Figure 9.5 Search Subsystem Start-up Sequence

In Figure 9.5, the following applies:

Preconditions:

- Successful user logon to Freedom.
- Search server, DBI subsystem, and FCD operational.
- One or more Knowledge Maps in the FCD.

Postcondition:

The Search Application displays its default user interface and populates the Explorer frame with one or more Knowledge Maps.

Search Sequence

When a user clicks on Search in the Nav frame, the following occurs:

1. `fs_search.asp` gathers from the Nav frame (`fs_nav.asp`) all search parameters that will ultimately be required by the Search Component and invokes `fs_query.asp`.
2. `fs_query.asp` persists the search parameters so that they can be referenced if the user subsequently issues additional requests to navigate the result set.
3. `fs_query.asp` instantiates a Search Component object (`oFSO`) on the Search server and sets the necessary `oFSO` properties (`QueryText`, `SortOrder`, `PageSize`, `Catalogs`, etc.).
4. `fs_query.asp` calls the `search()` method off the `oFSO` object. The `search()` method call results in the `oFSO` object performing some initialization routines in cooperation with the underlying search engine.
5. `fs_query.asp` calls the `getXMLResultSet()` method off the `oFSO` object, requesting an XML representation of a single page of the result set.
6. The `oFSO` object queries the underlying search engine and obtains a page of the native result set. `oFSO`'s first query obtains the first page of the result set. Each subsequent call obtains the next page.
7. The `oFSO` object translates the current page of the native result set into an XML string and passes the string to `fs_query.asp`.
8. `fs_query.asp` instantiates a Microsoft XML DOM object (`oXML`) to contain the XML string returned by the `oFSO` object.
9. `fs_query.asp` calls the `loadXML()` method off the `oXML` object and loads the XML string into the `oXML` object.
10. `fs_query.asp` creates another XML DOM object (`oXSL`) to contain the XSL translation file (`fs_resultset.xsl`).
11. `fs_query.asp` calls the `load()` method off the `oXSL` object and loads `fs_resultset.xsl` into the `oXSL` object.
12. `fs_query.asp` calls the `translateNodes()` method off the `oXML` object, passing in as an actual parameter the `oXSL` object. The `oXML` object uses the `oXSL` object to translate the XML string into HTML.
13. The `oXSL` object returns an HTML representation of the XML string sent to `fs_query.asp` by the Search Component (step 7).
14. `fs_query.asp` sends the HTML representation of the XML string (a single result set page) to `fs_search.asp`.
15. `fs_search.asp` sends the HTML representation of a single result set page to the user's browser for rendering in the Query frame.
16. When a user requests the next page in the result set, typically by clicking Next in the Query frame navigation bar, the process described above starts over at Step 5.

Figure 9.6 shows the Search subsystem's search sequence.

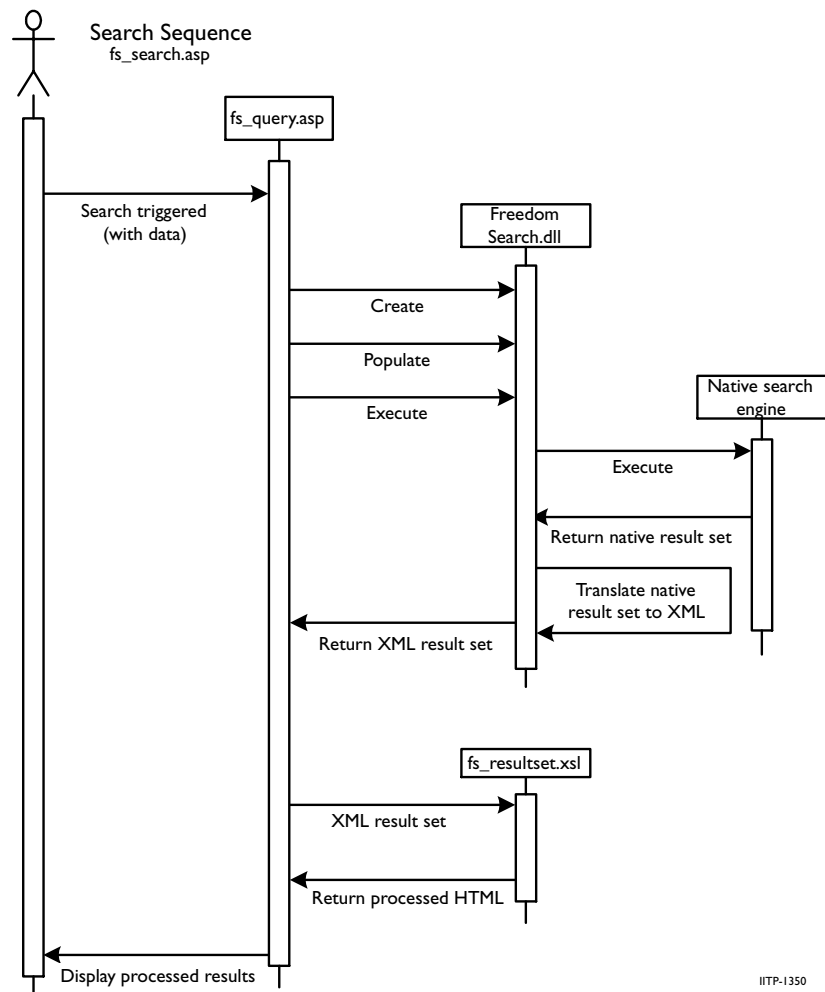


Figure 9.6 Search Subsystem Search Sequence

In Figure 9.6, the following applies:

Precondition:

- Successful display of the Search Application user interface.
- Search Component, Search sever, and native search engine operational.
- Legal search query constructed in the Nav frame.

Postcondition:

The user's browser displays a single page of the result set in the Query frame.

Search Subsystem Overall Architecture

Figure 9.7 summarizes the overall architecture of the entire Search subsystem.

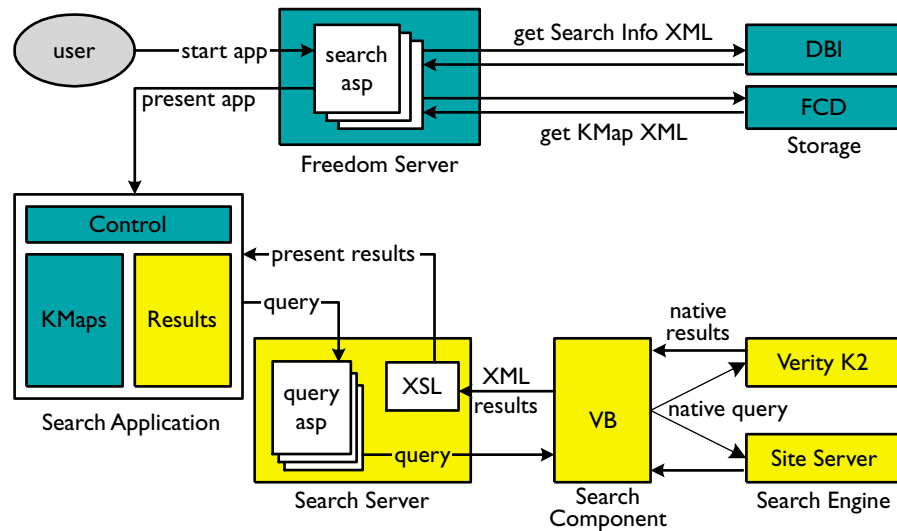


Figure 9.7 Search Subsystem Overall Architecture

Search Servers

Search servers are dedicated machines configured as IIS web servers. They run search engines. Each Search server can accommodate only one search engine, and each Freedom user can be assigned only one Search server. However, if each search engine is deployed on a distinct Search server, the Search subsystem can use multiple Search servers and multiple search engine types.

The Search subsystem uses catalogs you or an administrator build using software tools provided by the search engine. Search engine catalogs contain URLs and file pointers to repository documents and words within repository documents. Although the Search Component encapsulates search engine APIs and handles the details of acquiring a result set, you must ensure that the FCD contains metadata for each Search server and each search engine catalog.

The Search subsystem can be integrated with the following search engines: Microsoft Site Server Search, Verity K2.

Site Server Search

For detailed information on Microsoft's Site Server Search search engine, access the following URL:

<http://www.microsoft.com/siteserver/site/ProductInfo/EvalGuide.htm>

Verity K2

For detailed information on Verity's K2 search engine, access the following URL:

<http://www.verity.com/>

Search Server Metadata

Ensure that the FCD contains necessary metadata for each Search server. Table 9.19 describes all Search Server metadata object properties.

Property	Description	Required	Data Type
CreateDate	Creation date of this Search Server metadata object. Not editable. This property is automatically generated by the FCD.	No	String
Description	Use and origin of this Search Server metadata object.	No	String
GrammarXML	XML representation of the native search engine grammar.	Yes	String
IsHighlightDocs	True/False. Whether the native search engine can stream documents and highlight the search text.	For Site Server Search: No For Verity K2: Yes	Boolean
IsPostFix	True/False. If true, the grammar of the native search engine supports PostFix notation as well as InFix notation. If false, the grammar supports only InFix notation.	For Site Server Search: No For Verity K2: Yes	Boolean
IsStreamDocs	True/False. Whether the native search engine can stream documents.	For Site Server Search: No For Verity K2: Yes	Boolean
NativeSearchServerProgID	Programmatic ID associated with the COM class served up by the search engine COM DLL. This field allows administrators to resolve possible ProgID name collisions. It also allows them to select specific versions of the COM class.	Yes	String
NativeSearchURL	URL pointing to the search engine on this Search server.	Yes	String

Table 9.19 Search Server Metadata Object Property Descriptions

Search Catalogs

Search engines do not directly access repository documents. Rather, they access search catalogs containing URLs, file pointers, and pointers to words within documents.

Search engines create their catalogs using a “crawler,” which is software controlled by tools the search engine vendor provides. The crawler searches through directories and files on a network or a portion of a network. As it searches, the crawler inventories what it finds and the search engine catalogs the inventory. Later, when a search engine responds to a Search Component request, it returns a list of URLs and file pointers gathered from its catalogs.

Either you or an administrator use the tools provided by the search engine to create appropriate search catalogs. You can configure multiple Search servers running the same type of search engine to access a single set of search catalogs. For information on how to create, configure, and maintain search catalogs, refer to the search engine’s documentation.

Use the Freedom Administration Client to register search catalogs as Search Catalog metadata objects in the FCD. Refer to the *Freedom System Administrator’s Guide* for instructions. Table 9.20 describes all Search Catalog metadata object properties.

Property	Description	Required	Data Type
CatalogAlias	The search engine’s internal name for the catalog.	Yes	String
CreateDate	Creation date of this Search Catalog metadata object. Not editable. Automatically generated by the FCD.	No	String
Description	Use and origin of this Search Catalog metadata object.	No	String
NativeSearchServerType	Identifies the native search engine: <ul style="list-style-type: none"> • 1 for Site Server Search • 2 for Verity K2 Same value as the NativeSearchServerType property in the associated Search Server metadata object.	Yes	String

Table 9.20 Search Catalog Metadata Object Property Descriptions

Note: The FCD does not restrict user access to Search Catalog metadata objects.

Search Catalog Custom Fields

When setting up search engine catalogs, you have the option of including custom fields. `fs_query.asp` displays custom fields as column headings and creates the necessary columns in the displayed result set table to contain custom field values. Custom field columns appear in addition to the usual default columns (Rank, Score, Type, Title). The Search Component automatically includes custom field values in result set XML. However, to use custom fields, you must first perform somewhat extensive preparations, including the following:

- Using document management software, tag repository documents so that custom field names become custom tags. Then create values for the custom tags within each document. Refer to documentation for the document management software. Without document management software to create custom tags and values, you will probably be unable to use custom fields.
- Using search engine configuration tools, add the custom fields to the appropriate search catalogs. The search engine maps custom tag values to corresponding custom field values.
- Using the Freedom Administration Client, create Search Catalog Field metadata objects in the FCD. Use the same names as the custom field names in the search catalogs.
- Using the Freedom Administration Client, create associations between Search Catalog Field metadata objects and corresponding Search Catalog metadata objects.
- Edit `resultset.xml` so that it produces XML-to-HTML translations for the custom field values, the necessary columns, and column headings.

Table 9.20 describes all Search Catalog Field metadata object properties.

Property	Description	Required	Data Type
CreateDate	Creation date of this Search Catalog Field metadata object. Not editable. Automatically generated by the FCD.	No	String
Description	Use and origin of this Search Catalog Field metadata object.	No	String
FieldAlias	Name of the custom field. Same name used by the search engine.	Yes	String

Table 9.21 Search Catalog Field Metadata Object Property Descriptions

Note: The FCD does not restrict user access to Search Catalog Field metadata objects.

Administration

Administration specific to the Search subsystem involves administering Search subsystem metadata in the FCD, which you or an administrator do using the Freedom Administration Client.

You may need to edit metadata for the following metadata objects:

- Search Server
- Search Catalog
- Search Catalog Field
- Knowledge Map Root
- Knowledge Map Category

For instructions on how to use the Freedom Administration Client to administer metadata in the FCD, refer to the *Freedom System Administrator's Guide*.

You or an administrator perform all other administration tasks, such as the creation and maintenance of search catalogs, using administration tools provided with the search engine. Refer to “Search Catalogs” on page 9-30 for an overview of search catalogs. Refer to documentation provided by the search engine vendor for instructions on how to administer the search engine.

Security

Freedom restricts access to Search Application ASP pages by restricting access to the directory on the Freedom server in which they reside. Freedom restricts access to the Search Component by using MTS role-based security.

The default Freedom installation assigns the following MTS roles to the Search Component:

- Administrator
- Freedom User

Add or delete MTS roles assigned to the Search Component using the MTS Explorer.

Technical Procedures

If you plan to customize the Search subsystem, first thoroughly familiarize yourself with the preceding sections. The following procedures list steps for a few simple customizations.

Modify Knowledge Map Metadata

1. Intercept the XML string the FreedomFCD.FCDRuntime object's `getKmapInfoXML()` method returns to `fs_kmapinfo.asp`'s `getKMapXML()` method. Read the `getKMapXML()` method code to decide exactly where you want to intercept the XML string.
2. Process the XML string so as to impose your modifications. For simple modifications, you may want to include an additional method in `fs_kmapinfo.asp`, call that method from within the `getKMapXML()` method while passing in the XML string, and then return the processed XML to the `getKMapXML()` method. Or, to do more extensive processing, such as filtering metadata so as to restrict user access, you may want to process the XML string exterior to `fs_kmapinfo.asp`. Whatever you decide, it is important that you not drastically disrupt the Search Application's default flow and handling of the XML string.
3. Ensure that the processed XML string is available to the `getKMapXML()` method so that it can pass the string to `fs_explorer.asp`.

Modify Result Set XML

The procedure for modifying result set XML is similar to the procedure described above for modifying Knowledge Map metadata. However, it is a little trickier because `fs_query.asp` is organized to facilitate translating result set XML into HTML.

You may want to reorganize `fs_query.asp` code into additional methods. As with modifying Knowledge Map metadata, it is important that you not drastically disrupt the flow of the default Search Application.

1. Intercept the XML result set string the Search Component's `getXMLResultSet()` method returns to `fs_query.asp`. Read `fs_query.asp`'s code. You probably want to create a copy of the returned XML string as soon as it is passed to `fs_query.asp`.
2. Process the XML string. Unless the processing is very simple, reorganize `fs_query.asp` code into additional methods. If you are extensively modifying the XML string, such as when filtering to restrict user access, call an exterior file to do the processing.
3. Ensure that the XML result set string is processed and available to `fs_query.asp` before it applies the XSL in `fs_resultset.xsl`.

Modify the Display of Result Set HTML

The following sections describe simple examples for modifying the display of result set HTML.

Change the Result Set Page Size

To change the number of documents listed in a page of the returned result set, change the value of the `intPAGE_SIZE` constant in the `fs_query.asp` page located on the Search server.

For example: `const intPAGE_SIZE = 7`

Change the Number of Documents in a Result Set

To change the maximum number of documents allowed in the entire result set, change the value of the `intMAX_DOCS` constant in the `fs_query.asp` page located on the Search server.

For example: `const intMAX_DOCS = 200`

Add or Change a Document Type-Icon Association

1. Open the `fs_resultset.xsl` file and locate the `<TD>` tag that contains the comment "Add new document types . . ."
2. To add a document type, enter the type name and the filename containing the document type icon in the area indicated.
3. To change the icon for an existing document type, replace the filename for the image associated with the document type.
4. Save the `fs_resultset.xsl` file.
5. Ensure that the appropriate image files reside on the Search server in the FreedomSearch directory.

Table 9.22 lists `fs_resultset.xsl` code showing three examples of associating a document type with a file containing an icon.

```
<TD>
  <IMG>
    <xsl:attribute name="SRC">
      <xsl:choose>
        <!-- Add new document types and associated images here-->
        <xsl:when test="TYPE[.='xls']"> fs_spreadsheet.bmp
      </xsl:when>
        <xsl:when test="TYPE[.='ppt']"> fs_presentation.bmp
      </xsl:when>
        <xsl:when test="TYPE[.='doc']"> fs_worddoc.bmp
      </xsl:when>
    </xsl:choose>
  </IMG>
</TD>
```

Table 9.22 Example `fs_resultset.xsl` Code

Change a Column Position

The order in which child elements appear within the <HIT> element produces the column order in the displayed result set. By default, Title is the fourth column. The following example lists steps for making Title the third column.

Changing a column position is a two-step process. First, change the column titles in the header row. Then change the relative positions of the corresponding <TD> tags.

1. Open fs_resultset.xsl and locate the first complete <HIT> element within the first <TR> tag. The first <TR> tag represents the header row of the result set table.
2. Cut the <TITLE> element and its attributes. Paste them immediately before the <TYPE> element.
3. Scroll down to the <TD> tag that defines the <TITLE> element (Table 9.23).
4. Cut the entire <TD> tag that defines the <TITLE> element and paste it before the <TD> tag that defines the <TYPE> element (Table 9.24).
5. Save fs_resultset.xsl.

Table 9.23 and Table 9.24 show the XSL for <TYPE> and <TITLE> as they appear within their <TD> tags. The procedure described above changes the order of the two <TD> definitions.

The <TD> tag shown in Table 9.23 frames the <TYPE> element.

```
<TD>
  <xsl:choose>
  <xsl:when test="/RESULTS/HIT[0]/TYPE/@sortable[.='true']">
    <A>
      <xsl:attribute name="TITLE">Sort by <xsl:value-of
        select="/RESULTS/HIT[0]/TYPE/@alias"/>
      </xsl:attribute>
      <xsl:attribute name="HREF"> JavaScript:SortByField("
        <xsl:value-of select=
          "/RESULTS/HIT[0]/TYPE/@alias"/>")
      </xsl:attribute> <xsl:value-of select=
        "/RESULTS/HIT[0]/TYPE/@alias"/>
    </A>
  </xsl:when>
  <xsl:otherwise><xsl:value-of select=
    "/RESULTS/HIT[0]/TYPE/@alias"/>
  </xsl:otherwise>
  </xsl:choose>
</TD>
```

Table 9.23 XSL Showing the <TD> Tag Framing the <TYPE> Element

The <TD> tag shown in Table 9.24 frames the <TITLE> element.

```
<TD WIDTH="50%">
  <xsl:choose>
    <xsl:when
      test="/RESULTS/HIT[0]/TITLE/@sortable[.='true']">
      <A>
        <xsl:attribute name="TITLE">Sort by <xsl:value-of
          select="/RESULTS/HIT[0]/TITLE/@alias"/>
        </xsl:attribute>
        <xsl:attribute name="HREF">JavaScript:
          SortByField("<xsl:value-of select=
            "/RESULTS/HIT[0]/TITLE/@alias"/>")
        </xsl:attribute><xsl:value-of select=
          "/RESULTS/HIT[0]/TITLE/@alias"/>
        </A>
      </xsl:when> <xsl:otherwise><xsl:value-of select=
        "/RESULTS/HIT[0]/TITLE/@alias"/></xsl:otherwise>
    </xsl:choose>
  </TD>
```

Table 9.24 XSL Showing the <TD> Tag Framing the <TITLE> Element

Exceptions and Logging

The Search Application does not produce unique error messages or perform any error logging; however, it propagates some error messages generated by underlying software, such as the error messages generated by IIS running on the Freedom server. The Search Component also propagates some error messages, such as those generated by the search engine or IIS running on the Search server.

Unlike the Search Application, the Search Component (`FreedomSearch.dll`) can produce unique error messages. When it detects an error, the Search Component throws an exception and logs the error using `FreedomLog`.² For information on using `FreedomLog`, refer to Appendix B, "Freedom Logger."

When the Search Component throws an exception, it creates an exception object containing details about the error. Catch and handle thrown exceptions as necessary.

Search Component error message numbers range from 9000 to 9999:

9000

Message:

```
"You must first call Init() before using this object"
```

Cause:

A Search Application ASP page did not appropriately call the `Init()` method of the Search Component.

Resolution:

Modify the ASP page so that it calls `FreedomSearch.dll`'s `Init()` method immediately after instantiating the object.

9001

Message:

```
"xxx is not a valid value for the following argument: xxx"
```

Cause:

A Search Application ASP page passed to the method an invalid argument as an actual parameter.

Resolution:

Modify the ASP page so that it passes to the method only valid parameters.

2. Freedom's default installation creates a dataset name that points to the appropriate logging database.

9002**Message:**

"xxx encountered an unexpected error; refer to this Log ID: xxx"

Cause:

An internal error occurred during execution.

Resolution:

Examine the specified error log. It should contain a relevant error message propagated from the operating system.

9003**Message:**

"Before performing this operation, you must initialize this property: xxx"

Cause:

A Search Application ASP page did not appropriately initialize the specified property.

Resolution:

Modify the ASP page so that it initializes the specified property before attempting the current operation.

9004**Message:**

"A parsing error has occurred. Please refer to this Log ID: xxx"

Cause:

The search engine encountered an error while parsing the query.

Resolution:

Examine the identified error log for a relevant error message propagated from the search engine. It is likely that the query constructed by the Search Component uses incorrect syntax.

9005

Message:

"A search error has occurred. Please refer to this Log ID: xxx"

Cause:

The search engine encountered an error while processing the query.

Resolution:

Examine the identified error log for a relevant error message propagated from the search engine. It is likely that this error is internal to the Search Component.

9006

Message:

"A Search TCP/IP error has occurred. Please refer to this Log ID: xxx"

Cause:

The search engine encountered a TCP/IP error while processing the query.

Resolution:

Examine the specified error log. It should contain a relevant error message propagated from the search engine.

9007

Message:

"This query is too complex to process: xxx"

Cause:

The search engine determined that the query was too complex.

Resolution:

Modify Search Application ASP code to prevent users from generating such complex queries. For example, adjust Knowledge Map node command attributes as the query string is built.

9008

Message:

"Before performing this operation, you must call this method: xxx"

Cause:

A Search Application ASP page failed to appropriately call the specified method.

Resolution:

Modify Search Application ASP code so that it calls the specified method before attempting the current operation.

9009

Message:

"The specified search engine does not support this property: xxx"

Cause:

A Search Application ASP page submitted an inappropriate query to the Search Component. The query requests access to an unsupported property.

Resolution:

Modify the ASP page so that it does not form queries that request access to unsupported properties.