Infopark CMS Fiona

Search Server

Infopark CMS Fiona

Search Server

Contents

1 Introductory Comments

This manual has been written for system administrators and developers who would like to provide their CMS with many and diverse search functions. The cartridge is an optional CMS component that has to be licensed separately. It can be used in the editorial system as well as on the live server.

Readers of this manual should be familiar with the installation and configuration procedures of Infopark CMS Fiona and have knowledge of search engine concepts. Integrating the Search Cartridge into a web presence also requires knowledge of objects, templates, the export procedure, and scripting.

2

2 Concepts of Infopark Search Cartridge

2.1 Operation Basics

Infopark Search Cartridge consists of a server application (Search Engine Server, SES) and Autonomy's search engine that add advanced search features to Infopark CMS Fiona. It can be used in the editiorial system as well as on the live server in order to give both editors and users of your website the opportunity to search content according to their own criteria.

Due to its architecture, the Search Cartridge can be used very flexibly and may be adjusted to the individual customer's needs. It has interfaces, for example, that make it possible to pre-process the content to be indexed or to post-process the search results (see Architecture).

Just like the Content Manager and the Template Engine, the Search Engine Server has an XML interface that makes its functions available to the clients. Not only the Content Manager and the Template Engine may act as clients, but also scripts that communicate with the SES using the XML interface.

On the live server, the SES can be used with or without the Template Engine.

2.1.1 Editorial System

In the editorial system, the Search Engine Server receives the data to be indexed from the Content Management Server. The SES optionally processes the data and sends it to the search engine. While users are working with CMS files and contents, the Content Manager makes sure that the indexed data of the Search Cartridge is kept up to date. The users normally do not notice that their creation and deletion operations cause the indexes of the Search Cartridge to be updated in the background.

Content updates and search requests made by the users of the Content Manager are communicated to the Search Cartridge by means of XML documents. These documents are structured according to an XML DTD (*Document Type Definition*). The protocol is called *Method of Interacting with Search Engines* (MISE, see MISE as DTD)

The XML documents are transferred using HTTP requests. The Content Manager acts as a Search Engine Server client, i. e. It sends update or search requests to the Cartridge that are executed and answered by the Cartridge.

2.1.2 Live System

On the live system, update and search requests are not initiated by the same client (like in the editorial system). When content has been updated, the Template engine is responsible for sending a corresponding request to the Search Cartridge. In the case of a search request, it is the task of a special

application on the live server (a PHP script, for example) to send an appropriate request to the Search Engine Server and to output the search results as an HTML page.

You can use scripts in order to directly communicate with the Search Engine Server from within documents. A Script may be used for generating an XML document from the search query entered in a form and for sending it to the Search Cartridge. The script then receives the search results from the Search Cartridge and uses them to generate the result pages that are to be delivered. Scripts and search forms can be easily maintained as CMS files using the Content Manager.

The Search Cartridge can also be used on live systems based on Ruby on Rails and the Rails Connector for Infopark CMS Fiona.

2.2 Architecture

Infopark Search Cartridge consists of the Search Engine Server and a search module by Autonomy (formerly known as Verity).

The Search Engine Server has the same architecture as other CMS applications. It is always run as a master server. The master creates slaves to which it passes the individual requests as HTTP connections. Subsequently, the slaves communicate on their own with the HTTP clients that have initiated the requests. The master server acts as a supervisor. It makes sure that there are always enough slaves available to which it can dispatch incoming connections. The limits, i. e. the minimum and maximum numbers of running slaves, for example, are set via configuration values.

Just as the Content Manager and the Template Engine, the Search Engine Server has a Tcl interface for maintenance purposes such as creating and managing collections (used for storing indexes). Furthermore, the search functions of the Search Engine Server can be extended using Tcl scripts. It is possible, for example, to pre-process search requests or to post-process search results.

Clients such as the Content Manager or a search script pass search and indexing tasks to the Search Engine Server via its XML interface. The XML interface and the DTD of the documents passed to the server via this interface make the functions of the Search Cartridge available to the clients and standardize their access to it.

2.3 Content Indexing

2.3.1 Collections

The Search Cartridge indexes documents into so-called collections. Being able to index a particular document into a particular collection can be used to accelerate the search later on. On websites in two or more languages, for example, only the content for a particular language needs to searched if the language is a search criterion. However, it is also possible to search several or all collections.

As Infopark CMS Fiona is installed, two collections are created, one for the editorial and one for the live side. Further collections can be created using a Tcl command of the Search Server. When doing this, a pre-set configuration is used for the new collection. It includes – among other things – country-specific settings such as the character set of the indexed documents and the names of the document fields whose contents are to be returned in search results. The structure of a collection cannot be changed once it has been created.

If the Template Engine is used on the live server, a collection pair is used for indexing and searching. While search requests are served from the collection that is currently online, updated content is indexed into the offline collection. Such a collection pair is called a switchable collection.

If the Template Engine is not available on the live server, the live server collections can be created by the Content Manager. Documents that are exported using the `exportSubtree` command are automatically indexed if this option has been enabled in the system configuration.

2.3.2 Indexed Data

In the editorial system, the Search Cartridge indexes the versions of files as well as some important file fields. You can configure the kinds of versions to be indexed – edited, released and archived versions may be combined as desired.

On the live server, the Template Engine indexes the exported, UTF-8 encoded documents including all their meta-data before applying the configured export encoding to them. If a file has been exported using data from other files (like frame sets, layouts for the main content), only the meta-data of the main file are indexed.

Frame sets and their frames are jointly indexed as a single document. Therefore, the frame set is included in the search results, if an associated frame matches the search query. If the Template Engine is not available on the live system, the Content Manager can create the indexes during the static export.

Since the Search Cartridge not only indexes the version fields but also the most important file fields, even searching for fields such as the file name or file format may produce search results.

In the values of fields containing HTML text (such as `body`), the SGML comments `<!-- noindex -->` and `<!-- /noindex -->` have the effect that the text between these comments is not indexed. Comments themselves are never indexed.

Each time a value of an indexed file or version field is altered or the file status changes due to workflow actions such as *Release* or *Unrelease*, the Content Manager indexes the respective version for the search in the editorial system. For the search on the live system, either the Template Engine or the Content Manager carry out the indexing of the web documents during the export.

2.3.3 Document Zones and Fields

The Search Engine Server gets the data of a version or web document to be indexed as an XML document in a request. Each attribute in such a document corresponds to an XML element. The custom version field `abstract`, for example, is stored in the XML file as follows:

```
<abstract>Summary of the document</abstract>
```

After the Search Engine Server has passed the document that is to be indexed to the search module and the latter has indexed it, all the indexed fields of the CMS file have become so called zones. Zones are named document areas that can be searched.

You can explicitely restrict search queries to one or more zones in order to search for documents that contain the search term in these zones. Such a search is called attributed, because it is not applied to the whole document but only to selected areas. If a search request is not explicitely restricted to particular zones, all zones are searched.

While document zones enable you to search through specific document parts, document fields are used for enriching the search result of each found document with the information you want to display on the results pages. In the standard configuration, for example, the version field `title` not only becomes a zone, but in addition to this, its content is stored in the document field `title`. This enables

the Search Cartridge to include the titles of the documents in the search results, for clients to use them as intended.

Document zones and fields can be configured as desired (see Configuring Collections). A version of a CMS file can have any number of version fields. During indexing the fields are transformed into the same number of document zones, provided that the configuration does not exclude zones from indexing or restricts indexing to certain zones.

In contrast to this, all indexed documents always have all document fields. If the configuration defines, for example, that the content of a zone is to be stored in a particular field, this field is always included in the indexed document, even if the respective zone is not present in the document. In this case, the field remains empty.

A client that sends a search request to the Search Engine Server can explicitely name the fields whose content it wants to be included in the search result (see Search Requests). The zones and fields available in the standard configuration can be found in section Content Search.

2.3.4 Pre-Processing During Indexing

As the Search Engine Server indexes documents, you can have every document preprocessed by a script or a program. A pre-processor can be used, for example, to add information not included in the CMS file versions themselves to the documents to be indexed. Pre-processing can also be helpful if you need to alter the encoding of the documents, i. e. to change it to the required format (UTF-8).

The Search Engine Server passes the documents to be indexed to the pre-processor without prior modification, i.e. the script or program receives the original indexing request. The pre-processor modifies the data as required and returns it to the Search Engine Server that sends it to the Autonomy search module for indexing.

2.3.5 Identification of Indexed Contents

In the process of indexing, the search module assigns each document a unique identifier, the document ID. This ID is stored as a field in the search index and is returned in the search results. The search module receives the identifiers from the Search Engine Server which in turn has received them from the respective client. Thus, the client decides which CMS file or version field should be used as the document IDs.

While the Content Management Server uses version IDs as document identifiers, the Template Engine uses file IDs. Thus, in the editorial system, the ID of an indexed document corresponds to a version ID, whereas in the live system, the document ID is a file ID.

The document ID enables the client (e.g. the user interface of the editorial system, and the Template Engine) to retrieve additional information about the document. Next to this ID, other important items are also indexed as document fields by default, the individual CMS file paths, and the titles of the relevant CMS file versions, for example. Thus, a client can extract the file paths from the search results to create result pages on which the titles of the retrieved documents are linked to the corresponding web pages.

2.4 Content Search

In the editorial system, the Search Engine Server indexes the versions of CMS files. On the live system, it indexes the exported web documents. In addition to this, in both systems the most important file fields are indexed.

The search results returned by the Search Engine Server are composed of data records. The server returns one record for each document that matches the search query. Each record consists of a set of document fields whose contents have been set during indexing (see Content Indexing). The document fields can be configured as required (see Configuring Collections).

In addition to important version and file fields, the rank of a matching document is available as a document field (`score`) in the standard configuration. The rank of a document specifies the relevance of the latter in relation to the corresponding search query. The relevance is given as a number between 0 and 100.

For both the editorial and the live system, the following table lists the document fields returned by the server for each document that matches a search query:

Document Field	Editorial System	Live System
`collection`	•	•
`score`	•	•
`docId` (from version/file ID)	•	•
`lastChanged`	•	•
`objId` (from file ID)	•	
`title`	•	•
`visiblePath`	•	•

The Autonomy search module assigns the contents of the `lastChanged` and `title` zones to the document fields with the respective names. The version and file fields listed below are indexed as zones:

Indexed File Fields	Editorial System	Live System
`name`	•	•
`objClass`	•	•
`objType`	•	•
`suppressExport`	•	
`visiblePath`	•	•
`workFlowName`	•	
File Permissions	•	

Indexed Version Fields	Editorial System	Live System
`blobLength` (from version 6.5)	•	•
`exportBlob` (exported object, not for images)	•	•

`contentType`	•	•
`lastChanged`	•	•
`mimeType`	•	•
`state`	•	
`title`	•	•
`validFrom`	•	•
`validUntil`	•	•
custom fields (excluding signature and linklist fields)	•	•

The `visiblePath` zone and field are empty in the editorial system. On the live system, they contain the path to the document.

For custom version fields of the multi-selection (`multienum`) type, each field value is indexed as a zone with the name of the version field. If such a field has several values, for each value a zone with the same name is indexed.

The same applies to file permissions: Each user group with a certain permission is indexed as a zone with the name of that file permission. An exception to this is the live server read permission (`permissionLiveServerRead`). The corresponding zone contains the names of all groups that have been given this permission. For documents that are not subject to access restrictions, the zone `noPermissionLiveServerRead` is indexed with the content `free`.

In search requests, you can execute a targeted search for documents containing the search term in one or more zones using the operator `IN`.

2.4.1 Multiple Parsers

The Infopark Search Cartridge supports search queries in several formats. For each format, a so-called parser is responsible. A parser analyzes input – in this case, search queries – and converts them into a general internal format in order to perform the actions corresponding to the input.

Search queries can be made either as free text, in explicit syntax, or in simple syntax. The default configuration uses the parser for queries in simple syntax.

The free text parser can be used for making search queries in written language, i. e. without using operators (e. g. „peace negotiations in the Middle East"). The Infopark Search Cartridge internally converts free text queries into search queries by removing unimportant words like articles, conjunctions, or prepositions (so-called stop words) and by taking into account the specifics of natural language such as noun phrases and word order. (See also the information about the operator FREETEXT).

In contrast to this, for queries in explicit or simple syntax, the search engine takes into account the operators with which search terms may be combined. For further information regarding the simple and explicit syntax as well as operators, please refer to the sections The Syntax of the Search Queries and Operators and Modifiers.

2.4.2 Pre-Processing and Post-Processing

The Search Engine Server allows each search request it receives from a client (including the Content Manager or the Template Engine) to be processed by a pre-processor before the request is passed to the search module. With such a pre-processor, terms or operations can be added to search queries, or disallowed search terms can be removed from the queries, for example. Because the search request the pre-processor receives is the XML document originally sent to the Search Engine Server, the pre-processor must be able to process XML documents.

The post-processing of search results works analogously to the pre-processing. Once the Search Engine Server has passed a (if applicable, pre-processed) search request to the search module, the module returns a search result. This result can be processed by a post-processor, in order to, for example, extend or shorten the list of the found documents or to attach to each hit an additional document field whose respective value has been calculated by the the post-processor.

2.4.3 Character Sets

The Search Cartridge uses the character encoding UTF-8. In order to be able to return search results (i. e. primarily the contents of document fields) encoded in UTF-8, the indexed documents must have this character set, too. This is ensured by the Content Manager and the Template Engine, respectively.

3

3 Configuration and Administration

3.1 Executing the Search Engine Server

The Search Engine Server is installed together with Infopark CMS Fiona. It is therefore shown only briefly here how the Search Engine Server is started.

Under Linux and Solaris, the Search Engine Server can be executed as follows by the user under whose login the CMS was installed:

```
~/instance/default/bin/rc.npsd start ses
```

If you would like to start the Search Engine Server of a different instance, replace `default` with the respective directory name. Instead of `start`, you can alternatively use the parameters `stop`, `restart`, or `status` as arguments to the start script `rc.npsd` in order to stop the server, make a restart or have it return its status.

The `instance` directory is located below the CMS installation directory.

Please note that under Windows, you need to be logged-in as an administrator in order to start the Search Engine Server. You can execute it by means of the Windows start menu.

Please note that the Search Engine Server, the Content Management Server, and the Template Engine need to be adjusted to each other in order to be able to communicate with each other within a Fiona installation. Adjustments are also necessary with respect to the collections to be used, if you want to use more or other collections than are pre-configured.

3.2 Configuring Searching and Indexing

The Search Engine Server itself as well as its search and indexing behavior can be configured by means of the following system configuration entries.

- `server.ses`
- `indexing`
- `searching`
- `tuning`

The corresponding configuration files can be found in the `config` directory below the instance directory concerned, i.e., for example, in `instance/default/config`.

As with all CMS applications, the Search Engine Server reads its configuration only at start-up. It therefore needs to be restarted after changes have been made to the configuration.

3.3 Integrating an External Preprocessor

3.3.1 General Notes

By means of an external preprocessor, documents can be modified before they are indexed. This makes it possible to convert binary data to text, or to generate or extract meta data (from images, for example) for the purpose of indexing. As a result, searches will (better) find the documents concerned. You can define as many preprocessors as you require.

Documents of any MIME type can be associated with a preprocessor. This can be done by means of the `indexing` section in the system configuration. Any suitable program can be used as an external preprocessor. Optionally, arguments can be passed to such a program.

3.3.2 Functionality

The preprocessor program receives the document to be indexed via `stdin` from the Search Server. The document passed to the preprocessor is a serialized XML document. The preprocessor modifies it in the desired way and returns it to the Search Server via `stdout`. The Search Server then indexes the modified document. An example:

Original data:

```
<ses-indexDoc docId="2148" collection="cm-contents"
    mimeType="application/vnd.ms-excel">
<title encoding="plain">Ein Beispiel mit Excel-Daten</title>
<keyword encoding="plain">Beispiel</keyword>
<blob encoding="stream" mimeType="application/vnd.ms-excel">
  /Fiona_671/instance/default/tmp/externalPreprocessor/1.dat
</blob>
</ses-indexDoc>
```

Modified data:

```
<ses-indexDoc docId="2148" collection="cm-contents"
    mimeType="application/vnd.ms-excel">
<title encoding="plain">Excel-Daten als Text</title>
<keyword encoding="plain">Beispiel</keyword>
<blob encoding="stream" mimeType="text/plain">
  /Fiona_671/instance/default/tmp/text_data.dat
</blob>
</ses-indexDoc>
```

The XML document contains the fields to be indexed (the names of the XML elements) as well as their values (the content of the XML elements). A field value may either be contained directly in the element's content (encoding: `plain`) or it may have been encoded. The encoding can be determined by means of the `encoding` tag attribute of the field element. Its value can be one of:

- `plain`: The field value is the content of the XML element.
- `base64`: The field value can be determined by base64-decoding the content of the XML element.
- `stream`: The field value is contained in the file whose path is specified in the content of the XML element.

From version 6.7.1, for all encodings except `plain` the MIME type of the document is provided as the value of the `mimeType` tag attribute of the field element. If the MIME type is changed during preprocessing, the `mimeType` attribute must be set to the MIME type of the resulting field value. If the encoding is not `plain`, a field value will only be indexed if its MIME type matches `text/*`. In other words: if a preprocessor produces base64-encoded or streamed field values, it must set their MIME type to a text type.

Up to version 6.7.0, the preprocessed field values are required to be `plain`, i.e. not encoded. Encoded field values will not be indexed.

3.3.3 Configuration

The preprocessor to be used, the MIME types to which it is applied, and the arguments to be passed to it can be specified in the `indexing.xml` configuration file. The corresponding section might look like this, for example:

```
...
<contentPreprocessors type="list">
  <preprocessor>
    <mimeTypes type="list">
      <mimeType>application/pdf</mimeType>
    </mimeTypes>
    <processor type="external">
      bin/tclsh
    </processor>
    <processorArguments type="list">
      /Fiona_671/instance/default/script/custom/pdf2TxtWrapper.tcl
    </processorArguments>
  </preprocessor>
  ...
</contentPreprocessors>
...
```

Here, the Tcl interpreter was specified as the preprocessor program to use. To this program the name of the script to be executed is passed as an `argument` in the `processorArguments` element. Since the script cannot be loaded during server startup, it should not be placed into the `serverCmds` or `clientCmds` directory.

The following sample script, `pdf2TxtWrapper.tcl`, demonstrates how a PDF document, which is containd as the `blob` field in the XML document, can be read and converted to text. Please note that no preprocessor is required for the Search Server to index PDF documents.

```
# Libraries
package require dom
package require base64
proc safeInterp {args} {}
source [file join [file dirname [info script]]\
    ../../../share/script/common/clientCmds/util.tcl]

# Read Data
set xmlRequest [read stdin]

# Parse XML
set docNode [::dom::DOMImplementation parse $xmlRequest]
set rootNode [::dom::document cget $docNode -documentElement]

# Select and handle element "blob"
set blobElement [lindex [::dom::selectNode $rootNode descendant::blob] 0]
array set attributes [array get [$blobElement cget -attributes]]
set blobTextNode [$blobElement cget -firstChild]
if {$blobTextNode ne ""} {
```

```
  set value [$blobTextNode cget -nodeValue]
  if {$value ne ""} {
    switch $attributes(encoding) {
      plain {
        # shouldn't happen with pdf
        set blob $value
      }
      base64 {
        set blob [::base64::decode $value]
      }
      stream {
        set blobFile $value
      }
    }
    set deletePdfFile 0
    if {![info exists blobFile]} {
      set blobFile "/tmp/convert_me_[pid].pdf"
      writeFile $blobFile $blob
      set deletePdfFile 1
    }
    set textFile "/tmp/converted_[pid].txt"
    # convert using ps2ascii
    if {![catch {
      exec ps2ascii $blobFile $textFile
    }]} {
      # modify the dom tree
      $blobTextNode configure -nodeValue $textFile
      ::dom::element setAttribute $blobElement mimeType "text/plain"
      ::dom::element setAttribute $blobElement encoding stream
    }
    if {$deletePdfFile} {
      file delete -force $blobFile
    }
  }
}
set xmlToReturn [string trimright [::dom::DOMImplementation serialize $docNode] "\n"]
set lines [split $xmlToReturn "\n"]
if {[string match "<!D*" [lindex $lines 1]]} {
  set xmlToReturn [join [lreplace $lines 1 1] "\n"]
}
# return the (modified) xml data
puts -nonewline $xmlToReturn
```

3.4 Configuring Collections

New collections are created on the basis of configuration files. These files mainly determine the document zones to be indexed and the document fields to set in this process. Only those elements of the indexed documents that are treated as document zones can be transferred to document fields (see also Document Zones and Fields).

These properties of collections are controlled by so-called `style` files. When a collection is created (Administration Commands), the supplied style files located in the `config/vdk/style` directory are copied to the `style` directory below the collection directory. They are meant to be used as templates.

If you would like to change the basic properties of all future collections, you can change the zone and field definitions in the style templates. If, on the other hand, you would like to modify the configuration of a newly created collection, then you can edit the style files of this individual collection. The configuration of a collection to which documents have already been added should not be changed, unless it is purged with `purgeCollection` before. Changes to the configuration files of a collection must not be made during an indexing process. The Search Engine Server must be stopped first.

After having created a new collection, you should adapt the rules used to determine the collection into which CMS files are indexed (a CMS file can only be indexed into one collection). This can be done by means of the indexing.incrementalExport.collectionSelection system configuration entry. The search, however, always includes all collections unless the list of the collections to use is explicitly specified in the search request.

3.4.1 Defining Document Zones

Document zones can be defined in the file style.xml. Inside this file, the following elements are available as instructions for defining how XML tags (i. e. attributes in contents) are to be handled:

The following command ignores all XML tags in the document, indexing only the content of the XML elements:

```
<ignore xmltag = "*"/>
```

The following instruction skips indexing the specified xmltag but indexes the content between its start and end tags of the specified xmltag :

```
<ignore xmltag = "section_1"/>
```

The following instruction indexes the XML element identified by xmltag as a zone if there is also an ignore xmltag="*" instruction:

```
<preserve xmltag = "section_1"/>
```

The following instruction suppresses the entire element identified by xmltag. The tag, attributes, and content are not indexed:

```
<suppress xmltag = "section_1"/>
```

The following instruction indexes the content between the start and end tags of the specified xmltag as a field which is given the fieldname identified by fieldname. If fieldname is not specified, the tag name is used as field name. Any existing value of the field is overridden if the optional attribute index="override" has been specified.

```
<field xmltag="column_2" fieldname="vdk_field_2" index="override"/>
```

The elements to be indexed as zones can be defined inclusively or exclusively. When defined exclusively, all elements are indexed except the ones whose name has been specified using <ignore xmltag="..."/>. To define zones inclusively, <ignore xmltag="*"/> is used to exclude all elements first. Then the elements to be indexed are included explicitly by using <preserve xmltag="..."/> for each of them.

With both methods, inclusive and exclusive, the contents of the elements (i. e. the zones) can be stored in fields. This makes it possible to return these values in the search result for each selected document. It is not possible to ignore an element and to store its content in a document field at the same time.

3.4.2 Defining Document Fields

With document fields the Verity module makes a distinction between standard and user-defined fields. Standard fields are configured in the *style.sfl* file, user-defined ones in *style.ufl* . If you wish to define fields in order to provide for additional information concerning your documents in the search result, please take notice of the following:

- Each field increases the amount of memory needed in proportion to the number of indexed documents.
- Field values are set during the indexing process and can be included in the search result during a search.The more fields you have defined or are included in the search result the longer this process will take.

The names of the standard document fields are defaults of the Verity module which you should not change to keep your collections compatible with third party collections. However, you can define alias names for document fields instead, if required.

The format of the *style.sfl* and *style.ufl* files is identical. The Verity module reads them in via the *style.ddd* file. Fields in these files can be defined in accordance with the following example:

```
data-table: nps{
    varwidth: objId       dda
    /minmax = yes
    /alias = objectId
    autoval:  collection  DBNAME}
```

The keyword `data-table` serves to define the name of a table in a data segment for the Verity module. In the example above, the name is `nps`.

With each field a storage type is associated which determines the source of the field value and how it is stored. The following storage types exist:

Storage type	Meaning and usage
`autoval`	Assigns an internal value of the Verity module to the field. **Syntax:** `autoval:` *FieldName* `DBNAME|DBPATH` *FieldName*: Name of the field. `DBNAME`: Collection name. `DBPATH`: Path of the collection.
`constant`	Assigns a constant value to the field. **Syntax:** `constant:` *FieldName DataType Value* *FieldName*: Name of the field. *DataType*: One of the data types listed below. *Value*: The field value. Must be enclosed in quotes if it contains space characters.
`worm`	Assigns a value from a document zone to a field. The value cannot be changed afterwards (*write once, read many times*). **Syntax:** `worm:` *FieldName DataType* *FieldName*: Name of the field. *DataType*: One of the data types listed below.
`fixwidth`	Assigns a value from a document zone to a field. The value has a fixed length and will be truncated if necessary. **Syntax:** `fixwidth:` *FieldName Length DataType* *FieldName*: Name of the field.

	Length: The maximum length of the field value in characters. *DataType*: One of the data types listed below. **Modifier:** `/minmax = yes|no`: Speeds up searches in the values of this field, especially with larger collections and if the field values lie within a particular range.
`varwidth`	Assigns a value from a document zone with any length to a field. **Syntax:** `varwidth:` *FieldName StorageLocation* *FieldName*: Name of the field. *StorageLocation*: A three-character string which must not start with _ or `di`. **Modifier:** `/minmax = yes|no`: see `fixwidth`.

Field names can consist of up to 128 alphanumeric characters and must not begin with an underscore character. As storage location for the `varwidth` storage type, specify `dda`, for example.

Every document field can have one of the following data types:

Data type	Meaning
`text`	The field contains ASCII characters.
`date`	An internal date format used to store date and time values in the range from 1904 to 2037. This data type is not compatible with the date and time format of the Search Cartridge.
`signed-integer`	A signed integer number whose range depends on the size of the field. 1 Byte: -128 to 127 2 Bytes: -32768 to 32767 4 Bytes: -2e31 to 2e31 -1
`unsigned-integer`	An unsigned integer number whose range depends on the size of the field. 1 Byte: 0 to 255 2 Bytes: 0 to 65535 4 Bytes: 0 to 2e32 -1

Document fields that have been created using the `minmax` option allow for storing up to 256 bytes in them. If the size of the value to be stored is greater, then only the first 256 Bytes are stored in the field. By means of the `alias` option an alternate name can be specified under which the field can be addressed in search queries.

3.5 Defining Stopwords

Stopwords are words that can be found very often in texts (articles, prepositions, etc.) and therefore lead to a large number of irrelevant search results when searching for them.

With search queries in whose search text stopwords occur – however not exclusively – the hits relating only to one or more stopwords can be suppressed. To do this, proceed as follows:

1. Enter the desired stopwords into the the file *`installation_dir`*`/3rdparty/vdk/common/uni/vdk30.stp`. Place each stopword onto an individual line. (As a template you might use the stopwords in the files *`installation_dir`*`/3rdparty/vdk/common/uni/stopword.`*`xx`*.)
2. Restart the SES.

The defined stopwords will now be observed in search queries using the FREETEXT operator. Example: If "the" has been defined as a stopword, the search query

```
<#FREETEXT> "the Teddy"
```

will find only documents containing the word "Teddy".

If a FREETEXT search query exclusively contains stopwords, they will not be ignored.

3.6 Defining Synonyms

By means of the thesaurus function of the Autonomy search engine, synonyms can be defined. This makes it possible to get hits even if only a term with a similar meaning (a synonym) instead of the search term itself can be found in the content.

A thesaurus can be defined and installed in the following way:

1. Define the thesaurus as a text file, vdk30.txt. Example:

   ```
   $control:1
   synonyms:
   {
   list: "publication,magazine,newspaper,journal"
   list: "law,statute,bill"
   }
   $$
   ```

   The definition consists of synonym lists, each of them occupying an individual line.
2. Compile the thesaurus using mksyd. If you are using the uni locale, the following steps are required under Linux:

   ```
   > export PATH=$PATH:installation_dir/3rdparty/vdk/_ilnx21/bin
   > export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:installation_dir/3rdparty/vdk/_ilnx21/bin
   > mksyd -locale uni -f vdk30.txt -syd vdk30.syd
   ```

   Unter Windows you can extend the search path using

   ```
   set PATH=%PATH%;installation_dir\3rdparty\vdk\_ilnx21\bin
   ```

   Then execute the command given above.
3. To install the thesaurus, please copy it to the location where the search engine expects it to be:

   ```
   > cp vdk30.syd Installationsverzeichnis/3rdparty/vdk/common/uni
   ```

   Under Windows, please use copy instead of cp and replace the slashes with backslashes.
4. Restart the SES:

   ```
   > installation_dir/instance/instance_name/bin/rc.npsd restart SES
   ```

The defined synonyms can now be used in search queries made with the `THESAURUS` operator. Example: the search query

```
<#MANY><#THESAURUS>"publication"
```

will find all documents containing the word "publication" or one of its synonyms.

3.7 Treating Hyphens as Whitespace

With the `uni` locale selected, the search engine treats hyphens contained in the content as normal alphabetic characters by default. If a document contains the phrase `six-year-old sister`, for example, searching for `year*` will not retrieve this document.

To change this behavior, the hyphen needs to be removed from the list of additional alphabetic characters. This list is contained in the `ctype.cfg` file located in the *`installation_dir`*`/3rdparty/vdk/common/uni` directory. The list defaults to the underscore, the hyphen, and the ampersand characters, given as hexadecimal character values:

```
ALPHA: 0x26,0x2d,0x5f
```

Remove the hyphen (`0x2d`) from the list, save the `ctype.cfg` configuration file, and restart the SES. Afterwards, the content needs to be reindexed using the `indexAllObjects` Tcl command of the Content Manager.

3.8 The Tcl Interface

3.8.1 Administration Commands

Collections hold the data of the Infopark Search Cartridge. Essentially, they contain the indexes and word lists that are produced when documents are indexed.

Switchable collections - the collections used on the live system - are located in the `export/offline/collections` directory below the instance directory, i. e., for example, `instance/default/export/offline/collections`. Each collection is represented by an individual directory whose name equals the collection name.

Non-switchable collections are mainly used on the editorial system. They are only used on the live system if it does not include a Template Engine. The storage location of non-switchable collections is the `data/ses/collections` directory below Fiona's instance directory. Like switchable collections, non-switchable ones are located in individual subdirectories whose names correspond to the collection names.

The Autonomy search engine module uses so-called style files when it creates collections. If collections are created using the `createCollection` Tcl command described in this section, the supplied style files located in the `config/vdk/style` directory are used. If, however, this directory contains a directory named like the collection to be created, then the style files in this directory will be used. These files are copied to the `style` directory below the corresponding collection directory.

Of course you can also use Autonomy's `mkvdk` program to create collections. This tool as well as other administration tools can be found in the directory `3rdparty/vdk/_ilnx21/bin` (Unix) or `3rdparty/vdk/_nti40/bin` (Windows).

The locales and the Autonomy configuration files can be found in the `3rdparty/vdk/common` directory. Each locale has its own directory here whose name equals the name of the locale.

For the purpose of administering collections, numerous Tcl commands are available in the server's so-called single mode. In single mode the Search Engine Server is no server but a command line program.

Please note that the Search Engine Server must be stopped prior to executing commands with which one ore more collections can be modified (see Executing the Search Engine Server).

You can execute the Search Engine Server in single mode by running the executable file in a shell, specifying the command line argument `-single`. Under Linux and Solaris this can be done from the instance's `bin` directory:

```
./SES -single
```

Under Windows, use the following command:

```
SES -single
```

The Search Engine Server then displays a prompt. You can now use all standard Tcl commands as well as the commands listed in the following. Most of these commands call Autonomy's `mkvdk` program. The Search Engine Server's commands, however, are easier to use than `mkvdk`.

3.8.2 aboutCollection

Available for: Search Engine Server

Task: This command displays information about the specified collection (such as the number of documents in the index or the locale used).

Syntax

`aboutCollection` *`collectionName`*

Function parameters

- *`collectionName`*: The name of the collection for which information is to be displayed. For switchable collections this command refers to the offline collection.

Return value: Information about the collection in textform. If the specified collection does not exist, the error message "The collection '*`collectionName`*' does not exist" is displayed.

Example (extract)

```
SES>aboutCollection sportsNews
mkvdk - Verity, Inc. Version 5.0.1 (_ilnx21, Aug 6 2003)
Collection about resources:
  Last Purge Date: 19-Oct-2010 02:16:50 pm
  Creation Date: 16-Oct-2010 12:11:29 pm
  Modification Date: 19-Oct-2010 02:20:57 pm
  Last Squeeze Date: 0000000000
  Number of Documents: 64071
  Collection Creator: INFOPARK AG
  Collection Name: sportsNews
  ...
mkvdk done
```

3.8.3 backupCollection

Available for: Search Engine Server

Task: The command creates a copy of a collection.

Syntax

`backupCollection` *`collectionName targetDir`*

Function parameters

- *`collectionName`*: The name of the collection to copy. For switchable collections this command refers to the offline collection.
- *`targetDir`*: The target directory of the new collection, relative to the collection's parent directory. Under Windows the backslash must not be used as path separator. Use two backslashes or the normal slash instead.

Return value: none.

Example

```
SES>backupCollection sportsNews liveSportsNews
```

3.8.4 createCollection

Available for: Search Engine Server

Task: This command creates a new collection.

Syntax

`createCollection` *`collectionName`* [*`switchable`*]

Function parameters

- *`collectionName`*: The name of the collection to be created. The name must be a valid file name. A collection with this name, whether switchable or not, must not already exist.
- *`switchable`*: The value can be 0 or 1. It determines whether the collection to be created is to be switchable (1) or not (0, the default). In conjunction with the incremental export, i.e. when using the Search Server together with the Template Engine you require switchable collections. Non-switchable collections are meant to be used only with content that is exported statically.

Return value: If the command has been be executed successfully it returns the name of the collection created.

Additional Information

- The names of the default collections are cm-contents (editorial system) and live-docs (live system). The Search Engine Server creates these collections automatically at startup if no collections exist.

- The style files used for creation are located in the `config/vdk/style` directory. If, however, this directory contains a directory named like the collection to be created, then the style files in this directory will be used. These files are copied to the `style` directory below the corresponding collection directory.

Example

```
SES>createCollection sportsNews
sportsNews
```

3.8.5 deleteCollection

Available for: Search Engine Server

Task: The command completely deletes a collection, including all its data and its configuration. The user is not asked to confirm this action. Therefore, the command should be used only after the consequences have been considered thoroughly.

Syntax

`deleteCollection` *`collectionName`*

Function parameters

- *`collectionName`*: The name of the collection to delete. For switchable collections this command refers to the offline collection.

Return value: none.

Example

```
SES>deleteCollection sportsNews
```

3.8.6 listCollections

Available for: Search Engine Server

Task: This command returns the list of the names of all collections of the Verity search engine module.

Syntax

`listCollections`

Function parameters

The command has no function parameters.

Return value: The list of the collections known to the Autonomy search module

Example

```
SES>listCollections
cm-contents live-docs
```

3.8.7 purgeCollection

Available for: Search Engine Server

Task: This command empties a collection by removing all indexed documents from it. The collection itself, however, is preserved.

Syntax

`purgeCollection` *`collectionName`*

Function parameters

- *`collectionName`*: The name of the collection to purge. For switchable collections this command refers to the offline collection.

Return value: The command has no return value. If the specified collection does not exist, the error message "The collection '*`collectionName`*' does not exist." is displayed. If the command fails, the message "Purging collection '*`collectionName`*' failed." is displayed.

Example

```
SES>purgeCollection sportsNews
```

3.8.8 repairCollection

Available for: Search Engine Server

Task: This command tries to repair a collection that has become inconsistent on a file level because of an unwanted deletion, for example.

Syntax

`repairCollection collectionName`

Function parameters

- *`collectionName`*: The name of the collection to be repaired. For switchable collections this command refers to the offline collection.

Return value: The command returns information about the actions taken.

Example (for a collection that does not need to be repaired):

```
SES>repairCollection sportsNews
mkvdk - Verity, Inc. Version 2.7.0 (_ilnx21, Feb 15 2001)
mkvdk done
```

3.8.9 Other Commands

Next to the commands for collection administration, the following commands are available in the Tcl interface of the Search Engine Server.

Commands only available in the SES

app flushQueue
app holdQueue
app resumeQueue

Commands also available in other CMS Fiona applications

app get
decodeData
decodeFile
decodeToString
encodeData
encodeFile
stream_uploadFile
stream_uploadBase64
stream_downloadFile
stream_downloadBase64

app flushQueue

Available for: Search Engine Server

Task: This command causes the Search Engine Server to start processing indexing requests, independently of the system configuration entries mentioned below. The command can only be executed if the SES is in the `indexingNormal` state.

Syntax

```
app flushQueue
```

Additional information

The state of the SES with respect to indexing can be determined by reading out the `sesCommandState.state` file located in the *`cmsInstanceDir`*`/data/ses/otherData/002/008` directory. The file contents has the following meaning:

- `indexingNormal`: Indexing requests are accepted and processed in accordance with the presets given by the `tuning.indexing.interval` and `tuning.indexing.maxBulkSize` system configuration entries. This state is induced by the `app resumeQueue` command.
- `indexingDelayed`: Indexing requests are not processed. This state is induced by the `app holdQueue` command.

Function parameters: none.

Return value on success: status information.

app holdQueue

Available for: Search Engine Server

Task: This command causes the Search Engine Server to not accept new indexing requests. The Template Engine uses this command prior to synchronizing collections.

Syntax

```
app holdQueue
```

Additional information

The state of the SES with respect to indexing can be determined by reading out the `sesCommandState.state` file located in the *`cmsInstanceDir`*`/data/ses/otherData/002/008` directory. The file contents has the following meaning:

- `indexingNormal`: Indexing requests are accepted and processed in accordance with the presets given by the `tuning.indexing.interval` and `tuning.indexing.maxBulkSize` system configuration entries. This state is induced by the `app resumeQueue` command.
- `indexingDelayed`: Indexing requests are not processed. This state is induced by the `app holdQueue` command.

Function parameters: none.

Return value on success: status information.

app resumeQueue

Available for: Search Engine Server

Task: This command causes the Search Engine Server to accept and process indexing requests again. The Template Engine uses this command after it has finished synchronizing collections.

Syntax

```
app resumeQueue
```

Additional information

The state of the SES with respect to indexing can be determined by reading out the `sesCommandState.state` file located in the *`cmsInstanceDir`*`/data/ses/otherData/002/008` directory. The file contents has the following meaning:

- `indexingNormal`: Indexing requests are accepted and processed in accordance with the presets given by the `tuning.indexing.interval` and `tuning.indexing.maxBulkSize` system configuration entries. This state is induced by the `app resumeQueue` command.
- `indexingDelayed`: Indexing requests are not processed. This state is induced by the `app holdQueue` command.

Function parameters: none.

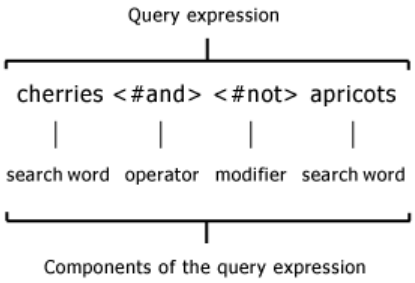
Return value on success: status information.

4

4 The Syntax of the Search Queries

4.1 Search Queries

A query expression (also called search query in the following) defines the criteria according to which the search module of the Infopark Search Server performs a search. A query expression consists of search words, operators, and modifiers:



Operators and modifiers are keywords that are used in search expressions to define the relationship between search words. Only documents in which the search words have the specified relationship can show up in the search result. For example, the query expression

```
cherries <#AND> <#NOT> apricots
```

searches for documents containing the search words `cherries` but not `apricots`.

Keywords should always be enclosed in angle brackets. If the Search Engine is operated with a non-English localisation, keywords must always be prefixed with a Pound sign (or hash mark, depending on your keyboard's layout). This is not necessary if the English locale is used.

With operator and modifier names the Search Engine does not distinguish between uppercase and lowercase letters.

4.1.1 Parser

The Search Engine is equipped with three so-called parsers which analyze the user's search queries and perform the corresponding search actions. The parsers differ from each other in the way they analyze and interpret query expressions. Each of the three parsers was designed for a certain field of application. The parser to be used can be specified in XML search query requests to the Search Engine Server. Therefore, a more or less complex query language can be provided in query forms, depending on the target user group.

A parser should not be equated with the query syntax it supports. It is, for example, possible to use the simple syntax as well as explicit syntax in the simple parser. Furthermore, explicit syntax allows you to use several notations.

4.1.2 Simple Parser

The simple parser is a universal one because it can be used for queries in simple as well as in explicit syntax. The parser is favoured in environments in which the users want to get the best results with the least effort.

The simple parser converts simple queries to explicit queries, adding operators where it seems appropriate. Each search word is implicitly preceeded with the `MANY` and the `STEM` operators.

- `MANY` causes the document's relevance to grow as the density of a word's occurrence in a document grows. Density is a relative measure that specifies the relationship between the number of occurrences of a search word and the amount of text it contains.
- `STEM` causes the search to include also words that are variations of the search word.

If a user enters search words separated with commas, then the words are combined with the `ACCRUE` operator. This operator causes a document's relevance to grow as the absolute number of occurrences of the search word grows. A query such as

```
apple, banana, orange
```

is therefore converted by the parser as follows:

```
<#accrue>(<#many><#stem>apple, <#many><#stem>banana, <#many><#stem>orange)
```

4.1.3 Explicit Parser

The explicit parser is limited to processing search expressions written in explicit syntax. It was designed for environments in which search queries are generated under control of a software program. In such environments, instead of typing operators, users use checkboxes, for example, to select the operations to be applied to the search words that have been entered.

Search queries in explicit syntax can be made using prefix or infix notation:

- Prefix notation uses parentheses to make an operator's precedence explicit. For example, the expression `<#OR> (a, <#AND> (b, c))` retrieves documents containing the search word `a` or a combination of `b` and `c`.
- With infix notation the precedence of operators is implicit, i. e. associated with the operators themselves, unless the precedence is modified with parentheses. This applies mainly to the `AND` and `OR` operators of which `OR` has less precedence. As a consequence, search words combined with `AND` are processed before those combined with `OR`. Example: `a <#AND> b <#OR> c` searches for documents containing `a` as well as `b`, or `c`.

The following query is an example for prefix notation:

```
<#paragraph>("vehicle", <#sentence>("safety", <#phrase>("no", "compromise")))
```

Using infix notation, this query is stated as follows:

```
"vehicle" <#paragraph> "safety" <#sentence> "no" <#phrase> "compromise"
```

Literal Text

When you enclose individual words in double quotation marks, the Autonomy search engine interprets those words literally. For example, by entering the word "film" explicitly in double-quotation marks, the words "films", "filmed", and "filming" will not be considered in the search:

```
"film"
```

The quotation marks are a syntactical element of the explicit syntax and can be used in the simple and the explicit parser. The following example retrieves documents that contain both the literal phrase "pharmaceutical companies" and the literal word "stock”:

```
AND ("pharmaceutical companies", "stock")
```

The following example retrieves documents containing the phrase "black and white":

```
<PHRASE> (black "and" white)
```

The PHRASE operator does require angle brackets, and the "and" is enclosed in double quotation marks because it is to be interpreted as a literal word, not as an operator.

Additionally, when you enter a topic name enclosed in double quotation marks, the search engine will interpret the topic name as a literal word instead of a topic. This is useful if you want to search for a word that is the same as the name of a topic.

4.1.4 Freetext Parser

The free text parser allows you to make search queries that equal sentences or part of a sentence. It treats all text as a series of search words. As a consequence, operators will not be identified.

The free text parser generates from a search query a request in explicit syntax by removing unimportant words like articles, prepositions, and conjunctions from the query and combining the search words, resulting in a sequence of words. The parser allows users to make queries in the form of short questions.

Using the FREETEXT operator, the free text functionality is also available in the simple and explicit parsers.

4.2 Non-English Environments

4.2.1 Using the English-Language Query Language

The keywords (operator and modifier names) in search queries are language-specific. If, for example, the Search Cartridge is operated using the German-language setting germanx (an optional so-called locale), the German equivalents of some English operator and modifier names will be identified as operators or modifiers, respectively, even if they were meant to be search words in a query. These

words "und", "oder", "nicht" and others) must be quoted when they are to be interpreted as search words. Analogously, English words such as "and", "or", "any", "not" that correspond to operator or modifier names need to be enclosed in quotes if they are meant literally.

It is possible and recommendable to use the English-language operator and modifier names in search queries. This can be done by prefixing each operator or modifier name with a Pound sign (or hash mark, depending on your keyboard layout). In order to search for the phrase "Infopark AG" in a non-English-language environment, enter the following expression:

```
<#PHRASE> Infopark AG
```

In explicit syntax the expression looks like this:

```
<#PHRASE> (Infopark, AG)
```

4.2.2 Tokenization

In environments in which languages such as English or German are used, the Search Cartridge can interpret space characters as word separators. For these languages no special language-specific tokenization modules are required since the Search Cartridge is able to identify words, phrases, and sentences appropriately. However, for other languages, such as Japanese and Chinese, a tokenization module is required in order to determine the bounds of words. If you are using a special tokenization module, some sections of the following query language descriptions might not apply.

5

5 Operators and Modifiers

5.1 Operator Types

An operator expresses an operation that is applied to a part of the search expression. This operation defines the restrictions a document must meet in order to be placed in the resulting list of hits. Many operators exist, and they can be classified according to their type:

Standard Operators

- Concept operators
- Evidence operators
- Proximity operators

Special operators

- Operators for analyzing written language
- Ranking operators
- Field operators and relational operators

5.1.1 Concept Operators

Concept operators combine the meaning of search elements to identify a concept in a document. Documents retrieved using concept operators are relevance-ranked. The following table describes each concept operator:

Operator name	Description
`ACCRUE`	Selects documents that include at least one of the search elements you specify. The more search elements are present, the higher the score will be.
`ALL`	Selects documents that contain all of the search elements you specify. A score of 100 is assigned to each retrieved document. `ALL` and `AND` are similar and they retrieve the same results. Queries using `ALL` are not relevance-ranked (all retrieval results are assigned a score of 100).
`AND`	Selects documents that contain all of the search elements you specify. A score is calculated for each retrieved document. `AND` and `ALL` are similar and they retrieve the same results. Queries using `AND` are relevance-ranked (retrieved documents are assigned a score between 0 and 100).

ANY	Selects documents that contain at least one of the search elements you specify. A score of 100 is assigned to each retrieved document. ANY and OR are similar and they retrieve the same results. Queries using ANY are not relevance-ranked (all retrieval results are assigned a score of 100).
OR	Selects documents that contain at least one of the search elements you specify. A score is calculated for each retrieved document. OR and ANY are similar and they retrieve the same results. Queries using OR are relevance-ranked (retrieval documents are assigned a score between 0 and 100).
TOPIC	Selects documents that contain at least one of the search elements you specify, covered by the specified topic. A score is calculated for each retrieved document. How topics work and how they can be configured is described in the Verity documentation which is available as an option.

5.1.2 Evidence Operators

Evidence operators can be used to specify either a basic word search or an intelligent word search. A basic word search finds documents that contain only the word or words specified in the query. An intelligent word search expands the query terms to create an expanded word list so that the search returns documents that contain variations of the query terms. For example, the THESAURUS operator selects documents containing the word specified, as well as its synonyms.

Documents retrieved using evidence operators are not relevance-ranked unless you use the MANY modifier. The following table describes each evidence operator.

Operator name	Description
WORD	Performs a basic word search, selecting documents that include one or more instances of the specific word you entered.
STEM	Expands the search to include the word you entered and its variations.
THESAURUS	Expands the search to include the word you entered and its synonyms.
WILDCARD	Matches wildcard characters included in search strings. Certain characters automatically indicate a wildcard specification.
SOUNDEX	Expands the search to include the word you enter and one or more words that "sound like", or whose letter pattern is similar to, the word specified. Collections do not have sound-alike indexes by default; to use this feature sound-alike indexes must be built.
TYPO/*N*	Expands the search to include the word you enter plus words that are similar to the query term. This operator performs "approximate pattern matching" to identify similar words.

5.1.3 Proximity Operators

Proximity operators specify the relative location of specific words in the document; i. e., specified words must be in the same phrase, paragraph, or sentence for a document to be retrieved. In the case of the NEAR and NEAR/*N* operators, retrieved documents are relevance-ranked based on the proximity of the specified words.

When proximity operators are nested, the ones with the broadest scope should be used first; that is, phrases or individual words can appear within SENTENCE or PARAGRAPH operators, and SENTENCE operators can appear within PARAGRAPH operators. The following table describes each proximity operator.

Operator name	Description
IN	Selects documents that contain specified values in one or more document zones. A document zone represents an attribute in a content, for example the date of the last change (lastChanged) or the body text.
PHRASE	Selects documents that include a phrase you specify. A phrase is a grouping of two or more words that occur in a specific order.
SENTENCE	Selects documents that include all of the words you specify within the same sentence.
PARAGRAPH	Selects documents that include all of the search elements you specify within the same paragraph.
NEAR	Selects documents containing specified search terms, where the closer the search terms are within a document, the higher the document's score.
NEAR/N	Selects documents containing two or more search terms within *N* number of words of each other, where *N* is an integer up to 1024. The closer the search terms are within a document, the higher the document's score.

5.1.4 Operators for Analyzing Written Language

The FREETEXT and LIKE operators serve to analyze written language. The search engine translates the text of the search query into the search syntax. The query is then performed, and the documents found are ranked according to their relevance. The two operators are intended mainly for application development.

Operator name	Description
FREETEXT	This operator evaluates the search query text as if the query had been made in the free text parser (see Search Queries). In particular, articles, prepositions, and conjunctions are removed. The word order is taken into account as well. The free text parser has been optimized for analyzing short questions in written language and translate them into the query language.
LIKE	This operator performs a search on the basis of positive or negative sample texts and ranks the documents according to the degree of correspondence (QBE = "Query by example").

5.1.5 Scoring Operators

The scoring operators influence the way in which the search module ranks the documents it finds, i. e. how it calculates their scores. These operators can be combined with each other and with other operators of the query language.

When a scoring operator is used, the search engine first calculates a separate score for each element of the search expression found in the document. Then the resulting score is calculated from these scores using a mathematical operation.

The YESNO operator can be useful in many situations, whereas the PRODUCT, SUM, and COMPLEMENT operators are meant to be used by application developers who want to generate queries programmatically.

Operator name	Description
COMPLEMENT	This operator subtracts a document's total score from 100.
PRODUCT	Using this operator, documents can be scored in a more differentiated way. The individual results of the search are multiplied with each other, and the resulting value is divided by 100.
SUM	This operator calculates the sum of the individual scores up to a total of 100.
YESNO	This operator allows you to limit a search to only those documents matching a query, without the score of that query affecting the final scores of the documents. The operator sets an individual result to 100 if it is greater than 0, otherwise it remains 0.

5.1.6 Field Operators and Relational Operators

Field operators search document fields that have been defined in a collection. These operators perform filter functions by selecting documents whose fields have the specified values. Searching in document fields is slower than searching in zones. Normally it is not necessary to search in fields because most of the fields are present as zones as well.

Documents that are retrieved using field operators are neither relevance-ranked nor can the MANY modifier be used in conjunction with field operators.

Using the following relational operators, the values of document fields can be compared with search words: = (equal), != (not equal), > (greater than), >= (greater than or equal), < (less than), <= (less than or equal).

For text comparisons the following field operators are available:

Operator name	Description
CONTAINS	Selects documents by matching the word or phrase you specify with the values stored in a specific document field. Documents are selected only if the search elements specified appear in the same sequential and contiguous order in the field value.
MATCHES	Selects documents by matching the character string you specify with values stored in a specific document field. Documents are selected only if the search elements specified match the field value exactly. If a partial match is found, a document is not selected.
STARTS	Selects documents by matching the character string you specify with the starting characters of the values stored in a specific document field.
ENDS	Selects documents by matching the character string you specify with the ending characters of the values stored in a specific document field.

SUBSTRING	Selects documents by matching the character string you specify with a portion of the strings of the values stored in a specific document field.

5.2 Operator Reference

This section describes each search operator in detail. Where appropriate, each description includes an example of simple syntax and explicit syntax. Operators are listed alphabetically.

Please note that in environments with a locale other than English, the operators must always be preceded by a Pound sign (or hash mark) and enclosed in angle brackets (example: <#AND>).

Furthermore, if "and", "or", "any", "all" or "not" or their equivalents in another language are used as search words, then these words must be enclosed in quotation marks. Otherwise they will be interpreted as operator or modifier names.

5.2.1 ACCRUE

Selects documents that include at least one of the search elements you specify. Valid search elements are two or more words or phrases. Retrieved documents are relevance-ranked.

The ACCRUE operator scores retrieved documents according to the presence of each search element in the document using "the more, the better" approach: the more search elements found in the document, the better the document's score. Following are examples of search syntax.

To select documents containing stemmed variations of the words "computers" and "laptops", you can enter any of the following:

```
computers <ACCRUE> laptops
computers, laptops
<ACCRUE> (computers, laptops)
```

5.2.2 ALL

Selects documents that contain all of your search elements. Documents retrieved using the ALL operator are not relevance-ranked. Scores cannot be assigned to this operator.

To select documents which contain stemmed variations of the phrase "pharmaceutical companies" and stemmed variations of the word "stock", you can enter the following:

```
pharmaceutical companies ALL stock
ALL (pharmaceutical companies, stock)
```

Only those documents that contain both search elements, or stemmed variations of them (for example, "pharmaceutical company", "stocks", etc.), are retrieved. Each retrieved document is assigned a score of 100.

5.2.3 AND

Selects documents that contain all of your search elements. Documents retrieved using the `AND` operator are relevance-ranked.

To select documents which contain stemmed variations of the phrase "pharmaceutical companies" and stemmed variations of the word "stock", you can enter the following:

```
pharmaceutical companies AND stock
AND (pharmaceutical companies, stock)
```

Only those documents that contain both search elements, or stemmed variations of them (for example, "pharmaceutical company", "stocks" etc.), are retrieved. A calculated score is assigned to each retrieved document.

5.2.4 ANY

Selects documents that show evidence of at least one of your search elements. Retrieved documents are not relevance-ranked. Scores cannot be assigned to this operator.

To select documents that contain stemmed variations of the word "election" or the phrases "national elections" or "senatorial race", you can enter the following:

```
election ANY national elections ANY senatorial race
ANY (election, national elections, senatorial race)
```

Only those documents that contain at least one of the search elements, or a stemmed variation of at least one of them, are retrieved. Each retrieved document is assigned a score of 100.

5.2.5 IN

Selects documents that contain specified values in one or more document zones. A document zone represents a field in a version, for example the title or the body text (`blob`). The specified name must exactly match the zone name.

The following query expression searches document zones named "summary" for the word "safety"

```
"safety" <IN> summary
```

To search with multiple words, phrases, or topics enclose them in parentheses. The following query expression searches document zones named "summary" for the word "safety" and stemmed variations of the word "warning"

```
("safety", warning) <IN> summary
```

To search multiple zones, separate them with commas and enclose them in parentheses. The following query expression searches both the "summary" zone and the "title" zone for the word "safety" and stemmed variations of the word "warning"

```
("safety", warning) <IN> (summary, title)
```

You must enclose query expressions containing commas in parentheses. The following example searches the "summary" zone for the word "safety" and stemmed variations of the phrase "environmental regulation".

```
("safety", environmental regulation) <IN> summary
```

The following query expression searches both the "summary" zone and the "title" zone for the word "safety" and stemmed variations of the phrase "environmental regulation".

```
("safety", environmental regulation) <IN> (summary, title)
```

5.2.6 NEAR

Selects documents containing the specified search terms within close proximity to each other. Document scores are calculated based on the relative number of words between search terms. For example, if the search expression includes two words, and those words occur next to each other in a document (so that the region size is two words long), then the score assigned to that document is 100. Thus, the document with the smallest possible region containing all search terms always receives the highest score. As search terms appear further apart, the score drops toward zero. A document receives a zero score only if it does not contain all search terms.

The `NEAR` operator is similar to the other proximity operators in the sense that the search words you enter must be found within close proximity of one another. However, unlike other proximity operators, the `NEAR` operator calculates relative proximity and assigns scores based on its calculations.

To retrieve relevance-ranked documents that contain stemmed variations of the words "war" and "peace" within close proximity to each other, you can enter the following:

```
war <NEAR> peace<NEAR>(war, peace)
```

5.2.7 NEAR/N

Selects documents containing two or more words within *N* number of words of each other, where *N* is an integer. Document scores are calculated based on the relative distance of the specified words when they are separated by *N* words or less.

For example, if the search expression `NEAR/5` is used to find two words within five words of each other, a document that has the specified words within three words of each other is scored higher than a document that has the specified words within five words of each other.

The *N* variable can be an integer between 1 and 1024, where `NEAR/1` searches for two words that are next to each other. If *N* is 1000 or above, you must specify its value without commas, as in `NEAR/1000`. You can specify multiple search terms using multiple instances of `NEAR/`*N*, as long as the value of *N* is the same.

For example, to retrieve relevance-ranked documents that contain stemmed variations of the words "commute", "bicycle", "train", and "bus" within 10 words of each other, you can enter the following:

```
commute <NEAR/10> bicycle <NEAR/10> train <NEAR/10> bus
```

You can use the NEAR/*N* operator with the ORDER modifier to perform ordered proximity searches.

5.2.8 OR

Selects documents that show evidence of at least one of your search elements. Documents selected using the OR operator are relevance-ranked.

To select documents that contain stemmed variations of the word "content" or "editing" or the phrase "content management", you can enter the following:

```
content OR editing OR content management
OR (content, editing, content management)
```

Only those documents that contain at least one of the search elements, or a stemmed variation of at least one of them, are retrieved. A calculated score is assigned to each retrieved document.

5.2.9 PARAGRAPH

Selects documents that include all of the search elements you specify within a paragraph. Valid search elements are two or more words or phrases. You can specify search elements in a sequential or a random order. Documents are retrieved only if the search elements appear in the same paragraph.

To retrieve relevance-ranked documents that contain stemmed variations of the word "drug" and the phrase "cancer treatment" in the same paragraph, you can enter the following:

```
drug <PARAGRAPH> cancer treatment
<PARAGRAPH> (drug, cancer treatment)
```

To search for three or more words or phrases, you must use the PARAGRAPH operator between each word or phrase.

You can use the PARAGRAPH operator with the ORDER modifier to perform ordered proximity searches.

5.2.10 PHRASE

Selects documents that include a phrase you specify. A phrase is a grouping of two or more words that occur next to each other in a specific order.

By default, two or more words separated by a space are considered to be a phrase in simple syntax. In addition, two or more words enclosed in double quotes are considered to be a phrase. To retrieve relevance-ranked documents that contain the phrase "mission oak", you can enter any of the following:

```
mission oak
"mission oak"
mission <PHRASE> oak
<PHRASE> (mission, oak)
```

5.2.11 SENTENCE

Selects documents that include all of the words you specify within a sentence. You can specify search elements in a sequential or a random order. Documents are retrieved only if the search elements appear in the same sentence.

To retrieve relevance-ranked documents that contain stemmed variations of the words "American" and "innovation" within the same sentence, you can enter the following:

```
american <SENTENCE> innovation
<SENTENCE> (american, innovation)
```

You can use the SENTENCE operator with the ORDER modifier to perform ordered proximity searches.

5.2.12 SOUNDEX

Selects documents that include one or more words that sound like, or whose letter pattern is similar to, the word specified. Words have to start with the same letter as the word you specify to be selected.

In order to be able to search in collections using the SOUNDEX operator, all documents must have been indexed with the Soundex option. This option can be added to the WORD-IDXOPTS parameter in the style.prm file. Afterwards all documents must be reindexed.

For example, to retrieve documents containing a word that is close in structure to the word "sale", you can enter the following:

```
<SOUNDEX> sale
```

The documents retrieved will include words such as "sale", "sell", "seal", "shell", "soul", and "scale". Documents are not relevance-ranked unless the MANY modifier is used, as in:

```
<MANY><SOUNDEX> sale
```

5.2.13 STEM

Selects documents that include one or more variations of the search word you specify. For example, to retrieve documents containing a variation of the word "film", you can enter the following:

```
<STEM> film
```

The documents retrieved will include words such as "films", "filmed", and "filming". Documents are not relevance-ranked unless the MANY modifier is used, as in:

```
<MANY><STEM> film
```

5.2.14 THESAURUS

Selects documents that contain one or more synonyms of the word you specify. For example, to retrieve documents containing synonyms of the word "altitude" you can enter the following:

```
<THESAURUS> altitude
```

The documents retrieved will include words suchas "height" or "elevation." Documents are not relevance-ranked unless the MANY modifier is used, as in:

```
<MANY><THESAURUS> altitude
```

5.2.15 TOPIC

Topics are search words that the Search Cartridge expands, resulting in a search query. The generated search query combines the search expressions that previously have been defined as belonging to this topic.

In the simple parser topics are recognized even if the TOPIC operator is not specified. In the explicit parser each of the following formats can be used to specify a topic in an expression:

```
{topicname}
<#TOPIC>topicname
<#TOPIC>(topicname)
```

In the examples above, *topicname* is the name of the topic to be used in the expression.

5.2.16 TYPO/N

Selects documents that contain the word you specify plus words that are similar to the query term. The TYPO/*N* operator performs approximate pattern matching to identify similar words. This makes it ideal for use in an environment where documents have been scanned using optical character recognition (OCR).

The optional *N* variable in the operator name expresses the maximum number of errors between the query term and a matched term, a value called the error distance. If *N* is not specified, an error distance of 2 is used.

The error distance between two words is based on the calculation of errors, where an error is defined to be a character insertion, deletion, or transposition. For example, for these sets of words, the second word matches the first within an error distance of 1:

```
mouse, house (m # h)
agreed, greed (a is deleted)
cat, coat (o is inserted)
```

For the query below, documents with the words "sweeping" and "swimming" will match, since there are 3 transpositions in the word (e ? i, e ? m, p ? m).

```
<TYPO/3> sweeping
```

Both of the queries below will return the same results. Documents containing the words "swept" and "kept" will match, since the "kept" word contains 1 transposition and 1 deletion.

```
<TYPO/2> swept
<TYPO> swept
```

The TYPO/*N* operator must scan the collection's word list in order to find candidate matching words. This makes it impractical for use in large collections (greater than 100,000 documents unless a current spanning word list is available) or in performance-sensitive environments. Performance can be improved by generating a spanning word list for the collections to be used.

Please note these limitations: A query term specified with TYPO/*N* can have a maximum length of 32 characters. Also, TYPO/*N* is not supported with multi-byte character sets.

5.2.17 WILDCARD

Selects documents that contain matches to a wildcard character string. The WILDCARD operator lets you define a wildcard string, which can be used to locate related word matches in documents. A wildcard string consists of special characters.

For example, to retrieve documents that contain words such as "pharmaceutical", "pharmacology", and "pharmacodynamics" you can enter the following:

```
pharmac*
```

Documents are not relevance-ranked unless the MANY modifier is used, as in:

```
<MANY> pharmac*
```

The wildcard characters "*" and "?" automatically enable wildcard searching. To use other constructs, use the WILDCARD operator explicitly with any of the characters below.

Character	Function
?	Specifies one of any alphanumeric character, as in ?an, which locates "ran", "pan", "can", and "ban". It is not necessary to specify the WILDCARD operator if you use the question mark. The question mark is ignored in a set ([ ]) or in an alternative pattern ({ }).
*	Specifies zero or more of any alphanumeric character, as in corp*, which locates "corporate", "corporation", "corporal", and "corpulent". It is not necessary to specify the WILDCARD operator when you use the asterisk; you should not use the asterisk to specify the first character of a wildcard string. The asterisk is ignored in a set ([ ]) or in an alternative pattern ({ }).
[ ]	Specifies one of any character in a set, as in <WILDCARD> 'c[auo]t', which locates "cat",

	"cut", and "cot". You must enclose the word that includes a set in backquotes, and there can be no spaces in a set.
{ }	Specifies one of each pattern separated by a comma, as in `<WILDCARD> 'bank{s,er,ing}'`, which locates "banks", "banker", and "banking". You must enclose the word that includes a pattern in backquotes, and there can be no spaces in a set.
^	Specifies one of any character not in the set, as in `<WILDCARD> 'st[^oa]ck'`, which excludes "stock" and "stack" but locates "stick" and "stuck". The caret (^) must be the first character after the left bracket ([) that introduces a set.
-	Specifies a range of characters in a set, as in `<WILDCARD> 'c[a-r]t'`, which locates every three-letter word from "cat" to "crt".

Searching for Non-Alphanumeric Characters

Remember that you can search for non-alphanumeric characters only if the `style.lex` file used to create the collections you are searching is set up to recognize the characters you want to search for. Please consult your collection administrator for more information.

Searching for Wildcard Characters as Literals

Provided the `style.lex` file is set up for the collections to be searched, you can search for a word containing a wildcard character such as "/" or "*" by preceding the wildcard character with a backslash. For example, if you enter the following search string:

```
abc\*d
```

the engine finds five-character words matching the "abc*d" string. When you want to match a literal backslash, enter two backslashes.

Searching for Special Characters as Literals

The following non-alphanumeric characters perform special, internal search engine functions, and by default are not treated as literals in a wildcard string:

- comma ,
- left and right parentheses ( )
- double quotation mark "
- backslash \
- at sign @
- left curly brace {
- left bracket [
- less than sign <
- backquote `

To interpret special characters as literals, the whole wildcard string needs to be enclosed in backquotes. For example, to search for the wildcard string "a{b", you enclose the string with backquotes, as follows:

```
<WILDCARD> `a{b`
```

To search for a wildcard string that includes the literal backquote character, you need to use two backquotes together and enclose the whole wildcard string with backquotes, as follows:

```
<WILDCARD> `*n``t`
```

You can search for backquotes only if the `style.lex` file used to create the collections you are searching is set up to recognize the backquote character.

5.2.18 WORD

Selects documents that include one or more instances of a word you specify. For example, to search for documents that contain the word "rhetoric", without also considering the words "rhetorical" and "rhetorician," you can enter the following:

```
<WORD> rhetoric
```

Documents are not relevance-ranked unless the MANY modifier is used, as in:

```
<MANY><WORD> rhetoric
```

5.3 Overview of Special Operators

The documents in this section describe the special operators of the Search Cartridge. Where appropriate, each description includes an example of simple syntax and explicit syntax. Operators are listed alphabetically. The relational operators are listed at the end of this section.

Please note that in environments with a locale other than English, the operators must always be preceded by a Pound sign (or hash mark) and enclosed in angle brackets (example: <#AND>).

Furthermore, if *and*, *or*, *any*, *all* or *not* or their equivalents in another language are used as search words, then these words must be enclosed in quotation marks. Otherwise they will be interpreted as operator or modifier names.

5.3.1 COMPLEMENT

Calculates scores for documents matching a query by taking the complement (subtracting from 1) the scores for the query's search elements. Following is an example of search syntax:

```
<COMPLEMENT> ("computers")
```

The COMPLEMENT operator is a unary operator. It multiplies search elements as specified. The elements are combined, using the ACCRUE operator by default, to generate a single score which is then complemented. A sample query expression with two search elements is below:

```
<COMPLEMENT> ("computers","laptops")
```

In the above example, the query is evaluated as the word "computers" accrued using the ACCRUE operator with the word "laptops". The COMPLEMENT operator is applied to the result.

5.3.2 CONTAINS

Selects documents by matching the word or phrase you specify with values stored in a specific document field. Documents are selected only if the search elements specified appear in the same sequential and contiguous order in the field value.

When you use the CONTAINS operator, you specify the field name to search, and the word or phrase to search for.

With the CONTAINS operator, the words stored in a document field are interpreted as individual, sequential units. You can specify one or more of these units as search criteria. To specify multiple words, each word must be sequential and contiguous, and must be separated by a blank space.

For example, the following title contains four sequential words:

```
Last Exit to Brooklyn
```

1. Last
2. Exit
3. to
4. Brooklyn

The following examples demonstrate how you can use the CONTAINS operator with sequential, contiguous words to match the document title listed above, assuming it is stored in a title field:

```
TITLE <CONTAINS> Last Exit
TITLE <CONTAINS> to Brooklyn
```

The following examples show how you can use a question mark (?) to represent individual variable characters of a word, and an asterisk (*) to match multiple variable characters of a word:

```
TITLE <CONTAINS> La?? Exit
TITLE <CONTAINS> Exit to Br*
```

Question marks and asterisks can be used to represent characters that are part of a word but not white space that appears between words.

The CONTAINS operator does not recognize non-alphanumeric characters. The CONTAINS operator interprets non-alphanumeric characters as spaces and treats the separated parts as individual units.

For example, if you have defined a dash (-) as a valid character, and you enter search criteria that include this character, as in "on-line", this results in two individual units, as follows:

```
TITLE <CONTAINS> on line
```

5.3.3 ENDS

Selects documents by matching the character string you specify with the ending characters of the values stored in a specific document field. For example, assume a document field named `AUTHOR` has been defined. To select documents written by Milner, Wagner, and Faulkner, you can enter the following:

```
AUTHOR <ENDS> ner
```

5.3.4 = (equal)

Selects documents whose document fields contain exactly the same values as the specified string that is being searched for. For example, you can search for documents of the `document` type as follows:

```
objType = document
```

5.3.5 FREETEXT

This operator is meant to be used in application development environments. It interprets text using the free text query parser and scores documents using the resulting query expression. All retrieved documents are relevance-ranked.

When search expressions are analyzed, unimportant words (so-called stop words) such as articles, conjunctions, and prepositions are removed. Characteristics of natural language, like noun phrases and word order, are taken into account as the resulting query is constructed. The retrieved documents are relevance-ranked. For example:

```
<FREETEXT> ("peace negotiations in the Middle East")
```

This query could be interpreted as follows:

```
<#ACCRUE> (<#PHRASE> "peace negotiations", <#PHRASE> "Middle East")
```

The `FREETEXT` operator makes the functionality of the free text parser available and additionally permits you to combine free text search queries with other search criteria, giving you full access to the query language. Example:

```
<#ACCRUE> (<#PHRASE> "peace negotiations", <#PHRASE> "Middle East") <#AND> (DATE >
 20101224000000)
```

The quotation marks are required. If you want to include embedded quotes, they must be preceded with backslashes, as:

```
<FREETEXT> ( "\"Independence Day\""), ("\"The Arrival\""), science fiction" )
```

Please note: In the case where a query or document contains only words defined as stop words in the collection `style.stp` file(s), the free text query parser uses the stop words for the query, ignoring the stop words list.

The `FREETEXT` operator can be combined with other operators in the same way as the `ACCRUE` operator.

5.3.6 > (greater than)

Selects documents whose document fields contain values that are greater than the specified string that is being searched for. This makes it possible, for example, to search for documents whose date of last change is after a particular date.

```
lastChanged > 20101223000000
```

5.3.7 >= (greater than or equal)

Selects documents whose document fields contain values that are greater than or equal to the specified string that is being searched for. This makes it possible, for example, to search for documents whose date of last change is on or after a particular date. Example:

```
lastChanged >= 20101223000000
```

5.3.8 < (less than)

Selects documents whose document fields contain values that are less than the specified string that is being searched for. This makes it possible, for example, to search for documents whose date of last change is before a particular date.

```
lastChanged < 20101223000000
```

5.3.9 <= (less than or equal)

Selects documents whose document fields contain values that are less than or equal to the specified string that is being searched for. The following example returns documents that were last changed on or before December 23rd, 2010:

```
lastChanged <= 20101223000000
```

5.3.10 LIKE

This operator is meant to be used in application development environments. It searches for documents that are similar to the sample one or more documents or text passages you provide. The search engine analyzes the provided text to find the most important terms to use for the search. If multiple samples are provided, the search engine assumes that all of the samples are about a single theme and selects important terms common across the samples. Retrieved documents are relevance-ranked.

The `LIKE` operator accepts a single operand, called the QBE (query-by-example) specification. The QBE specification can be either the literal text of the example to query on, or it can be a specification of one or more full documents and text passages to use as positive and negative examples.

document contains only words defined as stop words in the collections *style.stp* file(s), a QBE query with the `LIKE` operator returns no results.</p>

Syntax

Document specification is made with a series of text references enclosed in braces. The syntax for specifying references is:

```
{[name=]type:value [name=]type:value ...}
```

where:

- *`name`* is either `posex` ("positive example"), or `negex` ("negative example"). A negative example reduces the weights of terms when they occur in a positive example. If terms from a negative example do not exist within the positive example, the negative example has no effect. (Hence a `negex` by itself makes no sense.) The variable *`name`* is optional. If not specified, *`name`* is set internally to `posex`. In this case the equal sign must neither be present.
- *`type`* can be one of the following:
  - `VdkVgwKey`, to specify a document by its external ID, i. e. the content ID in the Content Manager or the object ID in the Template Engine.
  - `Text`, to specify the text directly
- *`value`* is a reference to a piece of text to use as the positive or negative example. The value of *`value`* depends on *`type`*.
  - `VdkVgwKey`: the document ID (i. e. content or object ID)
  - `Text`: Literal text.

If there is no explicit type specifier, *`value`* is interpreted in the following ways:

- Literal text if it starts with a quotation mark
- `VdkVgwKey` for all other cases

The `LIKE` operator can be combined with other operators using the same rules as for the `ACCRUE` operator.

Examples

The following examples illustrate uses of the `LIKE` operator. Integer numbers always represent a content or an object ID.

Just literal text:

```
<LIKE> ("The dog ate the shoe.")
```

Explicit specification of a single positive example:

```
<LIKE> ( "{posex=vdkvgwkey:650431}" )
```

Explicit specification of multiple positive and negative examples:

```
<LIKE> ( "{posex=vdkdocid:7369 posex=vdkvgwkey:8457
negex=text:"stock market"}" )
```

Same as the preceding but with implied reference types:

```
<LIKE> ( "{posex=#7369 posex=8457 negex=\"stock market\"}" )
```

Similar to the preceding but with implied `posex` names:

```
<LIKE> ( "{vdkdocid:7369 vdkvgwkey:8457}" )
```

Same as the preceding, but using the most implicit syntax:

```
<LIKE> ( "{#7369 8457}" )
```

You can combine a text reference list with literal text:

```
<LIKE> ( "{#7369 8457} And more text" )
```

The preceding QBE specification is equivalent to this:

```
<LIKE> ( "{#7369 8457 text: \"And more text\"}" )
```

The simplest way of specifying a single positive example by `VgwKey`:

```
<LIKE> ( "{650431}" )
```

The example is in the file `doc.txt`, starting at the 100th byte:

```
<LIKE> ( "{posex=file:doc.txt:100:200}" )
```

Quotation marks embedded in `LIKE` expressions must be preceded by backslashes. The backslash indicates to the engine that the following character is supposed to be treated as a literal character.

Efficiency Considerations

In order to process a `LIKE` expression, the search engine must analyze the full text of the examples in the QBE specification. This may be time consuming, especially if the example documents are large or require extensive filtering.

5.3.11 MATCHES

Selects documents by matching the character string you specify with values stored in a specific document field. Documents are selected only if the search elements specified match the field value exactly.

You can use question marks (?) to represent individual variable characters within a string, and asterisks (*) to match multiple characters within a string.

For example, assume a document field named SOURCE includes the following values:

```
COMPUTER
COMPUTERWORLD
COMPUTER CURRENTS
PC COMPUTING
```

To locate documents whose source is COMPUTER, the MATCHES operator is used as follows:

```
SOURCE <MATCHES> computer
```

Here, the MATCHES operator matches COMPUTER, but not COMPUTERWORLD, COMPUTER CURRENTS, or PC COMPUTING.

To locate documents whose source is COMPUTERWORLD, the MATCHES operator is used as follows:

```
SOURCE <MATCHES> computer?????
```

Now, the MATCHES operator matches COMPUTERWORLD, since each question mark (?) represents specific character positions within the string. COMPUTER and COMPUTER CURRENTS are not matched, because their character strings do not match the length represented by the specific character positions.

To locate documents whose sources are COMPUTER, COMPUTERWORLD, and COMPUTER CURRENTS, the MATCHES operator is used as follows:

```
SOURCE <MATCHES> computer*
```

Here, the MATCHES operator matches COMPUTER, COMPUTERWORLD, and COMPUTER CURRENTS, since the asterisk (*) represents zero or more variable characters at the end of the string.

To locate documents whose sources include COMPUTER, COMPUTERWORLD, COMPUTER CURRENTS, and PC COMPUTING, the MATCHES operator can be used as follows:

```
SOURCE <MATCHES> *comput*
```

Now, the MATCHES operator matches all four occurrences, since the asterisk (*) represents a character string of any length.

5.3.12 != (not equal)

Selects documents whose document fields do not contain the same values as the specified string that is being searched for. For example, you can search for documents that are not of the `document` type as follows:

```
objType != document
```

By means of the `NOT` modifier the same search can be performed using a test for equality:

```
<#NOT>(objType=document)
```

Although both queries return the same results, the first query expression is much more efficient than the second.

5.3.13 PRODUCT

This operator calculates scores for documents matching a query by multiplying the scores for the query's individual search elements. The result is then divided by 100.

Following is an example of the search syntax:

```
<PRODUCT> ("computers","laptops")
```

If a search on "computers" generated a score of 50 and a search on "laptops" generated a score of 20, the preceding search would produce a score of 10 (the product of both values, divided by 100).

5.3.14 STARTS

Selects documents by matching the character string you specify with the starting characters of the values stored in a specific document field. For example, you can search for documents, modified last on December 23, 2010 by searching for "20101223" in the *lastChanged* field. Since the values in this field include the date and the time, the field values must be matched with `STARTS`:

```
lastChanged <#STARTS> 20101223
```

Date values are indexed by the Content Manager and the Template Engine as 14-digit numbers with the following format: year (4 digits), month, day, hour, minute, second (two digits each).

5.3.15 SUBSTRING

Selects documents by matching the character string you specify with a portion of the strings of the values stored in a specific document field. The characters that comprise the string can occur at the beginning of a field value, within a field value, or at the end of a field value.

For example, to retrieve documents whose titles contain words such as "solution", "resolution", "solve", and "resolve", you can enter the following:

```
TITLE <SUBSTRING> sol
```

5.3.16 SUM

Calculates scores for documents matching a query by adding together, to a maximum of 100, the scores for the query's search elements. Following is an example query expression:

```
<SUM> ("computers","laptops")
```

If a search on "computers" generated a score of 50 and a search on "laptops" generated a score of 20, the preceding search would produce a score of 70. If a search on "computers" generated a score of 50 and a search on "laptops" generated a score of 75, the preceding search would produce a score of 100 (the maximum).

5.3.17 YESNO

Forces the score of an element to 100, if the element's score is nonzero. Examples help clarify this.

```
<YesNo> ("Chloe")
```

If the retrieval result of the search on "Chloe" was 75, with the `YesNo` operator, the result would be 100; if the retrieval result is 0, it remains 0.

This operator allows you to limit a search to only those documents matching a query, without the score of that query affecting the final scores of the documents. For example, to search among documents that contain "Chloe", with "Mead" as the determinant for ranking, you cannot simply specify the following:

```
"Chloe" <AND> "Mead"
```

because that would produce documents ranked with scores combined from both elements. The following would do what you want:

```
<YesNo> ("Chloe") <AND> "Mead"
```

If the retrieval result of the search on "Chloe" was 50 and that on "Mead" was 75, without the `YesNo` operator, the combined result would be 50; with the operator, however, it is 75, because the score of `AND` is calculated to be the minimum score of all its search elements.

5.4 Modifier Reference

Modifiers are used in conjunction with operators. A modifier changes the standard behavior of an operator. For example, you can use the `CASE` modifier with an operator to specify that the case of the search word you enter be considered a search element as well. Modifiers include `CASE`, `MANY`, `NOT`, and `ORDER`.

There are two syntax formats used to specify modifiers with operators. Using the first format, you specify the modifier name before the operator name, as shown in the table below. Please note that most of the modifiers can only be used with particular operators.

Modifier	Valid Operators	Examples
CASE	WORD WILDCARD	<CASE><WORD> iMac
MANY	WORD WILDCARD STEM SOUNDEX PHRASE SENTENCE PARAGRAPH	<MANY><WORD> virtual
NOT	all operators	cat <AND> dog <AND> <NOT> pet
ORDER	PARAGRAPH SENTENCE NEAR/N ALL	president <ORDER> <PARAGRAPH> washington <ORDER> <SENTENCE> ("president", "washington")

Using the second syntax format, you specify the modifier name with the operator name as follows: <OpName/ModName>. This second syntax is valid only for the CASE and NOT modifiers.

Modifier	Valid Operators	Examples
CASE	WORD WILDCARD CONTAINS MATCHES STARTS ENDS SUBSTRING	author <CONTAINS/CASE>Don
NOT	all operators	author<CONTAINS/NOT>don author<STARTS/NOT>xxx

5.4.1 CASE

Use the CASE modifier with the WORD or WILDCARD operator to perform a case-sensitive search, based on the case of the word or phrase specified. The modifier only needs to be specified if search words exclusively containing either uppercase or lowercase letters need to be found in the documents. When mixed uppercase and lowercase characters are included in a query, the search engine finds case-sensitive matches automatically.

To use the CASE modifier, you simply enter the search word or phrase as you wish it to appear in retrieved documents -- in all uppercase letters, in mixed uppercase and lowercase letters, or in all lowercase letters.

For example, to retrieve documents that contain the word "apple", all in lowercase letters, you can enter the following:

```
<CASE> <WORD> apple
```

Only those documents that contain the word "apple" will be selected. Occurrences of "Apple", "apples", or "APPLE" will not be selected.

5.4.2 MANY

Counts the density of words, stemmed variations, or phrases in a document, and produces a relevance-ranked score for retrieved documents. The more occurrences of a word, stem, or phrase proportional to the amount of document text, the higher the score of that document when retrieved. Because the MANY modifier considers density in proportion to document text, a longer document that contains more occurrences of a word can score lower than a shorter document that contains fewer occurrences. You can use the MANY modifier with these operators: WORD, WILDCARD, STEM, SOUNDEX, PHRASE, SENTENCE, PARAGRAPH.

For example, to select documents based on the density of stemmed variations of the word "apple", you can enter the following:

```
<MANY> <STEM> apple
```

To select documents based on the density of the phrase "mission oak", you can enter the following:

```
<MANY> mission oak
```

The MANY modifier cannot be used with AND, OR, ACCRUE, or relational operators.

5.4.3 NOT

Use the NOT modifier with a word or phrase to exclude documents that show evidence of that word or phrase. For example, to select only documents that contain the words "cat" and "mouse" but not the word "dog", you can enter the following:

```
cat <AND> mouse <AND> <NOT> dog
```

5.4.4 ORDER

Use the ORDER modifier to specify that search elements must occur in the same order in which they were specified in the query. If search values do not occur in the specified order in a document, the document is not selected. You can use the ORDER modifier with these operators: PARAGRAPH, SENTENCE, NEAR/*N*, and ALL.

Always place the ORDER modifier just before the operator. The following syntax examples show how you can use either simple syntax or explicit syntax to retrieve documents containing the word "president" followed by the word "washington" in the same paragraph:

Simple syntax:

```
president <ORDER><PARAGRAPH> washington
```

Explicit syntax:

```
<ORDER><PARAGRAPH> ("president", "washington")
```

To search for documents containing the words "diver", "kills", "shark" in that order within 20 words of each other, use one of the following queries:

```
diver <ORDER><NEAR/20> kills <ORDER> <NEAR/20> shark
<ORDER> <NEAR/20> (diver, kills, shark)
```

You can use the NEAR/*N* operator with the ORDER modifier to duplicate the behavior of the PHRASE operator. For example, to search for documents containing the phrase "world wide web", you can use the following syntax:

```
world <ORDER><NEAR/1> wide <ORDER><NEAR/1> web
```

To search for a word between two other words, you can use the ORDER modifier with the ALL operator, like this:

```
<ORDER><ALL>(dog, cat, squirrel)
```

The above query searches for "cat" between "dog" and "squirrel". Stemmed variations of the words will match the query.

The between query can be extended to include subquery expressions. For example:

```
<ORDER><ALL> (dog, fat cat, squirrel)
```

The above query searches for thephrase "fat cat" between the words "dog" and "squirrel". Again, stemmed variations of the words are considered a match.

5.5 Ranking the Search Results

The search results of a query to the Autonomy search engine can be sorted by one or more fields. The most common sorting is by score. The score is a value calculated by the search engine which expresses the relevance of this search result (hit) in relation to the search query.

The score for a particular document is the result of assigning a score to each search term (the exact algorithms for this are unknown) and then combining these partial scores. How they are combined depends on the search operator used:

- AND – minimum of the partial scores.
- OR – maximum of the partial scores.
- YESNO – a positive partial score results in 1, 0 remains 0.
- ANY – like YESNO(OR ...)
- [*xx*] – multiplies the partial score with 0.*xx*.

Examples

Assume the following partial scores for a search result:

- `"Teddy"` returns 0.4
- `"Bär"` returns 0.8
- `"admins" <IN> permissionLiveServerRead` returns 0.77
- `"free" <IN> noPermissionLiveServerRead` returns 0

Applying the rules for combination mentioned above yields the following results:

- `"Teddy" <OR> "Bär"` yields max(0.4, 0.8) -> 0.8
- `"Teddy" <AND> "Bär"` yields min(0.4, 0.8) -> 0.4
- `[90]"Teddy" <#OR> [10]"Bär"` yields max(0.36, 0.08) -> 0.36
- `<#ANY> (("admins" <#IN> permissionLiveServerRead), ("free" <#IN> noPermissionLiveServerRead))` yields `YESNO`(max(0.77, 0) -> 1

Further details on calculating scores can be found in the documentation for the individual operators and modifiers. In particular, see MANY and ACCRUE.

6

6 MISE, the Search Engine Server's XML Protocol

6.1 Payloads

The XML documents exchanged by CMS components via the XML Interface are called *Payloads*. The `ses-payload` element is the root element of all request and response documents:

```
<!ELEMENT ses-payload (ses-header, (ses-response+ | ses-request+))>
<!ATTLIST ses-payload
  payload-id CDATA #REQUIRED
  timestamp CDATA #REQUIRED
  version CDATA #REQUIRED
>
```

The attributes of `ses-payload` elements have the following meaning:

- `payload-id`
  Payload ID. This ID is generated by the creator of the payload and must be unique within a communication context. Such a context is formed by the Content Management Server (e.g. of a company) and all the clients that communicate with the server. As a rule, the `payload-id` is generated by an algorithm.
- `timestamp`
  Date and time (timestamp) of payload creation. The timestamp must be specified in canonical form as a 14-digit string (beginning on the left: year 4-digit, month 2-digit, day 2-digit, hour 2-digit, minutes 2-digit, seconds 2-digit) in GMT (Example: 20110716020223).
- `version`
  Version of the XML Interface protocol. The structure of the payload depends on the version. At the time of writing this manual, the XML Interface protocol has the version number 2.1.

In a request a client indicates which version of the XML Interface protocol it is using by specifying a value for `version`.

The Search Engine Server supports – in addition to the current version of the protocol – all versions which were previously valid. If the client is using one of these versions, the server creates a response payload in this version. Otherwise, it responds with an error message which indicates the protocol incompatibility. The server creates this message in its current version of the XML Interface protocol.

6.1.1 Header Element

The first subelement of all payloads is `ses-header`. The `ses-header` element contains information about the identity of the two parties that exchange the payload.

```
<!ELEMENT ses-header (ses-sender, ses-receiver?, ses-authentication?)>
<!ELEMENT ses-sender EMPTY>
<!ATTLIST ses-sender
  sender-id CDATA #REQUIRED
  name CDATA #REQUIRED>
<!ELEMENT ses-receiver EMPTY>
<!ATTLIST ses-receiver
  receiver-id CDATA #REQUIRED
  name CDATA #REQUIRED>
<!ELEMENT ses-authentication EMPTY>
<!ATTLIST ses-authentication
  login CDATA #REQUIRED
  password CDATA #REQUIRED>
```

ses-sender

The `ses-sender` element must always appear as the subelement of the `ses-header` element. It specifies the identity of the payload sender. The attributes of the `ses-sender` elements have the following meaning:

- `sender-id`
  Payload sender ID. During the installation, each CMS server application and the clients are allocated an ID which is unique within the communication context. This ID is used in the creation of the payload as the `sender-id`.
- `name`
  Payload sender name. The name is a string which identifies the application which sends the payload. The Search Engine Server uses *SES* as the name.

ses-receiver

The `ses-receiver` element specifies the identity of the desired receiver of the payload. This element is optional as long as there is an individual network connection between the server and the client. In this case, the sender of the payload and the receiver are uniquely identified. If, however, between the server and the client there is a proxy server which serves several clients or servers, both the server and the client can use the `ses-receiver` element to inform the proxy server of the receiver. The attributes of the `ses-receiver` elements have the following meaning:

- `receiver-id`
  Payload receiver ID. This attribute has the same semantics as the `sender-id` attribute of the `ses-sender` element. It contains the ID of the NPS server or NPS client which is to receive the payload. The ID is unique within a communication context. For response payloads, this attribute is always a copy of the `sender-id` attribute of the `ses-sender` element in the corresponding request payload.
- `name`
  Name of the receiver. The name is a string which identifies the desired receiver application. For response payloads, this attribute is a copy of the `name` attribute of the `ses-sender` element in the corresponding request payload.

ses-authentication

The `ses-authentication` element is optional. It can only be used in request payloads. It contains information about the user for whom the request is to be processed. The attributes of the `ses-authentication` element have the following meaning:

- `login`
  The login of the user of the Search Engine Server.

- `password`
  The password of the user (clear text).

6.1.2 Request Element

For request payloads, one or more `ses-request` elements follow the `ses-header` element in the `ses-payload` root element. These elements specify the operations which are to be executed by the CMS Server.

```
<!ELEMENT ses-request (ses-indexDoc|ses-deleteDoc|ses-search)>
<!ATTLIST ses-request
  request-id CDATA #REQUIRED
  preclusive (true | false) "false">
```

The subelements of the `ses-request` element are explained in the following sections. An `ses-request` element has the following attributes:

- `request-id`
  Request ID. This ID is issued by the creator of the request payload. It must be unique within all payloads that are exchanged in a particular communication context. It is not sufficient that the ID is unique within the current payload.
- `preclusive`
  Truth value (`true` or `false`). The `preclusive` attribute allows the NPS client to mark the requests which are critical for the continued processing of payloads. If the processing of a request marked as `preclusive` fails, all further requests in the payload are not processed but are answered with an error message.

All requests in a payload are processed in sequence beginning with the first request. In this way, it is possible for a client to put together requests that are dependent on each other in one request payload. For example, in a single payload, a client can first create an object class and then create objects based on this class. Using the `preclusive` attribute, the client can also ensure that the dependent request is only processed when the previous one has not caused an error.

6.1.3 Response Element

For response payloads, the `ses-payload` root element contains one or more `ses-response` elements after the `ses-header` element with which each result of the operations performed by the server is returned.

If a request payload has been completely recognized and processed by the server, each `ses-response` element in the response payload corresponds to a `ses-request` element in the request payload. In this case, the `ses-response` elements contain messages which refer to the contents of the requests (Request Level Message).

If, on the other hand, the Search Engine Server receives an invalid request payload (e.g. without a `ses-header` element or with unrecognizable `ses-request` elements), it returns a response payload containing only one `ses-response` element with which the general error is reported (see Payload Errors). In this case, the `ses-response` element refers to the payload (Payload Level Message). A `ses-response` element is constructed as follows.

```
<!ELEMENT ses-response (ses-code*)>
<!ATTLIST ses-response
  response-id CDATA #REQUIRED
```

```
  payload-id CDATA #IMPLIED
  request-id CDATA #IMPLIED
  success (true | false) #REQUIRED>
```

The individual responses to the requests are returned in `ses-code` elements.

```
<!ELEMENT ses-code ANY>
<!ATTLIST ses-code numeric CDATA #REQUIRED phrase CDATA #REQUIRED>
```

The attributes of the `ses-response` element have the following meanings:

- `response-id`
  Response ID. This ID is set by the creator of the response payload. It must be unique within all payloads that are exchanged in a particular communication context.
- `payload-id`
  Request payload ID. It corresponds to the `payload-id` attribute of the request payload. This attribute is added by the server only when the `ses-response` element contains a payload level message in the first `ses-code` element. A client can recognize by the occurrence of this attribute whether its request payload could be interpreted as such by the server (independently of the requests contained in it).
- `request-id`
  Request ID. It corresponds to the `receiver-id` attribute of the `ses-request` element in the request payload. This attribute is only present when the `ses-code` element contains a request level message.
- `success`
  The value of this attribute is `true` when the request could be successfully processed. Otherwise, it is `false`.

The results of the operations performed are returned by means of `ses-code` elements within the `ses-response` element. The `ses-code` elements contain a success or error message and other XML elements which represent the result of the operation or, if necessary, error information.

The attributes of the `ses-code` element have the following meanings:

- `numeric`
  Error number (or success message number). The messages corresponding to the numbers are described in section Error Handling.
- `phrase`
  The description of the error.

The content of the `ses-code` element depends on the operation indicated in the request. In operations which could not be performed successfully, the content of the `ses-code` element depends on the error which occurred. The possible contents of the `ses-code` element are listed for each operation in the following sections.

6.2 Indexing Requests

6.2.1 Request

An indexing request is coded using an `ses-indexDoc` element inside an `ses-request`. This is shown in the following example. It also shows that several requests can be placed into one payload:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ses-payload SYSTEM "http://www.infopark.com/ses.dtd">
  <ses-payload payload-id="B42TE241" timestamp="20100825172100" version="2.1">
  <ses-header>
    <ses-sender sender-id="FX45RTDT" name="CM-Server"/>
    <ses-authentication login="cm-server" password=""/>
  </ses-header>
  <ses-request request-id="BR12TI5X">
    <ses-indexDoc docId="4712" collection="collection1"
         mimeType="application/ms-word" usesStreaming="YES">
      <title encoding="plain">testdoc1</title>
      <customAttribute encoding="base64">MGHX2c5=</customAttribute>
      <blob encoding="stream">d--1157180779-000000001-X</blob>
    </ses-indexDoc>
  </ses-request>
  <ses-request request-id="BR12TI5Y">
    <ses-indexDoc docId="4713" collection="collection2">
      <title encoding="plain">testdoc2</title>
      ...
      <blob>Just the blob</blob>
    </ses-indexDoc>
  </ses-request></ses-payload>
```

The `ses-indexDoc` element has the following attributes:

- `docId`
  The ID of the document to index. Usually this is the content ID (Content Management Server) or the object ID (Template Engine).
- `collection`
  The name of the collection to which the document to be indexed is to be added.
- `mimeType`
  The MIME type of the document to be indexed. The Search Engine Server uses the MIME type to determine the preprocessor with which the document is to be preprocessed (see Configuring the Search Engine Server).
- `usesStreaming`
  YES , if at least one of the attributes to be indexed was transferred to the Search Engine Server via the Streaming-Interface. Otherwise `NO`.

The `ses-indexDoc` element contains as subelements all the attributes listed in section Content Indexing. Of these attributes only `title`, the custom attribute `customAttribute`, and `blob` were used in the example above. The encoding of the contents of the object and content attributes to be indexed is specified using the `encoding` tag attribute in the attribute tags concerned. `encoding` can have one of the following values:

- `plain`
  The value of the attribute to index is not encoded. It is included directly as the value of the element. This is the default.
- `base64`
  The attribute value is base64-encoded.
- `stream`
  The attribute value is a streaming ticket. The ticket refers to a content that the client has already transferred to the Search Engine Server using the streaming interface (see the explanation below).

If an attribute is base64-encoded or has been transferred to the Search Engine Server via the streaming interface, a preprocessor must have been configured for the MIME type of the document. This preprocessor's task is to convert the attribute's content to plain text and to set the value of `encoding` to `plain`.

6.2.2 Streaming

A client has the possibility to send the contents of attributes to the Search Engine Server in advance, i. e. prior to sending it an indexing request. This procedure is recommendable for large amounts of binary data because it is faster than base64-encoding the data and including it in the request.

A client uses the so-called streaming interface to transfer such data to the Search Engine Server. The streaming interface is addressed by sending a POST request to the HTTP port of the Search Engine Server, specifying `/stream` as URL. After the data have been transferred, the client receives a streaming ticket in the response. In the indexing request that follows, the client specifies the ticket ID in the manner described above in order to refer to the data.

The Content Management Server transferrs the contents of generic documents to the Search Engine Server via the streaming interface. This also applies to the body of `publication`, `document`, and `template` objects, if the body is larger than 8 kilobytes. Except for templates, this is also true for the Template Engine (the Template Engine does not send templates to the Search Engine Server for indexing). The minimum amount of data to be transferred via streaming can be configured in the system configuration of the Content Manager and the Template Engine using the `minStreamingDataLength` entry.

6.2.3 Response

The Search Engine Server uses an empty `ses-code` element in its response to indicate that an indexing request was processed successfully. The attributes of this element are described in section Response Element. Here is an example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ses-payload SYSTEM "http://www.infopark.com/ses.dtd">
<ses-payload payload-id="d--1950259307-000000002-X" timestamp="20101017150013"
 version="2.1">
  <ses-header>
    <ses-sender sender-id="SES-Infopark-DEV-0" name="SES"/>
    <ses-receiver name="CM Server" receiver-id="CM-Infopark-DEV-0"/>
  </ses-header>
  <ses-response response-id="0"
      request-id="d--1949717044-000000002-X" success="true">
    <ses-code phrase="OK" numeric="200"/>
  </ses-response>
</ses-payload>
```

6.3 Search Requests

6.3.1 Request

Search requests are formed using the `ses-search` element in an `ses-request` element. Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ses-payload SYSTEM "http://www.infopark.com/ses.dtd">
<ses-payload payload-id="B42TE241" timestamp="20100825172100" ses.version="2.0">
  <ses-header>
    <ses-sender sender-id="FX45RTDT" name="CM-Server"/>
    <ses-authentication login="cm-server" password=""/>
  </ses-header>
  <ses-request request-id="BR12TI5X">
    <ses-search>
```

```
      <query parser="simple">www &lt;#AND&gt; business</query>
      <minRelevance>50</minRelevance>
      <maxDocs>200</maxDocs>
      <offset>
        <start>61</start>
        <length>20</length>
      </offset>
      <searchBase>
        <query parser="simple">title &lt;#CONTAINS&gt; business</query>
        <collection>collection1</collection>
        <collection>collection2</collection>
      </searchBase>
      <sortOrder>
        <sortField direction="desc">field23</sortField>
        ...
      </sortOrder>
      <resultRecord>
        <resultField format="ISO" timezone="MEZ"
             formatter="formatterAliasName">field1</resultField>
        ...
      </resultRecord>
      <searchDirection start="newest" />
    </ses-search>
  </ses-request>
</ses-payload>
```

In the following, the subelements of the `ses-search` element are described. If left out in the request, the defaults specified here, such as `500` for `maxDocs` (the maximum number of documents in the search result), are filled in by the Search Engine Server, before the request is passed to the search engine module. This is only the case if the Verity search engine module is used. The preprocessor, however, always receives the original request document to which no default values have been added.

- `query`
  This element is optional. It has the optional `parser` attribute whose value can be `simple` (the default), `explicit`, or `freetext`. If the element is not specified, all indexed documents of the specified collection are returned. The content of the element is the search query, stated in the search processor's syntax.
- `minRelevance`
  This optional element is used to specify the minimum relevance of the documents to be included in the search result. Valid values are integer numbers in the range from 0 to 100 (inclusive), with 0 indicating the least and 100 the most relevant documents. The default value is 0.
- `maxDocs`
  This element is optional. Its content specifies the maximum number of documents to be included in the results list. Valid values are positive integer numbers and the character string `unlimited`. `unlimited` means that as many hits are to be included in the results list as the platform supports.
- `offset`
  This optional element allows you to define the subset of the search result to be returned. Subelements:
  - `start`
    The content of this element specifies the index of the first document to be returned. The index of the first document in the search result is 1. Default: 1.
  - `length`
    specifies the number of documents to return, starting with the document whose index is defined by `start` . Default: 20. If `offset` specifies a partly or completely nonexistent subset of the search result, no documents, or only those documents present in the range, respectively, are returned. In these cases, no error message is generated.
- `searchBase`

This optional element restricts the documents to search by a search query or a list of collections. If the element is not present, all indexed documents in all collections are searched. Subelements:

- `query`
  This element is optional. It specifies a search which is performed before the query given in the `query` element below `ses-search` is executed. If the element is not present, all indexed documents in the specified collections are searched. The element has the optional `parser` attribute, whose value can be `simple` (the default), `explicit`, or `freetext`. The content of the element is the search query stated in the search processor's syntax.
- `collection`
  This element is optional and can be specified more than once. Its content is the name of a collection whose documents are to be searched. If the element has not been specified, all the documents of all collections are searched.

- `sortOrder`
  This optional element can be used to determine the criteria by which the documents in the search result are sorted. By default, the documents are sorted by `score`, i. e. by relevance. The element must have at least one and can have up to 16 `sortField` subelements:
  - `sortField`
    Each `sortField` element determines the name of a document field to be used as sorting criterion, taking into account the order of the elements. The first `sortField` element defines the primary sorting criterion, the second element the second criterion and so forth. Of each field value only the first 64 characters are taken into account. All available fields plus `score` can be specified as sorting criterion. `sortField` has the optional `direction` attribute which specifes the sort order. Valid values are `asc` (ascending, the default) and `desc` (descending).
- `resultRecord`
  This element is optional. Its subelements specify the fields to be returned for each document in the search result. By default the fields `id`, `title`, and `score` are returned. Subelements:
  - `resultField`
    The contents of a `resultField` element specifies the name of the field to be returned for each document. The content of `resultRecord` may contain several `resultField` elements. All available fields as well as `id` and `score` can be specified. If a nonexistent field is specified, then the empty value is returned as its value. The element has the three attributes `format`, `timezone` and `formatter`:
    The value of `format` is the name of a date format used to format the field values concerned, provided they are date values. The format names and its formats are stored in the `validDateTimeOutputFormats` system configuration entry (see Executing the Search Engine Server). By default, the first format specified there is used.
    The `timezone` attribute can be used to specify the timezone into which date specifications are to be converted. By default the timezone of the server on which the Search Engine Server is running is used.
    Using the `formatter` attribute the values of the document fields returned can be formatted independently of their type. The value of the attribute is the alias name of a Tcl procedure. To this alias the true Tcl procedure name must have been assigned in the `tclFormatterCommands` system configuration entry (see Executing the Search Engine Server).
- `searchDirection`
  This optional element defines the order in which the documents in the specified collections are searched. This order is determined with the `start` attribute. Its value can be `newest` (the default) or `oldest`. `newest` causes the search to start with the most recent documents. Otherwise it starts with the oldest documents. The element does not have any content.

6.3.2 Response

The response to a search request is coded in the `searchResults` element in the response payload. This element is located below the `ses-code` element. It contains the requested subset of the search result. This subset is determined by the `offset` element contained in the request. Example:

```
<?xml version="1.0"?>
<!DOCTYPE ses-payload SYSTEM "http://www.infopark.com/ses.dtd">
<ses-payload payload-id="d--1950259307-000000004-X" timestamp="20101017150655"
 version="2.1">
  <ses-header>
    <ses-sender sender-id="SES-Infopark-DEV-0" name="SES"/>
    <ses-receiver name="CM Server" receiver-id="CM-Infopark-DEV-0"/>
  </ses-header>
  <ses-response response-id="0"
       request-id="d--1949717044-000000006-X" success="true">
    <ses-code phrase="OK" numeric="200">
      <searchResults hits="567" searched="2045381">
        <record index="21" offsetIndex="1">
          <title>A sample Document</title>
          <score>77</score>
          <docId>546381</docId>
        </record>
        ...
      </searchResults>
    </ses-code>
  </ses-response>
</ses-payload>
```

The `searchResults` element has the two attributes `hits` and `searched`. The value of `hits` specifies the total number of hits contained in the search result. The value of `searched` specifies the total number of documents that were searched. The content of `searchResults` is a list of `record` elements:

- `record`
  Each hit is represented by a `record` element. The element has the two attributes `index` and `offsetIndex`. The value of index is the index of the document in the complete search result, while `offsetIndex` is the index of the document in the requested subset of the search result. For both values the smallest value is 1.
  The content of the element is a list of elements each of which is the name of a document field (see Content Indexing). For each of the elements a corresponding `resultField` element was present in the search query.
  The content of a document field element is the value of the document field after formatting. Formatting can be achieved by means of the `format`, `timezone`, and `formatter` attributes in the corresponding `resultField` elements included in the search request.

6.4 Document Deletion Requests

6.4.1 Request

Document deletion requests are formed using the `ses-deleteDoc` element in a `ses-request` element, in accordance with the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ses-payload SYSTEM "http://www.infopark.com/ses.dtd">
<ses-payload payload-id="B42TE241" timestamp="20100825172100" version="2.1">
  <ses-header>
```

```
    <ses-sender sender-id="FX45RTDT" name="CM-Server"/>
    <ses-authentication login="cm-server" password=""/>
  </ses-header>
  <ses-request request-id="BR12TI5X">
    <ses-deleteDoc docId="4711" collection="collection1"/>
  </ses-request>
</ses-payload>
```

The `ses-deleteDoc` element has the following attributes:

- `docId`
  The ID of the document to be deleted. Normally this is the content ID (Content Management Server) or the object ID (Template Engine).
- `collection`
  The name of the collection from which the document is to be deleted.

6.4.2 Response

The Search Engine Server responds to a deletion request with an empty `ses-code` element which has the attributes described in section Response Element. Example:

```
<?xml version="1.0"?>
<!DOCTYPE ses-payload SYSTEM "http://www.infopark.com/ses.dtd">
<ses-payload payload-id="d--1950259307-000000003-X" timestamp="20101017150617"
 version="2.1">
  <ses-header>
    <ses-sender sender-id="SES-Infopark-DEV-0" name="SES"/>
    <ses-receiver name="CM Server" receiver-id="CM-Infopark-DEV-0"/>
  </ses-header>
  <ses-response response-id="0" request-id="d--1949717044-000000004-X" success="true">
    <ses-code phrase="OK" numeric="200"/>
  </ses-response>
</ses-payload>
```

6.5 Collection Deletion Requests

6.5.1 Request

All the documents indexed in a collection can be deleted by means of a collection deletion request. Such a request does not remove the collection itself, only the data in it is deleted. The collection's configuration is preserved in this process.

Collection deletion requests are formed by means of a `ses-purgeCollection` element contained in a `ses-request` element in accordance with the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ses-payload SYSTEM "http://www.infopark.com/ses.dtd">
<ses-payload payload-id="B42TE241" timestamp="20100825172100" ses.version="2.0">
  <ses-header>
    <ses-sender sender-id="FX45RTDT" name="CM-Server"/>
    <ses-authentication login="cm-server" password=""/>
  </ses-header>
  <ses-request request-id="BR12TI5X">
    <ses-purgeCollection>collection1</ses-purgeCollection>
  </ses-request>
</ses-payload>
```

The `ses-purgeCollection` element does not have any attributes.

6.5.2 Response

The Search Engine Server responds to a deletion request with an empty `ses-code` element which has the attributes described in section Response Element. Example:

```
<?xml version="1.0"?>
<!DOCTYPE ses-payload SYSTEM "http://www.infopark.com/ses.dtd">
<ses-payload payload-id="d--1950259307-000000003-X" timestamp="20101017150617"
 version="2.1">
  <ses-header>
    <ses-sender sender-id="SES-Infopark-DEV-0" name="SES"/>
    <ses-receiver name="CM Server" receiver-id="CM-Infopark-DEV-0"/>
  </ses-header>
  <ses-response response-id="0" request-id="d--1949717044-000000004-X" success="true">
    <ses-code phrase="OK" numeric="200"/>
  </ses-response>
</ses-payload>
```

6.6 Error Handling

In this section the errors are listed that can appear in `ses-code` elements contained in response payloads. Only protocol errors are mentioned here, and not the errors that can occur when the Search Engine Server performs an operation.

With protocol errors a distinction is made between errors on the payload level and on the request level.

6.6.1 Payload Errors

A payload error is generated if a request payload is illformed. If such an error occurs, all the requests contained in the payload are ignored. In the case of an error, the Search Engine Server returns a response payload containing a single `ses-response` element. Instead of the `request-id` attribute the opening tag of this element contains a `payload-id` attribute whose value is the ID of the invalid payload.

The `ses-response` element contains an `ses-code` element in whose opening tag the attributes `numeric` and `phrase` are set to the error number and the message text, respectively. The following example shows a response payload containing an error message.

```
<?xml version="1.0" encoding="UTF-8">
<!DOCTYPE ses-payload SYSTEM "http://www.infopark.com/ses.dtd">
<ses-payload payload-id="B3BWPOIU" timestamp="20100906100205" version="2.1">
  <ses-header>
    <ses-sender sender-id="U2JWUE09" name="SES"/>
  </ses-header>
  <ses-response response-id="BR12TI5X"
        payload-id="AHZ97I28" success="false">
    <ses-code numeric="1"
        phrase="Payload incomplete / cannot parse">
    </ses-code>
  </ses-response>
</ses-payload>
```

On the payload level the following errors can occur:

- **Payload incomplete / Cannot Parse**
  This is the error response to request payloads containing invalid XML code.
- **Not well-formed Payload** Request payloads that contain valid XML code but do not represent a valid request payload are answered with this message.
- **Incompatible Version**
  This error is returned for request payloads formulated in a version of the XML interface protocol not supported by the server. The behaviour of the server in such cases is described in section Payloads.
- **Authentication Failed**
  The information in the `ses-authentication` element is invalid.

6.6.2 Request Errors

Request errors refer to a single request only. Each request error is returned in the corresponding response in the response payload.

On the request level the following errors can occur:

- **Not well-formed request**
  The XML fragment contained in the `ses-request` element does not represent a valid request.
- **Execution precluded**
  The request has not been processed because a previous request that was marked as `preclusive` has failed.

7

7 MISE as DTD

This section contains the definition of MISE as DTD. It is subject to alterations.

7.1 Request

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT ses-request
  (ses-indexDoc|ses-deleteDoc|ses-search|ses-purgeCollection|
  ses-optimizeCollections|ses-flushQueue|ses-holdQueue|
  ses-resumeQueue)>
<!ATTLIST ses-request
  request-id CDATA #REQUIRED
  preclusive (true | false) "false"
>
<!ELEMENT ses-indexDoc ANY>
<!ATTLIST ses-indexDoc
  docId CDATA #REQUIRED
  collection CDATA #REQUIRED
  usesStreaming ("YES" | "NO") #DEFAULT "NO"
  mimeType CDATA #REQUIRED
>
<!-- The following element is an example for an attribute element
     contained in the ses-indexDoc element-->
<!ELEMENT blob (#PCDATA)>
<!ATTLIST blob
  encoding (base64|plain|stream) #DEFAULT plain
>
<!ELEMENT ses-deleteDoc EMPTY>
<!ATTLIST ses-deleteDoc
   docId CDATA #REQUIRED
   collection CDATA #REQUIRED
>
<!ELEMENT ses-search
  (query?,
  minRelevance?,
  maxDocs?,
  offset?,
  searchBase?,
  sortOrder?,
  resultRecord?,
  searchDirection?) >
<!ELEMENT query (#PCDATA)
<!ATTLIST parser (simple|explicit|freetext) #DEFAULT simple>
<!ELEMENT minRelevance (#PCDATA)>
<!ELEMENT maxDocs (#PCDATA)>
<!ELEMENT offset (start,length)>
<!ELEMENT start (#PCDATA)>
<!ELEMENT length (#PCDATA)>
<!ELEMENT searchBase (collection+,query?>
<!ELEMENT collection (#PCDATA)>
<!ELEMENT sortOrder (sortField+)>
<!ELEMENT sortField (#PCDATA)>
```

```
<!ATTLIST sortField direction (asc,desc) #DEFAULT asc>
<!ELEMENT resultRecord (resultField+)>
<!ELEMENT resultField (#PCDATA)>
<!ATTLIST resultField
  format (#CDATA) #IMPLIED
  timezone (#CDATA) #IMPLIED
  formatter (#CDATA) #IMPLIED
>
<!ELEMENT searchDirection EMPTY>
<!ATTLIST searchDirection start (newest|oldest) #DEFAULT newest
>
<!ELEMENT ses-optimizeCollections EMPTY>
<!ELEMENT ses-purgeCollection (#PCDATA)>
<!ELEMENT ses-flushQueue EMPTY>
<!ELEMENT ses-holdQueue EMPTY>
<!ELEMENT ses-resumeQueue EMPTY>
```

7.2 Response (ses-search)

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT ses-response (ses-code*)>
<!ATTLIST ses-response
  response-id CDATA #REQUIRED
  payload-id CDATA #IMPLIED
  request-id CDATA #IMPLIED
  successful (true | false) #REQUIRED
>
<!ELEMENT ses-code ANY>
<!ATTLIST ses-code
  numeric CDATA #REQUIRED
  phrase CDATA #REQUIRED
>
<!ELEMENT searchResults (record*)>
<!ATTLIST searchResults
  hits CDATA #REQUIRED
  searched CDATA #REQUIRED
>
<!ELEMENT record ANY>
<!ATTLIST record
  index CDATA #REQUIRED
  offsetIndex CDATA #REQUIRED
>
<!ELEMENT title ANY>
<!ATTLIST field
  type CDATA #REQUIRED
>
```