IDOL Server

Software Version 12.8

Administration Guide

Micro Focus®

Legal notices

Copyright notice

© Copyright 2021 Micro Focus or one of its affiliates.

The only warranties for products and services of Micro Focus and its affiliates and licensors ("Micro Focus") are as may be set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Micro Focus shall not be liable for technical or editorial errors or omissions contained herein. The information contained herein is subject to change without notice.

Documentation updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for updated documentation, visit https://www.microfocus.com/support-and-services/documentation/.

Support

Visit the MySupport portal to access contact information and details about the products, services, and support that Micro Focus offers.

This portal also provides customer self-solve capabilities. It gives you a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the MySupport portal to:

- Search for knowledge documents of interest
- Access product documentation
- View software vulnerability alerts
- Enter into discussions with other software customers
- Download software patches
- Manage software licenses, downloads, and support contracts
- Submit and track service requests
- Contact customer support
- View information about all services that Support offers

Many areas of the portal require you to sign in. If you need an account, you can create one when prompted to sign in. To learn about the different access levels the portal uses, see the Access Levels descriptions.

Contents

Part I: Introduction

This section provides a brief introduction to Micro Focus IDOL Server.

For more details on installing and running IDOL Server, refer to the *IDOL Getting Started Guide*.

- Introduction to IDOL Server

Chapter 1: Introduction to IDOL Server

The Micro Focus Intelligent Data Operating Layer (IDOL) Server integrates unstructured, semi-structured, and structured information from multiple repositories through an understanding of the content. It delivers a real-time environment to automate operations across applications and content, removing all the manual processes involved in getting information to the right people at the right time.

IDOL Server Operations

IDOL Server can perform the following intelligent operations across structured, semistructured, and unstructured data.

Agents	Expertise
Alerts	Hyperlinks
Automatic Query Guidance	Mailing
Categorization	Profiles
Channels	Retrieval
Cluster Data	Spelling Correction
Collaboration	Summarization
Dynamic Clusters	Taxonomy Generation
Dynamic Thesaurus	Viewing
Eduction	

NOTE: Your license determines which of these operations your IDOL Server installation can perform.

Agents

Users can create agents in IDOL Server to find and monitor information that is relevant to their interests. The agents collect this information from a configurable list of Internet and intranet sites, news feeds, chat streams, and internal repositories.

You can create agents in a user-friendly way using:

- Natural language descriptions

- Example content (point and click)

- Legacy keyword or Boolean expressions

IDOL Server provides the conceptual information that it needs to create agents. The server accepts a piece of content (training text, a document, or a set of documents) or reference (identifier), and returns an encoded representation of the concepts. This representation includes the specific underlying patterns of terms and associated probabilistic ratings for each concept.

Users can retrain their agents by submitting a new piece of content (training text, a document, or a set of documents). The server uses the new concepts to adapt the agent.

Alerts

IDOL Server analyzes data when it receives new documents, and compares the concepts in documents with user agents. If new data matches a user agent, it immediately notifies the user by email or a third-party system (for example by SMS or a pager).

Automatic Query Guidance

IDOL Server finds the most salient terms and phrases in query results, and automatically clusters these terms and phrases. It uses the clustered phrases to provide a hierarchical set of queries that guide users to the result area that they are looking for.

Categorization

IDOL Server can automatically categorize data. IDOL categorization allows you to derive categories from the concepts found in unstructured text. This process ensures that IDOL Server accurately classifies all data in the correct context. IDOL categorization is a scalable solution capable of handling high volumes of information accurately and consistently.

IDOL categorization does not rely on rigid rule-based category definitions such as legacy keyword and Boolean operators. Instead, IDOL infrastructure uses a pattern matching process based on concepts. It can then automatically tag data sets, route content, or alert users to relevant information pertinent to the user profile.

IDOL hooks into various repositories and data formats respecting all security and access entitlements, delivering complete reliability.

Category Matching

IDOL Server accepts a category or piece of content and returns categories ranked by conceptual similarity. This ranking determines the most appropriate categories for the piece of content, so that IDOL Server can subsequently tag, route, or file the content accordingly.

Channels

IDOL Server can automatically provide users with a set of hierarchical channels with highly relevant information pertinent to the respective channel. Channels are similar to agents, aggregating information that is relevant to the channel concept. Usually, administrators set up channels that are available to all users.

Real-time information dynamically updates in the channels. This process eliminates the requirement for manual intervention or pretagging, and minimizes the effort required to maintain it. Moreover, the administrator can add and remove channels on the fly, without recategorizing all the data.

Cluster Information

IDOL Server automatically clusters information. Clustering takes a large repository of unstructured data, agents, or profiles and automatically partitions the data to cluster similar information together. Each cluster represents a concept area in the knowledge base, and contains a set of items with common properties.

Collaboration

IDOL Server automatically matches users with common explicit interest agents or similar implicit profiles. This information creates virtual expert knowledge groups.

Dynamic Clusters

When it processes queries, IDOL Server automatically clusters the query results, and then clusters the first set of clusters to produce subclusters. This process allows you to generate a hierarchy of clusters that allows users to navigate quickly to their area of interest.

Dynamic Thesaurus

When it processes queries, IDOL Server can automatically suggest alternative queries, allowing users to quickly produce a variety of relevant result sets.

Eduction

Eduction is a tool that you can use to extract an entity (a word, phrase, or block of information) from text, based on a pattern you define. The pattern can be a dictionary of names such as people or places. The pattern can also describe what the sequence of text looks like without listing it explicitly, for example, a telephone number. The entities are contained inside grammar files.

When you use Eduction with IDOL Server, Eduction extracts the entities while the document is indexed and adds them into fields for easy retrieval.

The Eduction capability of IDOL Server is described in the *Eduction User and Programming Guide*.

Expertise

IDOL Server accepts a natural language or Boolean search string and returns users who own matching agents or profiles. This process allows instant identification of experts in a subject, eliminating time-consuming searches for specialists, and unnecessary researching of subjects for which expert knowledge is already available.

Hyperlinks

You can automatically generate hyperlinks in real time. These link to contextually similar content, for example to recommend related articles, documents, affinity products or services, or media content that relates to textual content.

IDOL Server automatically inserts these links when it retrieves the document. This process means that new documents can reference older documents, and that archived documents can link to the latest news or material on the subject.

Email Users

IDOL Server matches the agents and profiles against its document content at regular intervals, and automatically notifies users of documents that match their agents or profiles by emailing them.

Profiles

IDOL Server automatically creates interest and expertise profiles for users, in real time.

You can create interest profiles by tracking the content that a user views and extracting a conceptual understanding of it. IDOL Server then uses this understanding to keep user interest profiles up to date. You can use interest profiles to:

- Target information to users.

- Recommend content to users.

- Alert users to the existence of content.

- Put users in touch with other users who have similar interests.

You can create expertise profiles by tracking the content that a user creates and extracting a conceptual understanding of it. IDOL Server uses this understanding to keep user expertise profiles up to date. You can use expertise profiles to trace users who are experts in particular subject areas.

Search and Retrieval

IDOL Server offers a range of retrieval methods, from simple legacy keyword search to sophisticated conceptual querying.

Conceptual Matches

IDOL Server accepts a piece of content (a sentence, paragraph, or page of text, the body of an email, a record containing human-readable information, or the derived contextual information of an audio or speech snippet) or reference (identifier) as input and returns references to conceptually related documents ranked by relevance, or contextual distance. IDOL Server uses this process to generate automatic hyperlinks between pieces of content.

Advanced Keyword Search

IDOL Server matches any term or phrase that appears in quotation marks in its exact form before stemming.

Boolean and Bracketed Boolean Search

IDOL Server accepts simple or complex Boolean and bracketed Boolean expressions and returns a list of matching documents. You can form Boolean expressions using a set of Boolean and proximity operators.

AND	XOR/EOR	WNEAR
NOT	NEAR	BEFORE
OR	DNEAR	AFTER

Exact Phrase Search

IDOL Server provides the ability to search for exact phrases by putting quotation marks around a string of words. For example: "world market".

Field Restrictions

Simple field restrictions in query text allow you to restrict results to documents that contain specific values in specific fields.

FieldText Search

FieldText searches provide a wide range of field specifiers that you can use to query fields, restrict query results, or bias query result scores.

Fuzzy Search

If a search string is not quite accurate (for example, if it contains spelling mistakes), a fuzzy query returns results that contain words that are similar to the entered string.

Parametric Search

IDOL Server supports real-time navigation across multiple taxonomies with no additional manual configuration necessary, including full access to intersections of diverse taxonomy definitions.

From among the complete set of field names in the corpus, you can define a subset of fields in the server configuration as parametric fields.

After indexing, IDOL Server creates and stores a structure that contains information about all tag/value pairs that occur in defined parametric fields. A tag/value pair is a particular textual or numerical value paired with a specific field name.

You can then query IDOL Server with the name of a parametric field or fields. IDOL Server returns a list of all textual values that appear in the specified field or fields in all the documents stored in the server. This underlying operation can power a user interface that enables users to refine the scope of a query from the complete corpus to a subset of highly relevant documents.

Advanced Parametric Refinement provides an improved user experience coupled with increased productivity using an advanced real-time information discovery process.

Proper Names Search

IDOL Server recognizes names and treats them as a unit.

Proximity Search

IDOL Server returns documents in which specific terms occur within a specified proximity with a higher weighting.

Soundex Keyword Search

If the spelling of a keyword is not quite accurate but is phonetically correct, a Soundex keyword search returns results that contain the keyword and phonetically similar keywords. This process uses a configurable Soundex algorithm.

Synonym Search

A synonym query returns results that are conceptually similar to the query terms or conceptually similar to the synonyms that are available for the query terms.

Spell Check

IDOL Server can automatically spell check the query text it receives and suggest correct spelling for terms that its dictionary does not contain.

Summarization

IDOL Server accepts a piece of content and returns a summary of the information. IDOL Server can generate different types of summary.

- **Conceptual Summaries**. Conceptual summaries contain the most salient concepts of the content.

- **Contextual Summaries**. Contextual summaries relate to the context of the original query. They provide the most applicable dynamic summary in the results of a particular query.

- **Quick Summaries**. Quick summaries include a few sentences of the result documents.

Taxonomy Generation

Automatic taxonomy generation can automatically understand and create deep hierarchical contextual taxonomies of information. You can use clustering, or any other conceptual operation, as a seed for the process.

The resulting taxonomy can:

- Provide insight into specific areas of the information.

- Provide an overall information landscape.

- Act as training material for automatic categorization, which then places information into a formally dictated and controlled category hierarchy.

Automatic Taxonomy Based on Cluster Result

IDOL Server can use cluster results to build taxonomies automatically and in real time.

Automatic Taxonomy to Category Generation

After the automatic taxonomy generation process has taken place, IDOL Server contextually understands the type of data it is dealing with. It uses this understanding to generate a deep hierarchical contextual taxonomy, known as an information landscape. Much like the automatic cluster to category generation, this feature uses the taxonomy results to create categories.

View Documents

IDOL Server uses IDOL KeyView filters to convert documents into HTML format for viewing in a Web browser. It can convert documents that it retrieves from a local directory, intranet, or Internet source. It can also retrieve the document in its original format.

Getting Started

The *IDOL Getting Started Guide* contains information about how to install and run IDOL Server. It also provides an overview of the different IDOL setups that you can use.

You perform IDOL Server operations by sending *ACI actions* to IDOL Server from your Web browser.

You configure IDOL Server by modifying the appropriate configuration file.

A list of all available ACI actions, configuration parameters, index actions, and service actions, is available in the *IDOL Server Reference*.

Send Actions to IDOL Server

IDOL Server actions are HTTP requests, which you can send, for example, from your web browser. The general syntax of these actions is:

`http://host:port/action=action&parameters`

where:

`host`	is the IP address or name of the machine where IDOL Server is installed.
`port`	is the IDOL Server ACI port. The ACI port is specified by the `Port` parameter in the `[Server]` section of the IDOL Server configuration file. For more information about the `Port` parameter, see the *IDOL Server Reference*.
`action`	is the name of the action you want to run.
`parameters`	are the required and optional parameters for the action.

NOTE: Separate individual parameters with an ampersand (&). Separate parameter names from values with an equals sign (=). You must percent-encode all parameter values.

For more information about actions, see the *IDOL Server Reference*.

TIP: As an alternative to sending ACI actions directly, you can run any IDOL action from the IDOL Admin interface. For more information, refer to the *IDOL Admin User Guide*.

Display Online Help

The IDOL Server installers includes a help data file (help.dat) for each IDOL component that you can install. For the IDOL Server components, each help package includes *IDOL Expert*, the *IDOL Server Administration Guide*, and the appropriate component Reference (for example the *IDOL Content component Reference*). In a unified IDOL Server installation, it includes the full *IDOL Server Reference*.

You can display the help by sending an action from your Web browser.

> **NOTE:** For an IDOL component to display help, the help data file (`help.dat`) must be available in the same directory as the service instance.

To display help for an IDOL component

1. Start the IDOL component.

2. Send the following action from your Web browser:

    `http://IDOLhost:port/action=Help`

    where:

`IDOLhost`	is the IP address or name of the machine on which the component is installed.
`port`	is the ACI port by which you send actions to the component (set by the `Port` parameter in the `[Server]` section of the component configuration file).

    For example:

    `http://12.3.4.56:9000/action=Help`

3. On the help landing page, click one of the following options to open the relevant help set.

Admin Guide	The *Admin Guide* contains information about how to set up, configure, and administer IDOL Server.
Reference	The *Reference* contains details of all the actions and configuration parameters that you can set in the IDOL component.
IDOL Expert	*IDOL Expert* provides conceptual overviews and expert knowledge of IDOL Server and its features and functionality.

    If you are new to IDOL, you can use *IDOL Expert* to find out more about the different functions that IDOL can perform. You can use the *Reference* to look up specific configuration parameters and actions.

    The navigation panel for the *IDOL Server Reference* lists the following options to display reference information.

Tab	Description
Actions	Describes the actions that you can send to the IDOL component (or IDOL Server). Actions allow you to query the component, and to instruct it to perform a variety of operations.
Configuration Parameters	Describes the parameters that determine how the component operates. You can set configuration parameters in the component configuration file.
Index Actions	Describes the index actions that you can send to the component. This section is available only in components that accept index actions (primarily

Tab	Description
	Content and IDOL Proxy). Index actions allow you to index content into IDOL Server, and to administer the data index.
Service Actions	Describes service actions. Service actions allow you to return data about the IDOL component service, and to control the service.

Edit IDOL Configuration Files

You configure IDOL Server by manually editing the appropriate configuration file.

The configuration file that you use depends on your IDOL setup:

- In most cases, Micro Focus recommends that you configure and deploy IDOL components separately. In this case, each IDOL component has its own configuration file, which by default has the same name as the component executable file. For example, the IDOL Content component is `content.exe` and the configuration file is `content.cfg`.

- In simple testing deployments, you can use a unified IDOL Server configuration. In this case, you configure all the IDOL Server components (that is, Content, Category, Community, and View), as well as the IDOL Proxy component by using the IDOL Server configuration file (`idolserver.cfg`).

   NOTE: In the IDOL Server unified configuration, the IDOL Agentstore component is still configured separately, in the `agentstore.cfg`. The IDOL Agentstore component is a specially configured IDOL Content component for storing agents and categories.

   For more information about unified configuration, refer to the *IDOL Getting Started Guide*.

For more information about the IDOL Server configuration file, see IDOL Server Configuration File, on page 475.

Modify Configuration Parameter Values

You modify IDOL Server configuration parameters by directly editing the parameters in the configuration file. When you set configuration parameter values, you must use UTF-8.

   CAUTION: You must stop and restart IDOL Server for new configuration settings to take effect.

This section describes how to enter parameter values in the configuration file.

Enter Boolean Values

The following settings for Boolean parameters are interchangeable:

```
TRUE = true = ON = on = Y = y = 1

FALSE = false = OFF = off = N = n = 0
```

Enter String Values

To enter a comma-separated list of strings when one of the strings contains a comma, you can indicate the start and the end of the string with quotation marks, for example:

ParameterName=**cat,dog,bird,"wing,beak",turtle**

Alternatively, you can escape the comma with a backslash:

ParameterName=**cat,dog,bird,wing\,beak,turtle**

If any string in a comma-separated list contains quotation marks, you must put this string into quotation marks and escape each quotation mark in the string by inserting a backslash before it. For example:

ParameterName=**"<font face=\"arial\" size=\"+1\"><b>","<p>"**

Here, quotation marks indicate the beginning and end of the string. All quotation marks that are contained in the string are escaped.

Part II: Store Content in IDOL Server

This section explains the concept of indexing and describes how to index document content and metadata into IDOL Server.

- Configure Content Storage
- Index Data
- Fields
- Language Support
- Set Up Document Tracking

Chapter 2: Configure Content Storage

IDOL Server stores the content of documents in IDOL Content component data indexes. The process of storing content in the IDOL Content component is called *indexing*.

This section describes how to set up the IDOL Content component for indexing by editing the IDOL Content component configuration file.

Stored Content

Before you start to index files into the IDOL Content component, you must:

- decide how you want to store content.
- set up field indexing.
- configure Content to process required languages.
- optimize the indexing process according to your system.

You can also configure additional processing steps in your indexing process, for example to extract useful information from your documents to store as separate fields. You can set up additional processing in your IDOL connectors and Connector Framework Server (CFS). For more information, see the *Connector Framework Server Administration Guide*.

Related Topics

- Process Data before you Index, on page 50

Disable Content Storage

You can disable content storage if you do not require the IDOL Content component to return the content of fields or summaries with results. This option saves the memory that the storing of fields normally requires.

To disable content storage

- In the `[Server]` section of the IDOL Content component configuration file, set `NodeTableStoreContent` to **False**.

If you disable content storage, it affects the performance of the following actions.

GetContent	Returns only the references and the title of results.
GetTagValues	Unavailable.
List	Returns only the references and the title of results.
Query	Returns only the references and the title of results. You cannot restrict by fields.
Suggest	Returns only the references and the title of results. You cannot restrict by fields.
SuggestOnText	Returns only the references and the title of results. You cannot restrict by fields.
Summarize	Unavailable for indexed documents. You can generate summaries only if you supply text.
TermExpand	Expansion=Phrase unavailable.
TermGetBest	Content saves the best terms for a document on indexing. These are the only terms available.

Store Data Files on Multiple Disks

You can store data files across multiple partitions to gain space if the data in your IDOL Content component becomes too big to store on one volume (when the stored terms, references, content, and so on increase in size).

For example:

```
[PATHS]
DyntermPath=C:\IDOL\dynterm
NodetablePath=D:\IDOL\nodetable
RefIndexPath=E:\IDOL\refindex
MainPath=F:\IDOL\main
TagPath=.G:\IDOL\tagindex
GeospatialPath=H:\IDOL\geoindex
```

Compress the Data Index

You can configure the IDOL Content component to compress the nodetable directory to reduce the disk footprint, by using the NodeTableCompression configuration parameter.

> **NOTE:** If you change the compression method after you have started the server, the compression applies only for operations that index new data or rewrite existing data (such as DREREPLACE).
>
> To compress or change the compression of the whole index, you can run a DRECOMPACT operation with the NodeTableCompactWindowKB parameter set. For more information, refer to the *IDOL Server Reference*.

For more information about the NodeTableCompression option, and the compression method to use, refer to *IDOL Expert*.

To configure the IDOL Content component to compress the nodetable index

1. Open the IDOL Content component configuration file in a text editor.

2. In the `[Server]` section, add the `NodeTableCompression` parameter and set it to the compression method that you want to use to compress the index.

   You can set it to one of the following values:

   - `zlib`

   - `snappy`

   - `lz4`

   - `lz4hc`

3. Save and close the configuration file.

4. Restart the IDOL Content component for your changes to take effect.

Encrypt the Data Index

You can optionally configure the IDOL Content component to encrypt your index data.

When you enable index encryption, Content uses 256-bit AES encryption. Encryption includes:

- document data in the index.

- temporary data in the index cache.

- value mapping files used by the `parametric`, `match`, and `security` indexes.

- the term dictionary.

- the unstemmed term tree.

- the geospatial index.

- any data sent with index actions that is stored in your index queue before processing.

To use index encryption, you need an AES key file. You can create this key file by using the autpassword command-line tool. See Password Encryption, on page 507.

> **CAUTION:** If you lose your encryption keys after you enable encryption, you cannot recover your IDOL data.

In a distributed system, you can enable encryption both in your Content indexes, and in the Distributed Index Handler (DIH). The DIH encrypts the index data that it stores in the index queue while waiting to process it.

> **NOTE:** DIH decrypts the index queue content before it forwards data to its child Content indexes for processing. Micro Focus recommends that you enable TLS encryption to ensure that you have secure communication between Content and DIH.

Enable Index Encryption

There are two ways to enable encryption in your index:

- Set the `AESKeyFile` configuration parameter in the Content and DIH configuration file.

- Use the `-dataencryptionkey` command line parameter when you start Content and DIH.

The `dataencyptionkey` option overrides any `AESKeyFile` setting in your configuration file.

> **NOTE:** You can turn on AES encryption in an index that has existing content. In this case:
>
> - IDOL Server encrypts the value mapping and geospatial index files at startup
>
> - Document data encryption applies only to new data.
>
> - Any unprocessed data in your index queue remains unencrypted.
>
> - IDOL Server does not encrypt the unstemmed term tree unless you regenerate it (see).
>
> - IDOL Server encrypts the term dictionary when it next flushes the terms to disk during index, or when you run index compaction.
>
> To ensure that all your data is encrypted, Micro Focus recommends that you index your data into an empty index, with encryption enabled.

Enable Index Encryption in the Configuration File

You can enable index encryption by setting the `AESKeyFile` parameter in the IDOL Content component configuration file.

If you have a distributed system, you can also configure `AESKeyFile` in the DIH configuration file, to ensure that it encrypts the stored index queue data.

To configure index encryption

1. Open the IDOL Content component configuration file in a text editor.

2. Find the `[DataEncryption]`, or create one if it does not exist.

3. Set `AESKeyFile` to the full path to your AES encryption key file. For example:

   ```
   [DataEncryption]
   AESKeyFile=C:\idoldata\keys\indexkey.ky
   ```

4. Save and close the configuration file.

5. Restart the IDOL Content component for your changes to take effect.

> **NOTE:** Content does not start if it cannot find the specified key file, or if the key file is not valid.

Enable Index Encryption When You Start IDOL Components

When you start the IDOL Content component from the command line, you can use the `-dataencryptionkey` argument to enable encryption. You set this option to the 64-character

hexadecimal AES key that you want to use.

For example:

```
Content.exe -configfile Content.cfg -dataencryptionkey
D15B643D5332BB9B9871EB1828D91367FA5419FD7179C8254AA4CCB647AB8009
```

If you have a distributed system, you can also use this command-line parameter to add an encryption key when you start the DIH.

The data encryption key that you provide on startup overrides any configured value for `AESKeyFile`.

Check Your Encryption Status

You can check the encryption status of your Content index by sending the `GetStatus` action.

The response for Content includes a `data_encryption` section, which can have one of the following values:

- **false**. The index is not encrypted.

- **true**. All data in the index is encrypted.

- **partial**. Some data in the index is encrypted, but there is also some unencrypted data.

You can also send a `GetStatus` action to the Distributed Action Handler (DAH) to find the encryption status of all the DAH child servers. In this case, the response values are:

- **false**. None of your child servers are encrypted.

- **true**. All your child servers are fully encrypted.

- **partial**. At least one of your child servers contains encrypted data, but there are unencrypted or partially encrypted child servers.

Encryption Persistence

After you enable encryption, you cannot change your encryption settings or turn off encryption.

Each time you start Content, it verifies your encryption key to ensure that it matches the existing settings. If the encryption settings are different, or Content cannot find the key file, the server logs a warning and does not start.

Similarly, when you attempt to use the `DREINITIAL` index action with a backup path, Content checks the encryption settings in the backup directory before it restores the index. If the target index has incompatible encryption settings, the `DREINITIAL` index action fails with a **Bad Parameter** error.

> **NOTE:** Content stores only a hash generated from the encryption key on disk, so you cannot recover the original encryption keys from the saved information.

When you export data from Content by using `DREEXPORTIDX` or `DREEXPORTXML`, the exported content is unencrypted. Micro Focus recommends that you use appropriate authorization roles to ensure that only authorized users can export content from your index. See .

If you want to turn off encryption, or change your encryption settings, you must export your data and index into an empty Content index.

Allocate Files to IDOL Server Databases

You can configure the IDOL Content component to use a field in each document to determine the database into which it indexes the document.

To configure IDOL Server to read the database from a document field

1. Open the IDOL Content component configuration file in a text editor.

2. In the `[FieldProcessing]` section, list a process that identifies database fields.

   For example:

   ```
   [FieldProcessing]
   0=MyFirstProcess
   1=MySecondProcess
   2=DatabaseFields
   ```

3. Create a section for the database field identifying process. For example:

   ```
   [DatabaseFields]
   ```

4. Create a property for the process (you define the property later, by using one or more applicable configuration parameters). Identify the fields that you want to associate with the process.

   Optionally, use the `PropertyMatch` parameter to identify a specific value that fields must have to be processed.

   > **NOTE:** The property that you create must not have the same name as the process.

   For example:

   ```
   [MyFirstProcess]
   Property=MyFirstProperty
   PropertyFieldCSVs=*/MyField,*/MySecondField
   PropertyMatch=*myString*

   [MySecondProcess]
   Property=MySecondProperty
   PropertyFieldCSVs=*/MyOtherField,*/MyOtherSecondField

   [DatabaseFields]
   Property=Database
   PropertyFieldCSVs=*/DREDBNAME,*/DB,*/Database
   ```

5. Create a section for your indexing property and set the `DatabaseType` parameter to **True**.

   For example:

   ```
   [MyFirstProperty]
   HiddenType=True
   ```

```
[MySecondProperty]
Index=True

[Database]
DatabaseType=True
```

6. Save and close the configuration file.

7. Restart the IDOL Content component for your changes to take effect.

Enable the Phrase Index

The IDOL Content component phrase index stores short phrases from your index, for use with the `TermExpand` action phrase expansion. The `TermExpand` phrase expansion, and the phrase index is also used by Query Manipulation Server (QMS) `TypeAhead` action in index mode to find phrases.

The phrase index is not enabled by default, because it uses additional memory and on-disk storage, and adds overhead at index time. Micro Focus recommends that you enable the phrase index only if you want to use phrase expansion.

To enable the phrase index

1. Open the IDOL Content component configuration file in a text editor.

2. In the `[Server]` section, set `EnablePhraseIndex` to **True**.

3. Set any other optional parameters for the phrase index. For example, `PhraseIndexMaxTerms` controls the lengths of the phrases that Content stores (and therefore the length of the phrases is can expand).

   Other parameters (starting with the prefix `PhraseIndex`) control the memory usage for the phrase index, and the frequency that phrases must have in your index for them to be stored in the phrase index.

   For more information, refer to the *IDOL Content component Reference*.

4. Save and close the configuration file.

5. Restart the IDOL Content component for your changes to take effect.

> **TIP:** To enable the phrase index in an existing Content index, you can set `RegeneratePhraseIndex` to generate the required phrase index files at startup.

For more information about `TermExpand`, refer to the *IDOL Content component Reference*. For more information about `TypeAhead` in QMS, refer to the *Query Manipulation Server Reference*.

Set up the Field Index Process

IDOL connectors aggregate content and metadata, process it and then index it into the IDOL Content component in the form of fields. To improve IDOL Server performance, divide these fields into the following groups:

- **Prevent from storing**. Prevent Content from storing fields that you do not want to query by setting the `CantHaveFieldsCSVs` parameter in your IDOL Content component configuration file. Alternatively, add the `CantHaveFields` parameter to your `DREADD` or `DREADDDATA` index action.

- **Index fields**. Store fields that contain text that you want to query frequently as Index fields. Content processes Index fields linguistically when it stores them. Content applies stemming and stop word lists to the text, which allows it to process queries for these fields more quickly. Typically, you set up `DRETITLE` and `DRECONTENT` as Index fields.

  Do not use Index fields to store URLs or content that you are unlikely to use. Also, when you query the value only in its entirety, it is more efficient to query with a field specifier (for example, `MATCH`), than to store the data in an Index field.

  Micro Focus does not recommend indexing all fields in documents because it can potentially slow the indexing process, increase disk usage, and increase requirements.

- **Numeric fields**. Store fields that contain numerical values or dates as numeric fields and numeric date fields. When Content indexes these fields, it stores them in a fast look-up table in memory, which enables it to return the fields more quickly.

- **Field-check fields**. If a large number of the documents that you want to store in Content contain a field whose entire value is frequently used to restrict results, store this field as a `FieldCheckType` field. When Content indexes these fields, it stores them in a fast look-up table in memory, which enables it to return the fields more quickly.

- **Ordinary fields**. By default Content stores all fields that are not identified as special fields as ordinary fields.

> **NOTE:** You can query all stored fields using field specifiers in `FieldText` queries. You can also query Index fields using text queries.

Related Topics

Index XML Attributes

You can index XML attributes in the same way that you index ordinary fields. However, you must refer to them using the following format for the IDOL Content component to be able to read them:

*/*tagName*/_ATTR_*attributeName*

where:

tagName	is the name of the tag.
attributeName	is the name of the attribute that you want the IDOL Content component to read.

For example:

```
<FARM ANIMAL="sheep" COLOR="white"> Farmer Joe </FARM>
```

To identify the ANIMAL attribute to Content, refer to it as

```
*/FARM/_ATTR_ANIMAL
```

To identify the COLOR attribute to Content, refer to it as

```
*/FARM/_ATTR_COLOR
```

Example:

```
<ROOM Name="The Kitchen">
<FURNITURE>Table</FURNITURE >
<ITEM Type="China">Dish</ITEM>
</ROOM>
```

To identify the Name attribute to Content, refer to it as

```
*/ROOM/_ATTR_Name
```

To identify the Type attribute to Content, refer to it as

```
*/ITEM/_ATTR_Type
```

To store XML attributes in Index fields

1. Open the IDOL Content component configuration file in a text editor.

2. List an indexing process in the [FieldProcessing] section.

   For example:

   ```
   [FieldProcessing]
   0=MyFirstProcess
   2=IndexingFields
   ```

3. Create a section for the indexing process, in which you create a property for the process (you define a property later, by using one or more applicable configuration parameters). Identify the fields that you want to associate with the processes.

   > **NOTE:** The property that you create must not have the same name as the process.

   For example:

   ```
   [MyFirstProcess]
   Property=MyFirstProperty
   PropertyFieldCSVs=*/MyField,*/MySecondField
   PropertyMatch=*myString*

   [IndexingFields]
   Property=IndexFields
   PropertyFieldCSVs=*/FIELD/_ATTR_ANIMAL,*/FIELD/_ATTR_COLOR,*/ROOM/_ATTR_
   Name,*/ITEM/_ATTR_Type
   ```

4. Create a section for your indexing property in which you set the Index parameter to **True**.

For example:

```
[MyFirstProperty]
HiddenType=True

[IndexFields]
Index=True
```

5. Save and close the configuration file.

6. Restart the IDOL Content component for your changes to take effect.

Configure the IDOL Content component for Language and to Encode

Before you index documents that contain different languages into the IDOL Content component, you must configure it to recognize the language and encoding of documents, so that it can deal with them appropriately.

The IDOL Content component can automatically identify the language and encoding of a document when it indexes it. You must have an IDOL license that includes automatic language detection. To use this feature, set `AutoDetectLanguagesAtIndex` to **True** in the `[Server]` section of the IDOL Content component configuration file.

If your license does not include this functionality, you must specify the language and encoding of documents that you index into Content. Alternatively, you can configure a field process that allows Content to read the language of a document from one of its fields.

Related Topics

- Language Support, on page 109

Optimize Index Process

The speed of the indexing process is usually less critical than the speed of the query process. However, when you index large amounts of dat into the IDOL Content component, it is important to improve the efficiency of the process where possible. In addition, the way that you configure the indexing process can affect the efficiency of the query process.

Index Process

The indexing process works in two stages:

1. The IDOL Content component creates a representation of the new data in the index cache.

2. The IDOL Content component synchronizes the cache with data that it currently contains, and stores the new data on disk, removing it from the index cache.

When you schedule indexing, consider the recommendations in this chapter about IDOL content (particularly on selecting fields to index), and on running indexing and querying processes at different times. In addition, the delayed synchronization feature allows you to change the stage at which the IDOL Content component synchronizes the index cache. What you use depends on whether your priority is achieving fast query speeds or making new information available to the user as quickly as possible.

Delayed Synchronization

The delayed synchronization feature allows you to select how the IDOL Content component synchronizes the index cache with the permanent data index. This process is useful in systems where you schedule indexing tasks at times when Content is also handling queries.

By default, synchronization occurs as soon as a representation of data has been made in the index cache. New data is available to the user (as query results) quickly, so use this setting in systems where current data is the priority. However, synchronization uses resources that Content could otherwise use for querying.

Delayed synchronization reduces the impact of this effect by collecting multiple data representations in the index cache and then synchronizing them all in one go. This process is useful in systems where query speed is more important than having current data.

> **NOTE:** Micro Focus recommends using delayed synchronization if you index a lot of small files (files that are smaller than 100 MB).

The `DelayedSync` parameter in the `[Server]` section of the IDOL Content component configuration file allows you to specify whether the indexing process uses delayed synchronization.

- Set `DelayedSync` to **True** if you want the IDOL Content component to delay synchronization. In this case, Content stores data on disk only when:

    ○ the index cache is full.

    ○ the index cache contains some data and the timeout specified by `MaxSyncDelay` expires.

- Set `DelayedSync` to **False** if you do not want to delay synchronization.

Chapter 3: Index Data

IDOL Server stores the content of documents in data indexes. (The default data indexes are in the IDOL Server databases News and Archive.) The process of storing content in IDOL Server is called *indexing*. This section describes the indexing process.

Index Overview

You can index only files in XML or IDX format into the IDOL Content component. If the data that you want to index is in XML format, you can index it directly, without having to first *import* it (convert its content and metadata to IDX).

IDOL connectors use the DREADD and DREADDDATA index actions to index data. You can also use these actions to index data directly into the IDOL Content component.

> **NOTE:** Before you index data, review the setup instructions described in Configure Content Storage, on page 37.

If your data is not in XML format, you must first import it. You can import data using one of three methods.

- **Import with a connector**. The IDOL connectors (for example, File System Connector, HTTP Connector, Oracle Connector, and so on) retrieve documents from different repositories and use the Connector Framework Server (CFS) to import them into IDX or XML file format. Refer to the appropriate CFS or connector guide for further information on how to import documents.

- **Import manually**. You can create a text file in either XML or IDX format, which contains the information that you want to index into your IDOL Content component in specific IDOL fields.

- **Import with IDOL Admin**. You can use the wizard on the **Index** tab on the Console page in the Control section of IDOL Admin to submit data for the IDOL Content component to index. For more information, refer to the *IDOL Admin User Guide*.

Related Topics

- Manually Create IDX Files , on page 565

- IDX Format, on page 565

After the documents are in XML or IDX file format, you can index them into the IDOL Content component using one of two methods.

- **Index with a connector**. CFS indexes the IDX files that it creates into the IDOL Content component . Refer to the appropriate connector guide for detailed information on how to index documents.

- **Index directly**. You can index XML and IDX files into an IDOL Content component by issuing an HTTP request from your Web browser.

> **TIP:** If you index into IDOL from another host, you must configure the IDOL Content component to accept connections from the host, by applying the appropriate authorization roles. See Configure Client Authorization, on page 390.
>
> If you use CFS, you must assign CFS to the **Admin**, **Index**, and **Query** standard roles, or equivalents.

Related Topics

- DREADD: Index IDX and XML Files Directly, on the next page

- DREADDDATA: Index Data over a Socket, on page 57

Depending on where the data to index is located, the indexing steps for each document take place in the following order.

Local document (accessed through the file system)	Remote document (accessed over the indexing port)
- Content receives a file path. - Content opens the file and reads the document data. - Content indexes the document.	- Content receives a stream of data over the port. - Content saves the data locally. - Content opens the local file and reads the document data. - Content indexes the document.

Process Data before you Index

The IDOL Connector Framework Server (CFS) allows you to process documents that it receives (for example from an IDOL connector) before it sends them to the IDOL Content component index. For more information about using CFS to process your documents, refer to the *Connector Framework Server Administration Guide*.

DREADD: Index IDX and XML Files Directly

The DREADD index action (case sensitive) directly indexes an IDX or XML file that is located on the same machine as the IDOL Content component. For example:

http://*Contenthost*:*indexPort*/DREADD?*requiredParams*&*optionalParams*

where:

Contenthost	is the IP address or host name of the machine on which the IDOL Content component is installed.
indexPort	is the indexing port of the IDOL Content component (specified in the `IndexPort` parameter in the `[Server]` section of the IDOL Content component configuration file).

DREADD Parameters

The following parameters are available for the DREADD index action.

Required Parameters

You must include the following parameters.

`filename`
or
`path`
	The name or location of the IDX or XML file you want to index.
	
	The DREADD index action also accepts IDX or XML files that are compressed by gzip.

`DREDbName=database`	The name of the IDOL database into which you want to index the file content. You do not require this parameter if your IDX or XML files already contain a database field. By default, the IDOL Content component reads from this field.

Optional Parameters

You can include any of the following parameters as required. Separate parameters with an ampersand (&).

`ACLFields=fields`	One or more document fields from which Content reads ACLs (access control lists).
`CantHaveFields=fields`	One or more document fields to discard (not index). By default, all fields are indexed.
`DatabaseFields=fields`	One or more document fields that contain the name of the database into which you want to

	index the document.
`DateFields=fields`	One or more document fields from which you want Content to read the date of the document.
`Delete`	Content deletes the IDX or XML file after it is indexed.
`DocumentDelimiters=fields`	One or more fields in a file that indicates the beginning and end of an individual document, when the file contains multiple documents. Document delimiters cannot be nested.
`DocumentFormat=XML|IDX`	The format (XML or IDX) that Content assigns to a file when a file format is ambiguous.
`ExpiryDateFields=fields`	One or more fields that contain the expiration date of the document (the date when it is deleted or—if you set `ExpireIntoDatabase` in the Content configuration file—when it is moved into another database).
`FlattenIndexFields=fields`	One or more fields in a hierarchically structured document whose content you want to index as one level.
`IDXFieldPrefix=prefix`	When you index an IDX file, Content transforms it into XML by placing it under the `Document` subtree (each of the IDX fields is given the prefix `Document`, to construct a simple XML hierarchy). If you do not want to call the subtree `Document`, use this parameter to specify a different name.
`IndexFields=fields`	One or more fields that you want to index explicitly into Content. Explicitly indexing fields optimizes the query process when you use these fields to restrict queries. Use Index fields to hold data that is particularly significant to you (for example, the title of the document), and that you are likely to use frequently to restrict queries.
`KeepExisting=True|False`	If you set `KillDuplicates` to **Reference** or **FieldName**, you can set `KeepExisting` to **True** to discard the document received for indexing and keep the matching document that already exists in the database.
`KillDuplicates=option`	Specify one of the options described in Deduplication Options—KillDuplicates, on

	to determine how Content handles indexing of duplicate text. If you do not use the `KillDuplicates` option, indexing defaults to the setting specified for the `KillDuplicates` parameter in the `[Server]` section of the Content configuration file.
`KillDuplicatesDB=database`	The database to which Content moves duplicate documents.
`KillDuplicatesDBField=fields`	The name of a field in duplicate documents that contains the name of the database to which Content moves duplicate documents. If the field does not exist in the document, it uses the value of `KillDuplicatesDB`.
`KillDuplicatesMatchDBs=Db1+Db2+Db3`	Lists the databases to search for duplicate matches, separated by plus signs (+).
`KillDuplicatesMatchTargetDB=True|False`	Whether to search the database that the document is to index into for duplicate matches. Set to **True** to search the database.
`KillDuplicatesPreserveFields=True|False`	The name of IDX fields that Content must copy to a newer copy of the same document, when it performs `KillDuplicates`.
`LanguageFields=fields`	One or more fields that contain the language type of the document.
`LanguageType=type`	The language type to apply to a document if it has no fields that specify the language type. You define language types and how to handle them in the `[LanguageTypes]` section of the Content configuration file.
`MustHaveFields=fields`	One or more fields (in an IDX document only) that Content must store. By default, Content stores all fields. If you use this parameter, Content discards all document fields that are not specified—which means that you cannot query or print them.
`SectionFields=fields`	One or more fields that indicate the start of a new section in the document (for IDX only; Content automatically sections XML data).
`SecurityFields=fields`	One or more fields that contain the security type of the document.

`SecurityType=type`	The security type to apply to a document if the document has no fields that specify the security type. You define security types and how to handle them in the Content configuration file.
`TitleFields=fields`	One or more fields from which Content reads the document title.

Related Topics

- Specify Field Names, below
- Use KeepExisting to Minimize the Index Load, on page 71

> **NOTE:** Parameters used in the `DREADD` index action override the equivalent settings specified in the IDOL Content component configuration file.

DREADD Examples

```
http://MyHost:20001/DREADD?C:\Documents and
Settings\JBrown\Market.idx&DREDBNAME=Biz
```

```
http://MyHost:20001/DREADD?D:\Content\Price.idx&DREDBNAME=Shop&KillDuplicates=Refer
ence
```

Specify Field Names

If you specify multiple field names, separate them with commas. There must be no space before or after any comma. You can use Wildcards.

When naming fields, use the format */FieldName* to match root-level fields, **/FieldName* to match all fields except root-level, or */Path/FieldName* to match fields with the specified path. If you just specify *FieldName*, the IDOL Content component automatically adds **/* to it.

Related Topics

- Wildcards in Queries, on page 306

ACLFields Example

`&ACLFields=*/AUTONOMYMETADATA`

Content reads ACLs from any fields called `AUTONOMYMETADATA`.

DatabaseFields Example

`&DatabaseFields=Document/DREDbName,*/myDB`

Content indexes the document into the database with the name contained in any `DREDbName` field below the `Document` level and with the name contained in any fields called `myDB`.

DateFields Example

```
&DateFields=Document/DREDate,*/myDocDate
```

Content extracts dates from any fields called `DREDate` contained below the `Document` level and from any fields called `myDocDate`.

> **NOTE:** If you index documents that contain fields with partial dates into Content, the document is assigned a metadata field specifying a full date. This date is the first day of the month if the day is not specified, and the first of January if neither the day or the month are specified. The format of the partial date must be a format specified in the `DateFormatCSVs` configuration parameter.

DocumentDelimiters Example

```
&DocumentDelimiters=*/DOCUMENT,*/SPEECH
```

The IDOL Content component marks the beginning and end of individual documents in the file by opening and closing `DOCUMENT` and `SPEECH` tags.

ExpiryDateFields Example

```
&ExpiryDateFields=Document/DREExpiryDate,*/myExpiryDate
```

The IDOL Content component reads the expiration date from any `DREExpiryDate` field below the `Document` level and from any fields called `myExpiryDate`.

FlattenIndexFields Example

```
<documents>
   <article id="_21498602">
      <url>http://example.com/21490.html</url>
      <hltext_display>The history of pharmacogenetics </hltext_display>
      <source>Science Online</source>
      <media_type>text</media_type>
      <subject>
         <text>The prologue to pharmacogenetics began to play out around 1850 and
spanned some 60 years into the 1900s.</text>
         <text>In 1953, the molecular basis of heredity, the double helix of DNA,
was described.</text>
      </subject>
      <valid_time>Jul 13 2001  5:00AM</valid_time>
   </article>
</documents>
```

If you specify `FlattenIndexFields=*/subject`, and index the above document, the IDOL Content component indexes any content in a `subject` field or a field within a `subject` field as the content of the `subject` field.

If you then query the `subject` field for a particular term that is actually contained in a level below the `subject` field (such as the term "pharmacogenetics"), the content of both `text` fields returns. If you do not flatten the `subject` field the query does not return results, because the `subject` field itself does not contain the term.

IndexFields Example

`&IndexFields=*/DRECONTENT,*/DRETITLE`

The IDOL Content component explicitly indexes the `DRECONTENT` and `DRETITLE` field in documents.

LanguageFields Example

`&LanguageFields=Document/DRELanguageType,*/myLanguageType`

In this example, Content reads the language type of documents from any `DRELanguageType` field below the `Document` level and any `myLanguageType` fields.

MustHaveFields Example

`&MustHaveFields=*/DRECONTENT,*/DRETITLE`

In this example, Content stores only the `DRECONTENT` and `DRETITLE` fields in the document.

SectionFields Example

`&SectionFields=Document/DRESection,*/mySection`

In this example, any `DRESection` field below the `Document` level and any `mySection` fields indicate the start of a new section.

SecurityFields Example

`&SecurityFields=Document/DRESecurity,*/mySecurity`

In this example, Content reads the security type of documents from any `DRESecurity` field below the `Document` level and any `mySecurity` fields.

TitleFields Example

`&TitleFields=*/DRETITLE`

In this example, Content reads a document title from its `DRETITLE` field.

DREADDDATA: Index Data over a Socket

The DREADDDATA index action (case sensitive) allows you to directly index data over a socket into the IDOL Content component. For example:

DREADDDATA?*optionalParamsData*#DREENDDATA*killDuplicatesOption*\n\n

> **NOTE:** This index action requires a POST request method. See Send Data with a POST Method, on the next page.

DREADDDATA Parameters

The following parameters are available for the DREADDDATA index action.

Data	The content of the IDX or XML document to index. You can use gzipped data. You must add #DREENDDATA to the end of your data. #DREENDDATA must be uncompressed, even if your data is in gzip format. This parameter is required.
optionalParams	The DREADDDATA action accepts the same optional parameters as the DREADD action, except for KillDuplicatesOption. > **NOTE:** The DREDBName parameter, which is required for DREADD, is an optional parameter for DREADDDATA.
killDuplicatesOption	This optional parameter is equivalent to the KillDuplicates parameter for DREADD, except that it does not use the "KillDuplicates=*Option*" syntax. You append the option directly to the #DREENDDATA tag that ends the *Data* parameter (for example, #DREENDDATAREFERENCE). The following option values are available: • NOOP (available for DREADDDATA only) • NONE • REFERENCE • REFERENCEMATCH*N* • *FieldName* • *ReferenceField,*GREATER:*VersionField* For more information about these options, see Deduplication Options—KillDuplicates, on page 66.

Related Topics

> **NOTE:** Parameters that you set in the DREADDDATA action override any equivalent settings specified in the IDOL Content component configuration file.

Send Data with a POST Method

You must send the DREADDDATA action using a POST request method. There are two ways to send a POST request over a socket to the IDOL Content component:

- use the Curl command-line tool

- use a script

> **NOTE:** You can use these methods for other actions that require a POST request method, such as DREREPLACE, but you must modify the script.

Use the cURL Command-Line Tool

Curl is an open source command-line tool for transferring files with URL syntax. If you have cURL installed on your computer, and a command prompt that allows you to use new lines, you can use a cURL request to send the action. For example:

```
curl "http://host:port/DREADDDATA?" -d "
#DREREFERENCE Test
#DREDBNAME Default
#DREENDDOC
#DREENDDATA
"
```

Alternatively, you can add the #DREENDDATA tag into the IDX or XML document, or gzipped IDX or XML file, after the data to index. The #DREENDDATA tag must be uncompressed. Then you can use a cURL command to send this document to the IDOL Content component.

For example:

```
curl --data-binary @filename "http://host:port/DREADDDATA?"
```

where *filename* is the name of the IDX, XML, or gzipped IDX or XML file to index.

For information on cURL, refer to http://curl.haxx.se/.

Use a Script

Another method of sending a POST request is to use a script to open a socket and send the data. The following is an example script in the Perl programming language that sends a DREADDDATA index action to a specified port.

To run the script

- Open a command prompt and type the following command:

PerlScript.pl *HostNameIndexPortFilename* [*Parameters*]

where:

PerlScript .pl	is the name of the file that contains the Perl script that performs the DREADDDATA index action.
HostName	is the host name or IP address of the host on which the IDOL Content component runs.
IndexPort	is the index port for the IDOL Content component you send the data to.
Filename	is the name of the file that you want to index into the IDOL Content component by using the DREADDDATA action. You can use an IDX, XML, or gzipped IDX or XML file.
Parameters	are any optional parameters that you use for the DREADDDATA action.

Example Perl Script

```
# Performs a /DREADDDATA index action
use IO::Socket;

if (@ARGV<3)
{
    print "Usage: doDreAddData.pl <hostname> <indexport> <filename> [parameters]\n";
    exit;
}

my $host = $ARGV[0];
my $port = $ARGV[1];
my $filename = $ARGV[2];
my $params = $ARGV[3];
my $footer = "\r\n#DREENDDATA\r\n\r\n";
my $iaddr = inet_aton($host) or die "$!";
my $paddr = sockaddr_in($port,$iaddr);
my $proto = getprotobyname('tcp');

socket(SOCK, PF_INET, SOCK_STREAM, $proto) or die "socket: $!";
connect(SOCK, $paddr) or die "connect: $!";

my $filesize = -s $filename;
$filesize += length($footer);
open (FILE, $filename) or die "couldn't open file: $!";

print SOCK "POST /DREADDDATA?$params HTTP/1.1\r\n";
print SOCK "Connection: close\r\n";
print SOCK "Content-Length: $filesize\r\n\r\n";
while (<FILE>)
```

```
{
    print SOCK $_;
}
print SOCK $footer;

SOCK->autoflush(1);

while(<SOCK>){
    print $_;
}

close FILE;
close SOCK;
```

DREADDDATA Examples

Example 1:

```
POST /DREADDDATA?LanguageType=EnglishUTF8 HTTP/1.0
Content-Length: 604
#DREREFERENCE 392348A0
#DREFIELD authorname1="Brown"
#DREFIELD authorname2="Edgar"
#DREFIELD title="Dr."
#DREDATE 1998/08/06
#DRETITLE
Jurassic Molecules
#DRECONTENT
Using a technique called test tube evolution, scientists created a nucleic acid
enzyme, the first known enzyme that uses an amino acid to start chemical activity.
Scientists hope that the creation of this molecule will lead to the elusive
precursor. The precursor, by definition, will have to contain both the genetic code
for replication and an enzyme to trigger self replication.
#DREDBNAME Science
#DREENDDOC
#DREENDDATAREFERENCE
```

Example 2:

```
POST /DREADDDATA?DREDbName=Poetry&DateFields=*/Date HTTP/1.0
Content-Length: 515
Content-Type: application/xml
<DOCUMENT>
<DREREFERENCE>572801A2</DREREFERENCE>
<AUTHOR>George Eliot</AUTHOR>
<Date>2005/24/03</Date>
<DRETITLE>Roses</DRETITLE>
<DRECONTENT>
You love the roses - so do I. I wish
The sky would rain down roses, as they rain
```

```
From off the shaken bush. Why will it not?
Then all the valley would be pink and white
And soft to tread on. They would fall as light
As feathers, smelling sweet; and it would be
Like sleeping and like waking, all at after!
</DRECONTENT>
</DOCUMENT>
#DREENDDATANOOP
```

Index Stop Words

The IDOL Content component does not normally index stop words, but you can index them by using the `StopWordIndex` configuration parameter in the `[LanguageTypes]` section of the configuration file. For example:

```
[LanguageTypes]
StopwordIndex=1
```

Queries match stop words only in a phrase, as shown in the following examples.

Query	Result
`winnie the bear`	Content ignores the stop word *the*, as usual.
`"winnie the bear"`	Content includes the stop word *the* in the exact phrase search and matches it.
`copy of "winnie the bear"`	Content ignores the stop word *of*, but matches *the* within the phrase.

> **NOTE:** Wildcard search terms do not expand to stop words, even in phrases.

Related Topics

- Stop Word Lists for Supported Languages, on page 564

Index Nonalphanumeric Characters

You can configure nonalphanumeric characters to index differently based on the query result that you want. By using these configurations, you can query nonalphanumeric terms in exactly the same form as in a document to match that document.

You must set the configuration parameters discussed in this section in the `[MyLanguage]` section of the IDOL Content component configuration file.

> **NOTE:** If you want to change these settings after you index content into the IDOL Content

component, you must reindex the content.

Related Topics

- Enable Transliteration, on page 126

Term Separators

The IDOL Content component automatically generates separators for each language to determine where one term ends and another begins. These include characters such as spaces, tabs, carriage returns, and line feeds.

To ensure that Content uses a character as a separator, specify it in the `AugmentSeparators` configuration parameter. Content replaces all separator characters with a space.

For example, the following table describes the query matching for when `AugmentSeparators=,-`.

Indexed string	Query terms matched
second-hand guitar	*second* *hand* *guitar*

> **NOTE:** The hyphen is a separator only if it is not listed in `HyphenChars`, because `HyphenChars` takes precedence over separators.

To ensure that Content does not use a character as a separator, specify it in the `DiminishSeparators` configuration parameter. Content removes nonseparators at index time.

For example, the following table describes the query matching for when `DiminishSeparators=_%`.

Indexed string	Query terms matched
file_name	*filename*

To ensure that Content indexes a character as its own token, specify it in the `SoftSeparators` configuration parameter.

For example, the following table describes the query matching for when `SoftSeparators=1234567890`.

Indexed string	Query terms matched
459	*4* *5* *9* *45* *59*

Indexed string	Query terms matched
	• *459*

In this example, Content tokenizes all numbers as single digits, so that 459 is indexed as 4 5 9.

Related Topics

-

Index Nonalphanumeric Characters for Retrieval

To ensure that a nonalphanumeric character is available for querying, specify it in the `TangibleCharacters` configuration parameter.

For example, the following table describes the query matching for when `TangibleCharacters=?!`.

Indexed string	Query terms matched
help!	*help!* Queries do not match documents that contain the same word without the exclamation mark.

> **NOTE:** You cannot specify spaces, returns, and tabs as `TangibleCharacters`.

To ensure that Content indexes numbers with decimals or commas together as a single term for querying, specify both characters in the `NumberPunctuation` configuration parameter.

Content treats characters that you set as `NumberPunctuation` as `TangibleCharacters` when they occur in terms with a number on both sides.

For example, the following table describes the query matching for when `NumberPunctuation=.,`.

Indexed string	Query terms matched
815,290.50	*815,290.50*
73.8A	*73.8A*
738.	*738* (number punctuation does not apply.

> **NOTE:** These results can vary depending on your `IndexNumbers` configuration parameter setting.

When you configure a character in `NumberPunctuation` and another tokenization parameter, such as `HyphenChars`, `TangibleCharacters`, or `AugmentSeparators`, Content processes the parameters in order of precedence. For example, `TangibleCharacters` takes priority over `NumberPunctuation`, so if you configure the period in both parameters, you get the same results as if you only configured it in `TangibleCharacters`.

For more information about the order in which Content applies different tokenization configurations, refer to *IDOL Expert*.

When a configured `NumberPunctuation` character occurs in a value where it does not have a number on each side, the processing depends only on your other configuration. For example, the following table describes query matching for the value `25.R` in various configurations. This value does not activate `NumberPunctuation`, because there is a number on only one side of the period.

If the period (.) is configured as	Query terms matched
non-separator (default)	*25R*
`HyphenChars`	*25*, *R*, and *25R*
`TangibleCharacters`	*25.R*
`AugmentSeparators`	*25* and *R*.

Related Topics

- Configure the Number Index Process, on page 94

Hyphenated Terms

By default, when Content indexes a hyphenated term, it stems each of its components and indexes them. It also removes the hyphen from the term, stems the resulting term, and indexes that.

For example, the following table describes the default query matching for hyphenated terms.

Indexed string	Query terms matched
`second-hand guitar`	*second* *hand* *secondhand* *guitar*

To treat other characters as hyphens, specify them in the `HyphenChars` configuration parameter.

For example, the following table describes the query matching for when `HyphenChars=-&`.

Indexed string	Query terms matched
`Barnes&Noble`	*Barnes* *Noble* *BarnesNoble*

NOTE: To stop Content from indexing hyphenated terms this way, set `HyphenChars=NONE`. This means that no characters are used as `HyphenChars`. The default setting is `HyphenChars=-`.

At query time, Content tokenizes the hyphenated term as the whole term and individual subterms, all separated by the configured `DefaultQueryOperator`. For example, if the `DefaultQueryOperator` is OR (the default value), a query for `second-hand` becomes `secondhand OR second OR hand`.

> **NOTE:** If you change the `DefaultQueryOperator`, you might not get expected query results for hyphenated terms.

The separate parts of a hyphenated term (such as *second* and *hand*) are available only for basic keyword searches, and they do not match proximity expressions or field-restricted searches. For example, if a document contains the phrase `second-hand car`, a keyword search for `hand` matches the document, but an exact phrase search for `hand car` does not.

In most situations, Micro Focus recommends that you set `HyphenChars` to **NONE**, and use the hyphen as a separator (for example, set `AugmentSeparators` to **-**). For more information, refer to *IDOL Expert*.

Character Tokenization

You can tokenize characters into *N-gram*s of a specified size. Set the `NGram` configuration parameter in your language configuration section to the number of characters to use in each N-gram group.

> **NOTE:** You must not use `NGram` with the `SentenceBreaking` configuration parameter.

For example, if you set `NGram` to **2**, Content tokenizes the word *Hello* as:

```
he el ll lo
```

To tokenize only multibyte strings, set `NGramMultiByteOnly` to **True**.

```
[Japanese]
NGram=2
NGramMultiByteOnly=True
```

For this configuration, if you have a document that contains both English and Asian (multibyte) text, Content tokenizes the Asian text according to the `NGram` parameter. It does not tokenize the English text.

To tokenize only multiple-byte strings in Chinese, Japanese, and Korean characters (and ignore multiple-byte strings in other languages), set `NGramSentenceBrokenScriptOnly` to **True**.

```
[Japanese]
NGram=2
NGramSentenceBrokenScriptOnly=True
```

For this configuration, if you have a document that contains multibyte text in both Japanese and Greek, Content tokenizes the Japanese text according to the `NGram` parameter. It does not tokenize the Greek multibyte text.

Prevent Duplicate Documents

You can configure the IDOL Content component to implement *deduplication* when indexing documents. This process prevents storage of the same document or document content. If Content determines that the document to index matches an existing document, it replaces the existing document with the new document.

The IDOL Content component uses deduplication options to determine whether documents match. See Deduplication Options—KillDuplicates, below.

You can enable deduplication in one of three ways:

- Enable deduplication for all indexing jobs by using the `KillDuplicates` configuration parameter in the `[Server]` section of the IDOL Content component configuration file. See Enable Deduplication for all Index Jobs, on page 68.

    You can use the `KillDuplicatesChecksumField` configuration parameter with deduplication to prevent the IDOL Content component from unnecessarily updating existing documents. See Use KillDuplicatesChecksumField to Prevent Unnecessary Indexing, on page 70.

    You can also use the `KillDuplicatesPreserveFields` configuration parameter with deduplication to copy the specified IDX fields from an existing document to a newer version.

- Enable deduplication for individual indexing jobs by using the `KillDuplicates` action parameter in the `DREADD` and `DREADDDATA` actions. See Enable Deduplication for Individual Index Jobs, on page 70.

    Use the `KeepExisting` action parameter with deduplication to discard the incoming document instead of replacing the existing document, This option reduces the indexing load. See Use KeepExisting to Minimize the Index Load, on page 71.

- Enable deduplication when indexing with Connector Framework Server (CFS) by setting the `KillDuplicates` configuration parameter for the connector. See Enable Deduplication for Connector Index Jobs, on page 71.

Some other IDOL Content component parameters affect the behavior of the deduplication settings. See Deduplication Constraints, on page 71.

You can deduplicate after indexing by using the `DREDUPLICATE` index action. See Locate Duplicate Documents, on page 421.

Deduplication Options—KillDuplicates

Use the following parameters to specify deduplication options. The IDOL Content component uses these parameters to determine whether documents match.

- The `KillDuplicates` parameter specified in either the `[Server]` section of the IDOL Content component configuration file or in the `DREADD` or `DREADDDATA` index action.

- The `KillDuplicates` parameter specified in the `[Indexing]` section of the CFS configuration file.

The following options are available for the deduplication parameters.

NONE	Allows duplicate documents in Content index. Content does not replace or delete documents.
REFERENCE	Replaces an existing document with the new document if the document to index has the same value in its DREREFERENCE field.
REFERENCEMATCH *N*	Replaces the existing document with the new document if the content of the document to index is more than *N* percent similar to the existing document. Content determines the similarity by comparing the content of the SourceType fields in the document, or the Index fields if no SourceType fields are configured. **NOTE:** This method can deduplicate only documents that are already synced in the IDOL Server index. It cannot deduplicate similar documents in the same index job.
FieldName	Replaces the existing document with the new document if the document to index contains a ReferenceType field named *FieldName* that has the same content as the *FieldName* field in the existing document. You can specify multiple ReferenceType fields in this option (separated by a plus symbol or space), in which case Content deletes documents that contain any of the specified fields with identical content. You must percent-encode any punctuation characters in the field name. **NOTE:** You identify fields as ReferenceType fields through field processes in the IDOL Content component configuration file. If you list multiple fields in the same PropertyFieldCSVs parameter where you list the *FieldName* for deduplication, Content uses all the fields to eliminate duplicate documents. If you want to define multiple ReferenceType fields but do not want to use all fields for duplicate elimination, set up multiple field processes.
ReferenceField ,GREATER: *VersionField*	Replaces the existing document with the new document if the document to index contains a ReferenceType field named *ReferenceField* that has the same content as the *ReferenceField* field in the existing document, and if the *VersionField* field in the document to index has a higher value than the *VersionField* in the existing document. For XML documents, you must fully qualify the path of the XML field that you want to use as the version field (you cannot use wildcard values). *VersionField* must contain a positive integer value, but you do not need to configure it as a numeric field. If only one of the incoming and current documents has a valid value in the *VersionField*, IDOL Server keeps the version with a valid *VersionField*. When both documents have the same *VersionField*, IDOL Server keeps the existing document. **NOTE:** When you index IDX documents, for the version comparison to work correctly, the value in the field that you use as the *VersionField* must be listed in quotation marks (""). That is, the field must have the following format in the IDX:

	`#DREFIELD `*`MyField`*`="N"` IDOL Server treats existing documents with a missing or non-numeric value in the *VersionField* as having a version number of negative infinity. It treats a new document with a missing or non-numeric value in the *VersionField* as having a version number of 0.
NOOP (DREADDDATA only)	Use the `KillDuplicates` parameter in the `[Server]` section of the IDOL Content component configuration file to determine how to treat duplicate documents. **NOTE:** This option is available only for the `DREADDDATA` action.

Related Topics

- ReferenceType Fields, on page 99
- Enable Deduplication for all Index Jobs, below

When you specify a deduplication option, note that:

- If you postfix any of these options with **=2**, Content applies the `KillDuplicates` process to all databases, rather than just the database into which the current IDX or XML file indexes. For example:

`KillDuplicates=REFERENCE=2`

- The setting in the `KillDuplicates` option in either the `DREADD` or `DREADDDATA` index action overrides the setting in the `KillDuplicates` configuration parameter.

Enable Deduplication for all Index Jobs

To enable deduplication for all indexing jobs—in other words, to set deduplication by default for the `DREADD` and `DREADDATA` actions—use the `KillDuplicates` configuration parameter in the `[Server]` section of the configuration file.

IMPORTANT: You must enable deduplication before you start indexing documents into the IDOL Content component.

You can use the `KillDuplicatesChecksumField` parameter to configure Content to reverse normal deduplication and retain the existing document instead of the incoming document, based on the value of a specified field in the incoming document.

You can use the `KillDuplicatesPreserveFields` parameter to configure one or more IDX fields that Content copies to a newer version of a duplicate document.

Related Topics

- Use KillDuplicatesChecksumField to Prevent Unnecessary Indexing, on page 70

To enable deduplication as the default for all indexing jobs

1. Open the IDOL Content component configuration file in a text editor.

2. In the [Server] section, set the KillDuplicates parameter to **REFERENCE**, **REFERENCEMATCH*N***, the names of the ReferenceType fields to use to determine which documents are duplicates, or a combination of ReferenceType field and a field that contains a document version number. For more information about these options, see Deduplication Options—KillDuplicates, on page 66, or refer to the *IDOL Server Reference*.

You can identify fields that contain document references by setting up an appropriate field process. When you index a document that has the same value in the same ReferenceType field as an existing document in Content, Content detects the duplicate. It deletes the existing document and replaces it with the new one.

3. Save and close the configuration file.

4. Restart the IDOL Content component for your changes to take effect.

You can now index documents into the IDOL Content component.

Related Topics

- Deduplication Options—KillDuplicates, on page 66

- Set up ReferenceType Fields, on page 99

Limit ReferenceType Fields used for Deduplication

You identify fields as ReferenceType fields through field processes. If you list multiple fields in the same PropertyFieldCSVs parameter where you list the *FieldName* for deduplication, Content uses all the fields to eliminate duplicate documents. For example:

```
[SetReferenceFields]
Property=Reference
PropertyFieldCSVs=*/DREREFERENCE,*/URL
```

In this example, Content uses both the DREREFERENCE field and URL field to eliminate duplicate copies if you set KillDuplicates to **DREREFERENCE**.

If you want to define multiple ReferenceType fields but do not want to use them all for duplicate elimination, set up multiple field processes. For example:

```
[SetReferenceFields]
Property=Reference
PropertyFieldCSVs=*/DREREFERENCE

[SetMoreReferenceFields]
Property=Reference
PropertyFieldCSVs=*/URL
```

In this example, Content uses only the DREREFERENCE field to eliminate duplicate copies if you set KillDuplicates to **DREREFERENCE**. It does not use the URL field.

Related Topics

- Set up ReferenceType Fields, on page 99

Use KillDuplicatesChecksumField to Prevent Unnecessary Indexing

By default, when Content detects that a new document is a duplicate of an existing one, it replaces the existing document with the new one.

For either of these two `KillDuplicates` options, you can also use the `KillDuplicatesChecksumField` configuration parameter to specify a checksum field. Content then checks the value of this field in both documents. If the value is the same, Content keeps the existing document rather than replacing it with the new document.

This process prevents unnecessary updates. For example, when refetching a Web site, use `KillDuplicatesChecksumField` to configure Content to update the index for this site only if the site has changed.

> **NOTE:** The `KillDuplicatesChecksumField` must be a `ReferenceType` field.

Use KillDuplicatesPreserveFields to Preserve a Field

If there is a field that you want to keep in all versions of a document, regardless of whether it is later deleted or changed, you can use the `KillDuplicatesPreserveFields` configuration parameter.

To preserve fields, set `KillDuplicatesPreserveFields` to a comma-separated list of fields that you want to save.

When Content receives a duplicate document, it copies this field from the existing version of the document to the newer version when it performs `KillDuplicates`.

> **NOTE:** If there is more than one copy of the document in the Content index when a new version arrives, Content copies the preserve field from the existing duplicate with the highest document ID.

Enable Deduplication for Individual Index Jobs

To enable deduplication for individual indexing jobs, use the `KillDuplicates` action parameter in the `DREADD` or `DREADDDATA` index actions.

You can use the `KeepExisting` action parameter when directly indexing data into the IDOL Content component with deduplication to reduce the indexing load.

You can use the following action parameters to move duplicates to a specified database.

- `KillDuplicatesDB`

- `KillDuplicatesDBField`

- `KillDuplicatesMatchDBs`

- `KillDuplicatesMatchTargetDB`

- `KillDuplicatesPreserveFields`

You can use any of the deduplication options with `DREADD` and `DREADDDATA` actions. When you use either of these actions:

- The `KillDuplicates` setting specified in either action overrides the same setting in the
  `KillDuplicates` configuration parameter.

- If you do not specify the `KillDuplicates` action parameter with either of the actions, the setting
  in the `KillDuplicates` configuration parameter is used.

Related Topics

- [DREADD: Index IDX and XML Files Directly, on page 51](#)

- [DREADDDATA: Index Data over a Socket, on page 57](#)

- [Use KeepExisting to Minimize the Index Load, below](#)

- [DREADDDATA Parameters, on page 57](#)

- [Deduplication Options—KillDuplicates, on page 66](#)

Use KeepExisting to Minimize the Index Load

If you set `KillDuplicates` to **Reference** or *FieldName*, you can use the `KeepExisting` action
parameter to minimize the indexing load on Content when deduplicating.

Set `KeepExisting` to **True** to reverse normal deduplication and discard the document it has received
for indexing and keep the existing matching document that it already contains instead.

Enable Deduplication for Connector Index Jobs

If you use a connector to retrieve documents from a remote repository for indexing into the
IDOL Content component, you can configure the `KillDuplicates` configuration parameter for the CFS
to set deduplication.

- The options available for the `KillDuplicates` IDOL Content component configuration parameter
  are also available for the CFS configuration parameter.

- The same constraints for deduplication apply when you configure for deduplication using a
  `KillDuplicates` option for the CFS.

For more information on connector deduplication, refer to the *Connector Framework Server Reference*.

Related Topics

- [Deduplication Options—KillDuplicates, on page 66](#)
- [Deduplication Constraints, below](#)

Deduplication Constraints

There are some constraints on deduplication when using other IDOL parameters.

Use the Combine Operation

The IDOL Content component cannot use the same `ReferenceType` field for deduplication as it uses
for the `Combine` action parameter. The `Combine` operation occurs at query time and clashes with

deduplication. If you intend to deduplicate when indexing and use the `Combine` action parameter, you must set up separate `ReferenceType` fields for these processes.

Related Topics

Use Deduplication with DIH Reference-Based Indexing

You can enable the DIH for reference-based indexing. Refer to the *DIH Administration Guide*.

If you index documents into IDOL with the DIH enabled for reference-based indexing, it might prevent deduplication of documents with different references. In this case, use only one of the following deduplication options:

- `KillDuplicates=REFERENCE`

- `KillDuplicates=NONE`

Use Deduplication with DIH Field-Based Indexing

You can use field-based indexing in the DIH to ensure correct deduplication in a distributed system. For more information on configuring the DIH for field-based indexing, refer to the *DIH Administration Guide*.

If you set `KeepExisting` to **False**, or use `KillDuplicatesDB` options, it might prevent correct deduplication. To deduplicate correctly, you can distribute data by the `DeDupeHash` field (MD5 hash) of the documents. In this way, DIH sends all duplicates to the same child server. Setting `KillDuplicates` to **DeDupeHash** during the indexing action then ensures accurate deduplication.

To use a field for deduplication, you must configure it as a `ReferenceType` field. You do not need to configure it as `ReferenceType` in the DIH configuration file.

Deduplication of content occurs for all reference fields specified in a single `PropertyFieldCSVs` list in the IDOL Content component configuration file. To use only the `DeDupeHash` field to deduplicate, and not also the `DREREFERENCE`, you must set these reference fields in separate field processing sections in the IDOL Content component configuration file.

Add Metadata to Documents

When you index a document into the IDOL Content component, Content automatically stores all its metadata as fields. You can add additional fields to a document after you index it by running a `DREREPLACE` index action.

Related Topics

Check Index Status

You can check whether the indexing of data into the IDOL Content component is successful by entering the following URL into your Web browser:

`http://`*`Contenthost`*`:`*`port`*`/action=IndexerGetStatus`

where:

Contenthost	is the IP address or host name of the machine on which the IDOL Content component is installed.
port	is the IDOL Content component ACI port (the `Port` value specified in the `[Server]` section of IDOL Content component configuration file).

The `IndexerGetStatus` action displays the status of the IDOL Content component index queue.

> **TIP:** Alternatively, you can use the IDOL Admin interface. In the Monitor section, you can view the current queue of indexing jobs on the **Indexer Status** page. You can also view information about the index queue on the **Index Queue** tab in the Status section. For more information, refer to the *IDOL Admin User Guide*.

IndexerGetStatus Example

An `IndexerGetStatus` action is sent to the IDOL Content componentfollowing a `DREADD` index action. Content returns the following output:

```
<autnresponse>
<action>INDEXERGETSTATUS</action>
<response>SUCCESS</response>
<responsedata>
<timeformat>YYYY/MM/DD HH:NN:SS</timeformat>
<state>Inactive (no jobs)</state>
<item>
    <id>1</id>
    <origin_ip>127.0.0.1</origin_ip>
    <received_time>2015/03/31 16:14:43</received_time>
    <start_time>2015/03/31 16:14:44</start_time>
    <end_time>2015/03/31 16:16:44</end_time>
    <duration_secs>120</duration_secs>
    <percentage_processed>100</percentage_processed>
    <documents_processed>44710</documents_processed>
    <documents_deleted>0</documents_deleted>
    <status>-1</status>
    <description>Finished</description>
    <docidrange>1-44710</docidrange>
```

```
    <index_command>
    /DREADD?myfile.idx&KILLDUPLICATES=REFERENCE&DREDBNAME=Archive
    </index_command>
</item>
</responsedata>
</autnresponse>
```

where:

Tag Name	Description
`<timeformat>`	The time and date format that the response uses.
`<state>`	The status of the Content component indexing thread.
`<id>`	The ID number of the index action.
`<origin_ip>`	The IP address of the machine that sent the index action to Content.
`<received_time>`	The time that Content received the action.
`<start_time>`	The time that Content started processing the index action.
`<end_time>`	The time that Content finished processing the index action.
`<duration_secs>`	The total amount of time in seconds that Content spent processing the index action.
`<documents_processed>`	The number of documents that Content processed during the indexing job.
`<documents_deleted>`	The number of documents deleted during the indexing process.
`<status>`	The status code of the current status of the index action in the Content index queue.
`<description>`	The description of the `<status>` number.
`<docidrange>`	The range of `DocIDs` of documents that were processed during the index job.
`<index_command>`	The index action for the index job.

For the `DRECOMPACT` index action, `IndexerGetStatus` also returns a `<drecompact_status>` section to show the time that each stage of the compaction takes. This section contains `<stage>` tags, with the following attributes.

`name`	The name of the compaction stage.
`description`	The description of the compaction stage.
`percentage_complete`	The percentage of the compaction stage that is complete.

`time_seconds`	The time that the compaction stage has taken.
`paused`	If you pause the `DRECOMPACT` index action, this attribute is set to **True** for the paused stage.

The Content component returns the following status messages in the `<state>` tag to show whether it is currently processing jobs, or if it is paused:

Running
Paused
Paused (DREFLUSHANDPAUSE)
Paused (out of disk space)
Paused (waiting for flush lock)
Inactive (no jobs)

Related Topics

- IndexerGetStatus Status Codes, below

IndexerGetStatus Status Codes

Codes in **bold** are status messages. All other codes indicate that there is a problem with the indexing process.

Code	Message	Explanation
–1	Finished	The indexing process is complete.
–2	Out of disk space	Content ran out of disk space before it completed the indexing process.
–3	File not found	The index file could not be found.
–4	Database not found	The database into which you are trying to index could not be found.
–5	Bad parameter	The indexing action syntax is incorrect.
–6	Database exists	The database that you are trying to create already exists.
–7	Queued	The indexing action is queued and it is run when all preceding indexing actions are complete.
–8	Unavailable	Content is about to shut down or indexing is paused.
–9	Out of Memory	Content ran out of memory before the indexing

Code	Message	Explanation
		process could be completed.
–10	Interrupted	The indexing action was interrupted.
–11	XML is not well formed	Indexing failed because the XML is not well formed.
–12	Retrying interrupted command	Content is processing an index action that was previously interrupted.
–13	Backup in progress	Content is performing a backup.
–14	Max index size reached	The indexing job exceeds the maximum indexing size (your license determines the maximum indexing size).
–15	Max number of sections reached	The indexing job exceeds the maximum number of sections that you can index. Your license determines the number of sections that you can index.
–16	Indexing Paused	The indexing process was paused.
–17	Indexing Resumed	The indexing process was restarted.
–18	Indexing Cancelled	The indexing process was cancelled.
–19	Out of file descriptors	Content ran out of file descriptors.
–20	LanguageType not found	The language type of the index data could not be found.
–21	SecurityType not found	The security type of the index data could not be found.
–22	Child engines returned differing messages	The child servers returned different messages to the DIH. This code is reported by DIH only.
–23	Badly formatted index command	The indexing action was rejected by a child server because the syntax is not valid.
–25	To be sent to DRE	The index action is queued to be sent to the IDOL Content component. This code is reported by DIH only.
–26	`DREADDDATA`: Data received did not include `#DREENDDATA`	The data in the `DREADDDATA` action did not contain a `#DREEDNDDATA` statement indicating the end of the data.
–27	Command failed more times than the configured retry limit	The indexing action exceeded the maximum number of retries specified by the `MaximumRetries` parameter in the DIH configuration. This code is reported by DIH only.
–28	The index ID specified is invalid	The index ID returned by the child server is not valid. This code is reported by DIH only.

Code	Message	Explanation
–29	Command was redistributed to sibling engines as this engine was either unavailable or not accepting index jobs	The indexing action was sent to sibling servers because the child server was either unavailable or not accepting indexing jobs. This code is reported by DIH only.
–30	Database name too long	The name of the database in which you are indexing documents is too long. The length is defined internally as 63 characters.
–31	Command ignored due to id match	The `DREINITIAL` action was ignored because it did not match the ID specified in the `InitialID` parameter.
–33		The database cannot be created because the maximum number of databases was exceeded. The maximum is defined internally as 32,767.
–34	Pending commit	The indexing job is complete and the documents become available for searching after the next delayed sync cycle, which is specified in the `DelayedSync` parameter.
–35	Initializing	The indexing job is being started. This code is reported by DIH only.
–36	Reading IDX	The IDX file is being read from disk, prior to being sent to the DRE. This code is reported by DIH only.
–38	Processing in remote engine	The target engine of a `DREEXPORTREMOTE` operation is processing the exported documents.

NOTE: If the `IndexerGetStatus` action returns a positive number, this number indicates the percentage of the indexing queue that has been completed.

Tag Documents into Clusters

After indexing, you can tag documents into clusters of similar documents. Tagging can be useful for grouping duplicate documents together.

Use the index action `DRETAGDOCCLUSTERS`. This action takes the following parameters.

`TagField`	The full field name that contains document tags.
`MinScore`	The matching threshold to determine whether a document belongs to a cluster.
`TagSourceField`	The full field name to use as the source of the `TagField` value.

MinID	The first document ID to tag.
MaxID	The last document ID to tag.
CheckSumField	A reference field to use to determine whether a document is an exact match of another document.
TaggedDBName	The database which Content moves tagged documents to and retrieves tags from.
RelevanceField	The full field name that holds the relevance score of the document to its cluster.
DatabaseMatch	The names of databases that contain documents that you want to tag.
CheckSumDBs	The names of databases that you can checksum match against.
ClusterDBs	The names of databases that you can cluster against. This list includes TaggedDBName if specified.

DRETAGDOCCLUSTERS Example

The IDOL Content component indexes three documents:

```
#DREREFERENCE A
#DREDBNAME Default
#DREFIELD CHECKSUM="ABCD1234"
#DRECONTENT
apple banana cheese
#DREENDDOC

#DREREFERENCE B
#DREDBNAME Default
#DREFIELD CHECKSUM="ABCD1234"
#DRECONTENT
apple banana cheese
#DREENDDOC

#DREREFERENCE C
#DREDBNAME Default
#DREFIELD CHECKSUM="XYZ9876"
#DRECONTENT
apple banana data
#DREENDDOC
```

After indexing, you send the following action:

```
[...]/DRETAGDOCCLUSTERS?TagField=DOCUMENT/CLUSTERID&MinScore=60&TagSourceField=DOCU
MENT/DREREFERENCE&MinID=1&MaxID=3&CheckSumField=DOCUMENT/CHECKSUM&TaggedDBName=tagg
ed&RelevanceField=DOCUMENT/CLUSTERSCORE
```

The IDOL Content component modifies the data:

```
#DREREFERENCE A
#DREDBNAME Tagged
#DREFIELD CHECKSUM="ABCD1234"
#DREFIELD CLUSTERID="A"
#DREFIELD CLUSTERSCORE="100.00"
#DRECONTENT
apple banana cheese
#DREENDDOC

#DREREFERENCE B
#DREDBNAME Tagged
#DREFIELD CHECKSUM="ABCD1234"
#DREFIELD CLUSTERID="A"
#DREFIELD CLUSTERSCORE="100.00"
#DRECONTENT
apple banana cheese
#DREENDDOC

#DREREFERENCE C
#DREDBNAME Tagged
#DREFIELD CHECKSUM="XYZ9876"
#DREFIELD CLUSTERID="A"
#DREFIELD CLUSTERSCORE="70.00"
#DRECONTENT
apple banana data
#DREENDDOC
```

A is tagged as A because it does not match any existing clusters.

B is tagged as A because its CHECKSUM field matches A's.

C is tagged as A because it is similar to A and has a score higher than the specified MinScore (60).

Chapter 4: Fields

Both document content and metadata are stored in the IDOL Content component as fields. Retrieving data means retrieving the values of one or more fields. This section describes how to set up and use fields.

About Fields

Data passes to the IDOL Content component (for example, from IDOL connectors) in the form of IDX or XML fields. Content stores all the fields that it receives so that you can search any field by using `FieldText` queries. To optimize Content performance, you can specify how it processes and stores the fields it receives.

You can associate fields with special properties in the IDOL Content component configuration file. For example, you can instruct Content to treat these fields (or documents that contain them) in a specific way or read specific information from them.

You can associate a field with more than one property, as long as the properties do not clash.

You can assign the following properties to fields.

`ACLType`	Fields that hold access control lists (ACLs).
`AlwaysMatchType`	Fields that queries always match when they are present with a non-empty value.
`AutnRankType`	Fields that hold the document rank.
`BitFieldCompressed`	The index for `BitFieldType` fields is compressed.

`BitFieldMaxMemoryKB`	The maximum memory (in KB) to allocate for each associated `PropertyFieldCSVs` field that has the `BitFieldType` property.
`BitFieldType`	Fields that hold information on document sets. See BitFieldType Fields, on page 103.
`CountType`	The number of occurrences of the associated fields are stored in a fast look-up table in memory to optimize matching of the fields when you use the `MATCHCOVER` and `EQUALCOVER` `FieldText` specifiers.
`DatabaseType`	Fields that hold the database that documents belongs to.
`DateType`	Fields that hold the date of documents.
`DocumentTrackingType`	Fields that hold the tracking IDs of documents.
`ExpireAfterDelay`	An offset, in hours, to add to the expiration date in the associated `ExpireDateType` field.
`ExpireDateType`	Fields that hold the expiration dates of documents.
`FieldCheckType`	A field that occurs in a large number of documents and holds a value that is frequently used to restrict query results.
`FlattenIndexType`	Fields that originate from hierarchically structured documents and whose content is stored as one level.
`GeospatialType`	Fields that contain geospatial (location) information.
`HiddenType`	Fields whose content is hidden.
`HighlightType`	If fields contain terms that match a query, these terms are highlighted. See Highlight Fields, on page 102.
`Index`	Fields that are stored as Index fields. See Index Fields, on page 92.
`IndexNumbers`	Restrict the numbers to index for `IndexNumbersType` fields.
`IndexNumbers1MaxLength`	Restrict the length of pure numeric terms for `IndexNumbersType` fields.
`IndexNumbers2MaxLength`	Restrict the length of alphanumeric terms for `IndexNumbersType` fields.
`IndexNumbersType`	Fields to index as numeric or mixed-alphanumeric fields.
`InvertedAgentType`	Fields that are contained within inverted agents.
`LangDetectType`	Fields to use for automatic language detection when `AutoDetectLanguagesAtIndex` is set to **True**.
`LanguageType`	Fields that hold the language type of documents.

MatchType	Fields to store in a fast look-up table in memory to optimize matching of the fields when you use the following FieldText specifiers: ARANGE, BIASVAL, EQUALCOVER, MATCH, MATCHALL, MATCHCOVER, STRING, WILD.
MemCachedType	Fields to store in a memory cache.
NonReversibleType	Fields whose content is not line-reversed on index, even if the document is detected as being right-to-left Arabic or Hebrew.
NumericDateType	Fields that contain numeric dates and are stored in a fast look-up table in memory to optimize matching of the fields when you use FieldText specifiers. See NumericDateType Fields, on page 95.
NumericIntegerOnly	Fields with the NumericType property store signed 64-bit integer-only values, rather than doubles.
NumericNormalMaxMem	The maximum memory (in KB) to allocate for each associated PropertyFieldCSVs field that has one of these properties: NumericDateType, NumericType, ParametricType.
NumericType	Fields that contain numeric data and are stored in a fast look-up table in memory to optimize matching of the fields when you use FieldText specifiers. See NumericType Fields, on page 96.
OcrFilterType	Fields to evaluate by the OCR filter, and not to index or store if its quality is unsatisfactory (the field is stored, but is empty).
ParametricType	Fields that hold parametric values.
ParametricRangeType	Fields that contain numeric values to use to generate numeric ranges for parametric searches.
PrintType	Fields whose content is displayed with results when you set the query action Print parameter to **Fields**.
Ranges	The number of ranges to use for ParametricRangeType fields.
ReferenceMemoryMappedType	Fields that hold a value that is an existing value in a different ReferenceType field. This is then used in combination with the FieldRecurse action parameter and the field specifier MATCHRECURSE.
ReferenceType	Fields that hold document references. See ReferenceType Fields, on page 99.
SectionBreakType	Fields that hold the section number of documents that were split by the Import module.
SecurityType	The security type of documents that contain associated fields.
SortType	Fields to store for fast sorting when you use the ARANGE FieldText specifier or an alphabetical Sort option when querying Content.

	NOTE: Micro Focus recommends that you optimize `ARANGE` by using the `MatchType` property, rather than `SortType`.
`SourceType`	Fields to use to generate summaries and to suggest conceptually similar documents.
`SynonymType`	Fields that hold the name of the synonym job whose settings apply to documents that contain associated fields.
`TextParseIndexType`	Fields to treat as Index fields when sending a query using the `TextParse` and `AgentBooleanField` parameters.
`TitleType`	Fields that hold document titles.
`TrimSpaces`	Fields from which to remove multiple, leading, or trailing spaces before storing in Content.
`Weight`	The factor by which the weight of terms in associated fields is increased if they match query terms.

NOTE: You cannot configure a field with more than one numeric-based type concurrently. Numeric-based types include `NumericType`, `NumericDateType`, `MatchType`, `ParametricRangeType`, `ReferenceMemoryMappedType`, and `ParametricType` when the `ParametricNumericMapping` configuration parameter is set to **True**.

If your configuration file contains a field with a conflicting configuration, the server uses an internal precedence to set the field property. It flags the configuration conflict in the logs, and in configuration validation.

You can view information about the indexing fields that are defined for the documents in the data index on the **Field Types** page in the Monitor section of the IDOL Admin interface. For more information, refer to the *IDOL Admin User Guide*.

Related Topics

Configure a Field Process

The `[FieldProcessing]` section in the IDOL Content component configuration file allows you to identify particular fields in documents. You can then apply any type of processing to them or the document that contains them during the indexing process, depending on the field value.

In this way you can apply multiple processes to documents without needing to set up a configuration section for each process combination.

> **NOTE:** When identifying fields, use the following formats:
>
> - `/FieldName` to match root-level fields.
>
> - `*/FieldName` to match all fields except root-level.
>
> - `/Path/FieldName` to match fields that the specified path points to.
>
> Field names must not contain spaces, accents, or multibyte characters, and they must not start with a number. For IDX documents, Content converts these text elements to underscores (_) when it indexes the fields. You must also change any queries that reference these field names to use the modified field name.

To apply processes to fields or documents that contain specific fields

1. Open the IDOL Content component configuration file in a text editor.

2. In the `[FieldProcessing]` section, list the processes to apply to fields. For example:

   ```
   [FieldProcessing]
   0=MyFirstProcess
   1=IndexFields
   2=MyCombinedProcess
   3=IndexAndWeightHigher
   ```

3. Create a section for each process that you listed. In each section, declare a property for the process (you define the property later by setting one or more applicable configuration parameters). Identify the fields to associate with the processes.

   You can use the `PropertyMatch` parameter to identify a specific value that fields must have to be processed. (This is useful if you set up a process that identifies security or language fields.)

   > **NOTE:** The properties that you create must not have the same name as the processes.

   For example:

   ```
   [MyFirstProcess]
   Property=MyFirstProperty
   PropertyFieldCSVs=*/MyField,*/MySecondField
   PropertyMatch=*myString*

   [IndexFields]
   Property=MySecondProperty
   PropertyFieldCSVs=*/DRECONTENT,*/DRETITLE

   [MyCombinedProcess]
   Property=MyCombinedProperty
   PropertyFieldCSVs=*/MyDateField,*/MyIndexField

   [IndexAndWeightHigher]
   Property=IndexHigherWeight
   PropertyFieldCSVs=*/SUMMARIES
   ```

4. Create a section for each of the properties and specify appropriate configuration parameters for each. These configuration parameters define the processes that are applied to all the fields (or all documents that contain the fields) that you previously associated with the processes.

For example:

```
[MyFirstProperty]
HiddenType=True

[MySecondProperty]
Index=True

[MyCombinedProperty]
DateType=True
Index=True

[IndexHigherWeight]
Index=True
Weight=2
```

Related Topics

- Display Online Help, on page 30

Example:

```
[FieldProcessing]
0=IndexFields
1=IndexAndWeightHigher
2=SectionBreakFields
3=DateFields
4=DatabaseFields
5=SetReferenceFields

[IndexFields]
// Controls which fields are indexed
Property=Index
PropertyFieldCSVs=*/DRECONTENT,*/DRETITLE

[IndexAndWeightHigher]
// Fields to index with a weight
Property=IndexWeight
PropertyFieldCSVs=*/SUMMARIES

[SectionBreakFields]
// Field containing document section number
Property=Section
PropertyFieldCSVs=*/DRESECTION

[DateFields]
// Fields containing the document date
Property=Date
```

```
PropertyFieldCSVs=*/DREDATE,*/harvest_time

[DatabaseFields]
// CSV of field names that define the document database
Property=Database
PropertyFieldCSVs=*/DREDBNAME

[SetReferenceFields]
// CSV of fields that define the document URL
Property=Reference
PropertyFieldCSVs=*/DREREFERENCE,*/DRETITLE


//--------------------------Properties---------------------//

[Index]
Index=True

[IndexWeight]
Index=True
Weight=2

[Section]
SectionBreakType=True

[Date]
DateType=True

[Database]
DatabaseType=True

[Reference]
ReferenceType=True
TrimSpaces=True
```

Update Field Configuration

The field processes that you configure affect how the IDOL Content component processes data at index time. As a result, if you want to change the configuration after you have indexed data, there are often additional steps required to update your index.

There are three methods that you can use to update your field configuration:

- Manually update the configuration file, and restart the server. In this case, you can use the `Regenerate*Index` configuration parameters, where applicable, to automatically regenerate the modified indexes when you restart the server.

- Use the `DREREGENERATE` index action to modify the field configuration, and run any index regeneration at the same time.

- Use the IDOL Admin interface. This option provides a user interface for the DREREGENERATE index action.

These methods are the same as the methods that you can use to regenerate an index after a validation failure. For more information, see Repair an Index After Validation Fails, on page 445.

You can make many field configuration changes using any of these methods. The DREREGENERATE index action allows you to make these configuration updates without restarting your server. However, for some changes you must reindex your content to make configuration changes. The following table describes how to update the field configuration for a particular property type.

Field type	Update method
ACLType	Requires reindex
AlwaysMatchType	Requires reindex
AutnRankType	Requires reindex
BitFieldType (and BitFieldCompressed, BitFieldMaxMemoryKB)	RegenerateBitFieldIndex or DREREGENERATE with Type=BitField.
CountType	RegenerateCountIndex or DREREGENERATE with Type=Count.
DatabaseType	Requires reindex
DateType	Requires reindex
DocumentTrackingType	Requires reindex
ExpireDateType (and ExpireAfterDelay)	Requires reindex
FieldCheckType	Requires reindex
FlattenIndexType	Requires reindex
GeospatialType	RegenerateGeospatialIndex or DREREGENERATE with Type=Geospatial.
HiddenType	No additional regeneration required
HighlightType	No additional regeneration required
Index	Requires reindex
IndexNumbers (and IndexNumbersNMaxLength, IndexNumbersType)	Requires reindex
InvertedAgentType	Requires reindex

Field type	Update method
LangDetectType	Requires reindex
LanguageType	Requires reindex
MatchType	RegenerateMatchIndex or DREREGENERATE with Type=Match.
MemCachedType	You must restart the server to change the configuration
NonReversibleType	No additional regeneration required
NumericDateType	RegenerateNumericDateIndex or DREREGENERATE with Type=NumericDate.
NumericType (and NumericIntegerOnly, NumericNormalMaxMem)	RegenerateNumericIndex or DREREGENERATE with Type=Numeric.
OcrFilterType	Requires reindex
ParametricType	RegenerateParametricIndex or DREREGENERATE with Type=Parametric.
ParametricRangeType (and Ranges)	RegenerateParametricIndex or DREREGENERATE with Type=Parametric.
PrintType	No additional regeneration required
ReferenceMemoryMappedType	Requires reindex. You can regenerate this index to restore parent-child relationships for documents that were indexed out of order, but not to change the configuration. For more information, refer to the *IDOL Server Reference*.
ReferenceType	Requires reindex. You can regenerate this index after a validation failure, but not to change configuration. See Repair an Index After Validation Fails, on page 445.
SectionBreakType	Requires reindex
SecurityType	Requires reindex. You can regenerate this index after a validation failure, but not to change configuration. See Repair an Index After Validation Fails, on page 445.
SortType	RegenerateSortIndex or DREREGENERATE with Type=Sort.
SourceType	Requires reindex
SynonymType	Requires reindex

Field type	Update method
TextParseIndexType	Requires reindex
TitleType	Requires reindex
TrimSpaces	Requires reindex
Weight	You must restart the server to change the configuration

NOTE: If you attempt to use the DREREGENERATE index action to change a property that requires reindexing, the DREREGENERATE index action returns an error response and does not make the change.

When you use the DREREGENERATE index action to update the field configuration, you can set the Type parameter to **Auto** to automatically regenerate the indexes for the fields that you have changed. You can also set Type to **None** if you do not want to regenerate the indexes immediately, for example so that you can make a series of field configuration changes and then regenerate the indexes in an additional index action.

Related Topics

- Repair an Index After Validation Fails, on page 445

Update Fields in the Configuration File

Use the following procedure to update fields in the configuration file.

NOTE: You must use this method for changes where you must reindex content for the changes to take effect.

To update field configurations in the configuration file

1. Open the IDOL Content component configuration file in a text editor.

2. Find the field configuration section that you want to modify.

3. Modify any of the configuration parameters that you want to change. For details of the configuration parameters, refer to the *IDOL Server Reference*.

4. (Optional) If you are modifying a field type that can be regenerated, find the [Server] section, and set the appropriate Regenerate*Index configuration parameter to **True**. For example, to update the configuration for MatchType fields, set the RegenerateMatchIndex parameter to **True**.

    You can alternatively skip this step and run a DREREGENERATE index action after you restart the server.

5. Save and close the configuration file.

6. Restart the IDOL Content component for your changes to take effect.

7. Update your content:

- If you have modified a field type where the change requires you to reindex your data, reindex your data.

- If you need to regenerate the index, run a `DREREGENERATE` index action with `Type` set to the appropriate index. For example:

  `http://idolhost:9001/DREREGENERATE?Type=Match`

  For more information, refer to the *IDOL Server Reference.*

8. If you used the `Regenerate*Index` configuration parameters, set the parameters to **False** again in the configuration file. This step means that the server does not waste time by regenerating the index every time you restart the server.

Update Field Configuration with an Index Action

Use the following procedure to make field configuration changes for fields where you can regenerate the content. This method allows you to update the field configuration without restarting the server, which avoids downtime.

> **NOTE:** You can update the lists of fields associated with a particular property, but if you want to add a new field process or add an additional property to an existing process, you must update the configuration file manually.

To update field configurations with an index action

1. Open the IDOL Content component configuration file in a text editor.

2. Find the field configuration section that you want to modify.

3. Send a `DREREGENERATE` index action to the IDOL Content component, with the `FieldProcessingSection` parameter set to the name of the configuration section that you want to modify. Set any of the following parameters to update the lists of fields:

   - `AugmentFieldCSVs`. A list of fields that you want to add to the `PropertyFieldCSVs` parameter.

   - `DiminishFieldCSVs`. A list of fields that you want to remove from the `PropertyFieldCSVs` parameter.

   - `AugmentNegativeFieldCSVs`. A list of fields that you want to add to the `PropertyNegativeFieldCSVs` parameter.

   - `DiminishNegativeFieldCSVs`. A list of fields that you want to remove from the `PropertyNegativeFieldCSVs` parameter.

   - `Type`. Set this parameter to **None** if you do not want to regenerate the index immediately, for example if you want to modify several field configuration sections and then run a single regenerate operation. The default value is `Auto`, which automatically regenerates the index for the field configuration that you modify.

     > **NOTE:** If you set `Type` to **None**, you must run a `DREREGENERATE` index action manually to regenerate the indexes that you have modified. The `DREREGENERATE` index action does not automatically check all field configurations that might need regeneration.

For example:

```
DREREGENERATE?FieldProcessingSection=SetMatchFields&AugmentFieldCSVs=*/NewMatch
Field,*/SpecialMatch&DiminishFieldCSVs=*/ExistingMatchField
```

This example updates the [SetMatchFields] configuration section, adding NewMatchField and SpecialMatch to the PropertyFieldCSVs, and removing ExistingMatchField. The DREREGENERATE index action automatically regenerates the Match index to make the configuration changes available.

Update Field Configuration with IDOL Admin

Use the following procedure to update your field configuration by using the IDOL Admin interface. IDOL Admin uses the DREREGENERATE index action.

To update your field configuration by using IDOL Admin

1. In the **Service Control** tab in the Console page, click **Regenerate**.

   The Regenerate dialog box opens.

2. In the **Type** list, click the field type that you want to update.

3. Select a priority for the index action. This determines how Content queues the action.

4. Click **Regenerate**.

You can monitor the progress of the DREREGENERATE action in the Recent Tasks panel.

Index Fields

Store fields that contain text which you want to query frequently as Index fields. Index fields are processed linguistically when they are stored in IDOL Server. This means that stemming and stop word lists are applied to text in Index fields before they are stored, which allows IDOL Server to process queries for these fields more quickly. Typically DRETITLE and DRECONTENT are fields that are set up as Index fields.

Do not use the Index field type for fields that contain:

- URLs or content that you are unlikely to use.

- content that you query frequently, but whose values you query only in their entirety. It is more efficient to use field specifiers (for example, MATCH) to query these values.

- numeric values or dates as Index fields. Instead, store these fields as numeric fields and numeric date type fields.

Related Topics

To set up Index fields

1. Open the IDOL Server configuration file in a text editor.

2. List an indexing process in the `[FieldProcessing]` section. For example:

   ```
   [FieldProcessing]
   0=MyFirstProcess
   1=MySecondProcess
   2=IndexingFields
   ```

3. Create a section for the indexing process, and in each section, create a property for the process (you define the property later by setting one or more applicable configuration parameters). Identify the fields that you want to associate with the process.

   You can use the `PropertyMatch` parameter to identify a specific value that fields must have to be processed.

   > **NOTE:** The properties that you create must not have the same name as the processes.

   For example:

   ```
   [MyFirstProcess]
   Property=MyFirstProperty
   PropertyFieldCSVs=*/MyField,*/MySecondField
   PropertyMatch=*myString*

   [MySecondProcess]
   Property=MySecondProperty
   PropertyFieldCSVs=*/MyOtherField,*/MyOtherSecondField

   [IndexingFields]
   Property=IndexFields
   PropertyFieldCSVs=*/DRECONTENT,*/DRETITLE
   ```

4. Create a section for your indexing property in which you set the `Index` parameter to **True**. For example:

   ```
   [MyFirstProperty]
   HiddenType=True

   [MySecondProperty]
   Index=True

   [IndexFields]
   Index=True
   ```

5. Save and close the configuration file. Restart IDOL Server for your changes to take effect.

Configure the Number Index Process

You can configure the IDOL Content component to index numeric and alphanumeric fields in several ways by using the configuration parameter `IndexNumbers`. Set `IndexNumbers` to one of the following values to specify how Content treats numbers:

0	Numbers are not indexed.
1	All numbers are indexed (irrespective of whether they appear on their own or as part of a word).
2	Numbers are indexed only if they are part of a word (for example DRE4, Y2K and so on).

To restrict this to a narrower set of data, use the field processing property `IndexNumbersType`. Create an `IndexNumbersFields` section and specify which fields qualify. You can limit only terms that are indexed within the field property.

For example:

```
[English]
IndexNumbers=2

[IndexNumbersFields]
PropertyFieldCSVs=*/MYFIELD
Property=IndexNumbers

[IndexNumbers]
IndexNumbersType=True
IndexNumbers=2
```

This means that Content indexes the numeric terms in `*/MYFIELD` that satisfy `IndexNumbers=2` (non-numeric and mixed-alphanumeric), whereas all other fields index with `IndexNumbers=1` (all numeric terms).

> **NOTE:** If the `IndexNumbers` configuration parameter is not specified in a property section, its default is `0`.

You can also limit the indexing of numeric or mixed-alphanumeric terms by the length of the term.

For example:

```
[IndexNumbers1]
IndexNumbersType=True
IndexNumbers=1
IndexNumbers1MaxLength=5
IndexNumbers2MaxLength=6
```

This means that fields with this property have all numbers indexed, assuming the language has `IndexNumbers=1` configured, except for pure numeric terms longer than five characters, which are not indexed. Alphanumeric terms longer than six characters are also not indexed.

> **NOTE:** You cannot set the length to more than 255.

NumericDateType Fields

You can configure the IDOL Content component to identify fields that contain dates. When these fields are indexed, Content stores them in a fast look-up table in memory, so that it can quickly return the fields.

> **NOTE:** You cannot configure a field with two numeric-based types concurrently. Numeric-based types include `NumericType`, `NumericDateType`, `MatchType`, `ParametricRangeType`, `ReferenceMemoryMappedType`, and `ParametricType` when the `ParametricNumericMapping` configuration parameter is set to **True**.

Content converts dates to numerical values (epoch seconds) and identifies the fields that contain the numerical date values.

To set up memory mapping for NumericDateType fields

1. Open the IDOL Content component configuration file in a text editor.

2. List a process that identifies numeric date fields in the `[FieldProcessing]` section. For example:

   ```
   [FieldProcessing]
   0=MyFirstProcess
   1=NumericDateFields
   ```

3. Create a section for each process that you listed, and in each section, create a property for the process (you define the property by setting one or more applicable configuration parameters). Identify the fields that you want to associate with the process.

   > **NOTE:** The properties that you create must not have the same name as the processes.

   For example:

   ```
   [MyFirstProcess]
   Property=MyProperty
   PropertyFieldCSVs=*/MyField,*/MyOtherField

   [NumericDateFields]
   Property=NumDate
   PropertyFieldCSVs=*/BIRTHDAY,*/STARTDATE
   ```

4. Create a section for the property in which you set the `NumericDateType` parameter to **True**. This enables Content to memory map the associated `PropertyFieldCSVs` fields, and identify them as fields that contain date values. For example:

   ```
   [NumDate]
   NumericDateType=True
   ```

5. Save and close the configuration file.

6. Restart the IDOL Content component for your changes to take effect.

If you now send a query for a specific value that is stored in the BIRTHDAY field, Content memory maps the range that this value is in, so that it can return results more quickly next time a value that lies in this range is queried.

For example:

```
http://12.3.4.56:4000/action=Query&FieldText=RANGE{01/01/1980,31/12/1980}:BIRTHDAY
```

The BIRTHDAY field must contain a numeric date value that is between 01/01/1980 and 31/12/1980 for this document to be returned.

NumericType Fields

You can configure the IDOL Content component to identify fields that contain numerical values. When these fields are indexed, Content stores them in a fast look-up table in memory, so that it can quickly return the field. A numeric field can contain a comma-separated list of numbers. Content stores each value as a numeric value for this field, for this document.

> **NOTE:** You cannot configure a field with two numeric-based types concurrently. Numeric-based types include NumericType, NumericDateType, MatchType, ParametricRangeType, ReferenceMemoryMappedType, and ParametricType when the ParametricNumericMapping configuration parameter is set to **True**.

To set up NumericType fields to speed numeric queries

1. Open the IDOL Content component configuration file in a text editor.

2. List a process that identifies numeric fields in the [FieldProcessing] section. For example:

```
[FieldProcessing]
0=MyFirstProcess
1=PriceFields
```

3. Create a section for each process that you listed, and in each section, declare a property for the process (you define the property later by setting one or more applicable configuration parameters). Identify the fields that you want to associate with the process.

> **NOTE:** The properties that you create must not have the same name as the processes.

For example:

```
[MyFirstProcess]
Property=MyProperty
PropertyFieldCSVs=*/MyField,*/MyOtherField


[PriceFields]
```

```
Property=Price
PropertyFieldCSVs=*/PRICE
```

4. Create a section for the property in which you set the `NumericType` parameter to **True**. This enables Content to memory map the associated `PropertyFieldCSVs` fields. For example:

```
[Price]
NumericType=True
```

5. Save and close the configuration file.

6. Restart the IDOL Content component for your changes to take effect.

If you now send a query for a specific value that is stored in the `PRICE` field, Content memory maps the range that this value is in, so that it can return results more quickly next time a value that lies in this range is queried.

Examples:

```
http://12.3.4.56:4000/action=Query&FieldText=NRANGE{50,100}:PRICE
```

The `PRICE` field must contain a number between 50 and 100 (including decimal numbers) for this document to return.

```
http://12.3.4.56:4000/action=Query&Text=computer&Sort=PRICE:numberincreasing
```

IDOL Server sorts the results that it returns for the query according to the values that their `PRICE` fields contain. The results whose `PRICE` field contains the smallest value is listed first, followed by results with increasing values in the `PRICE` field.

FieldCheckType Fields

You can configure the IDOL Content component to identify a field contained in a large number of documents whose entire value is frequently used to restrict results (for example, a field that stores category names). When this field is indexed, Content stores it in a fast look-up table in memory, so that it can quickly return the field.

> **NOTE:** If you set `URLAnalysis` to **True** in the `[Server]` section of the IDOL Content component configuration file, you cannot identify a field as a `FieldCheckType` field, because Content automatically uses the domain it finds in `ReferenceType` fields as the FieldCheck value.

To set up FieldCheckType fields

1. Open the IDOL Content component configuration file in a text editor.

2. List a process that identifies numeric fields in the `[FieldProcessing]` section. For example:

```
[FieldProcessing]
0=MyFirstProcess
1=FieldCheckTypeIdentification
```

3. Create a section for each process that you listed, and in each section, create a property for the process (you define the property later by setting one or more applicable configuration

parameters). Identify the fields that you want to associate with the process.

> **NOTE:** The properties that you create must not have the same name as the processes.

For example:

```
[MyFirstProcess]
Property=MyProperty
PropertyFieldCSVs=*/MyField,*/MyOtherField

[FieldCheckTypeIdentification]
Property=FieldCheck
PropertyFieldCSVs=*/CATEGORY
```

4. Create a section for the property in which you set the `FieldCheckType` parameter to **True**. This enables Content to memory map the associated `PropertyFieldCSVs` fields. For example:

```
[FieldCheck]
FieldCheckType=True
```

5. Save and close the configuration file.

6. Restart the IDOL Content component for your changes to take effect.

When you now use a `Query`, `Suggest`, or `SuggestOnText` action to query for results, you can:

- Use the `Combine` action parameter to restrict the result output to the most relevant result for each available `FieldCheckType` field value (by setting it to **FieldCheck**).

- Use the `FieldCheck` action parameter to restrict the result output to documents whose `FieldCheckType` field matches a specific value (this is also available for the `GetQueryTagValues` action).

Combine Parameter Example

In this example, Content is configured to store the `Category` field as a `FieldCheckType` field.

The following query is sent to Content.

```
http://12.3.4.56:4000/action=Query&Text=The best thing to do in your spare time&Combine=FieldCheck
```

If Content contains 50 documents that match the query text, of which eight contain a `Category` field with the value `Sport`, five contain a `Category` field with the value `Gardening`, and one contains a `Category` field with the value `Cooking`, the above query returns only three results:

- The most relevant of the documents whose `Category` contains the value `Sport`.

- The most relevant of the documents whose `Category` contains the value `Gardening`.

- The document whose `Category` contains the value `Cooking`.

FieldCheck Parameter Example

In this example, Content is configured to store the `Color` field as a `FieldCheckType` field.

The following query is sent to Content.

```
http://12.3.4.56:4000/action=Query&Text=A fast sports car&FieldCheck=Red
```

This query returns only results whose content matches the specified `Text` and whose `FieldCheckType` field has the value `Red`.

ReferenceType Fields

`ReferenceType` fields are used to identify documents. Before you index a document into the IDOL Content component, you have to set up a field process that determines which of the fields in a document are used as its `ReferenceType` field (note that a document can have multiple `ReferenceType` fields).

At index time, you can use `ReferenceType` fields to eliminate duplicate copies of documents. At query time, you can use `ReferenceType` fields to filter results (for example, by using the `Combine` action parameter or by specifying references that results must or must not match).

Note that if you want to eliminate duplicate document copies and use the `Combine` action parameter, you must set up separate `ReferenceType` fields for these processes.

Related Topics

- Prevent Duplicate Documents, on page 66
- Combine Parameter, on page 353
- Use KillDuplicates and Combine on ReferenceType Fields, on the next page

Set up ReferenceType Fields

You must set up a field process to identify `ReferenceType` fields before you start indexing documents into the IDOL Content component.

To set up ReferenceType fields

1. Open the IDOL Content component configuration file in a text editor.

2. In the `[FieldProcessing]` section, add a process that identifies `ReferenceType` fields. For example:

   ```
   [FieldProcessing]
   0=MyFirstProcess
   1=MySecondProcess
   2=SetReferenceFields
   ```

3. Create a section for the process that you added, and in each section create a property for the process (you define the property later by setting one or more applicable configuration parameters). Identify the fields that you want to associate with the process.

   > **NOTE:** The properties that you create must not have the same name as the processes.

   For example:

   ```
   [MyFirstProcess]
   Property=MyFirstProperty
   PropertyFieldCSVs=*/MyField,*/MySecondField

   [MySecondProcess]
   Property=MySecondProperty
   PropertyFieldCSVs=*/MyThirdField

   [SetReferenceFields]
   Property=Reference
   PropertyFieldCSVs=*/DREREFERENCE,*/URL
   ```

4. Create a section for each of the properties and specify appropriate configuration settings for each. These configuration parameters define the processes that are applied to all the fields (or all documents that contain the fields) that you previously associated with the processes. For example:

   ```
   [MyFirstProperty]
   HiddenType=True

   [MySecondProperty]
   Index=True

   [Reference]
   ReferenceType=True
   TrimSpaces=True
   ```

5. Save and close the configuration file.

6. Restart the IDOL Content component for your changes to take effect.

You can now index documents into IDOL Server.

> **NOTE:** If you do not set up a field process that identifies `ReferenceType` fields, Content automatically allocates a unique number to each document that is indexed. Content uses this number as the reference for the document.

Use KillDuplicates and Combine on ReferenceType Fields

When you instruct the IDOL Content component to eliminate duplicate document copies at index time using a specific `ReferenceType` field (by setting the `KillDuplicates` parameter in the IDOL Content component configuration file), it automatically uses any field listed for `PropertyFieldCSVs` alongside

this `ReferenceType` field in the IDOL Content component configuration to eliminate duplicate document copies as well.

However, Content cannot use the same field for deduplication as for the `Combine` action parameter, because the `Combine` operation clashes (carried out at query time) with Content eliminating duplicate fields. This clash means that, if you want to eliminate duplicate document copies and use the `Combine` action parameter, you must set up separate `ReferenceType` fields for these processes.

Related Topics

- Prevent Duplicate Documents, on page 66

To use KillDuplicates and Combine on ReferenceType fields

1. Open the IDOL Content component configuration file in a text editor.

2. In the `[FieldProcessing]` section, add two processes that identify `ReferenceType` fields (note that you must set up a field process to identify `ReferenceType` fields before you start indexing documents into Content). One of them is used to eliminate duplicate copies of documents, and the other one is used for the `Combine` operation.

   For example:

   ```
   [FieldProcessing]
   0=MyFirstProcess
   1=MySecondProcess
   2=SetUpReferenceFields
   3=SetUpMoreReferenceFields
   ```

3. Create a section for the processes that you added, and in each section, create a property for the respective process (you define the property later by setting one or more applicable configuration parameters). Identify the fields that you want to associate with each process.

   > **NOTE:** The properties that you create must not have the same name as the processes.

   For example:

   ```
   [MyFirstProcess]
   Property=MyFirstProperty
   PropertyFieldCSVs=*/MyField,*/MySecondField

   [MySecondProcess]
   Property=MySecondProperty
   PropertyFieldCSVs=*/MyThirdField

   [SetUpReferenceFields]
   Property=ReferenceFields
   PropertyFieldCSVs=*/DREREFERENCE,*/URL

   [SetUpMoreReferenceFields]
   Property=MoreReferenceFields
   PropertyFieldCSVs=*/DRETITLE
   ```

4. Create a section for each of the properties and specify appropriate configuration settings for

each. These configuration parameters define the processes that are applied to all the fields (or all documents that contain the fields) that you previously associated with the processes. For example:

```
[MyFirstProperty]
HiddenType=True

[MySecondProperty]
Index=True

[ReferenceFields]
ReferenceType=True
TrimSpaces=True

[MoreReferenceFields]
ReferenceType=True
TrimSpaces=True
```

5. Save and close the configuration file.

6. Restart the IDOL Content component for your changes to take effect.

After you index documents into Content, you can use, for example, the */DREREFERENCE field to eliminate duplicate copies of documents. (Content then automatically also uses the */URL field for deduplication because it is listed alongside */DREREFERENCE for PropertyFieldCSVs.) This leaves you free to use the */DRETITLE field for the Combine operation.

Highlight Fields

When you run a Query, Suggest, or SuggestOnText action, you can highlight sentences or words in the results that are related to the terms in the query (or the terms in the text or document that you are suggesting on).

The IDOL Content component checks which fields highlighting applies to and then highlights all sentences or words that are based on the terms in the results that it returns.

To set up highlight fields

1. Open the IDOL Content component configuration file in a text editor.

2. List a highlighting process in the [FieldProcessing] section. For example:

```
[FieldProcessing]
0=MyFirstProcess
1=HighlightFields
```

3. Create a section for each process that you listed, and in each section, create a property for the process (you define the property later by setting one or more applicable configuration parameters). Identify the fields that you want to associate with the process.

    NOTE: The properties that you create must not have the same name as the processes.

For example:

```
[MyFirstProcess]
Property=MyProperty
PropertyFieldCSVs=*/MyField,*/MyOtherField

[HighlightFields]
Property=Highlight
PropertyFieldCSVs=*/DRETITLE,*/DRECONTENT
```

4. Create a section for the property in which you set the `HighlightingType` parameter to **True**. This enables the highlighting of all matched terms that are contained in the associated `PropertyFieldCSVs` fields. For example:

```
[Highlight]
HighlightType=True
```

5. Save and close the configuration file.

6. Restart the IDOL Content component for your changes to take effect.

> **NOTE:** To use document highlighting with the IDOL View component, you must also specify the IDOL Content component host and port in the `[Server]` section of the IDOL View component configuration file.

BitFieldType Fields

If you have documents that can be part of several different document sets, you can use `BitFieldType` fields to efficiently store information on which sets the documents belong to.

The value in a `BitFieldType` field is a hexadecimal number, which in turn represents a binary number. The binary number is a representation of the sets that a document belongs to, with each binary digit representing a particular set of documents. If a document is part of a set, the bit corresponding to that set is a **1**. If a document is not part of that set, the bit is a **0**.

For example, if a document is present in sets 0, 5, 9, 11, 12, and 13, it has the following binary representation:

```
11101000100001
```

where the digit at the furthest right position represents set 0, the digit to the left of set 0 represents set 1 and so on. Set numbers increase from right to left.

This number is the binary representation of the decimal number 14881, and the hexadecimal number `3A21`. Therefore, the `BitField` contains the value `3A21` to indicate that the document is part of these sets:

```
#DREFIELD BitField="003A21"
```

In this way, information on sets can be stored in a single field per document, for an arbitrarily large number of sets.

To set up BitFieldType fields

1. Open the IDOL Content component configuration file in a text editor.

2. List a Bit Field process in the `[FieldProcessing]` section. For example:

```
[FieldProcessing]
0=MyFirstProcess
1=BitFields
```

3. Create a section for each process that you listed, and in each section, create a property for the process (you define the property later by setting one or more applicable configuration parameters). Identify the fields that you want to associate with the process.

   > **NOTE:** The properties that you create must not have the same name as the processes.

   For example:

```
[MyFirstProcess]
Property=MyProperty
PropertyFieldCSVs=*/MyField,*/MyOtherField

[BitFields]
Property=BitFieldSetFields
PropertyFieldCSVs=*/WORKBOORK,*/BITFIELD
```

4. Create a section for the property in which you set the `BitFieldType` parameter to **True**. This enables Content to store the contents of the `PropertyFieldCSVs` fields as bit fields. For example:

```
[BitFieldSetFields]
BitFieldType=True
```

5. To compress the BitField index, set `BitFieldCompressed` to **True** in the property section. For example:

```
[BitFieldSetFields]
BitFieldType=True
BitFieldCompressed=True
```

6. Set the `BitFieldMaxMemoryKB` parameter to the maximum memory (in KB) that can be used by `BitFieldType` fields. If this is zero (the default) there is no limit to the memory.

```
[BitFieldSetFields]
BitFieldType=True
BitFieldMaxMemoryKB=True
```

7. If you want to define `BitFieldType` fields or add extra `BitFieldType` fields, but have already indexed content into Content, you can set `RegenerateBitFieldIndex` to **True** in the `[Server]` section. This allows Content to generate the files it requires to internally identify `BitFieldType` fields on startup, so that you need only to restart Content to able to use `BitFieldType` fields, rather than having to reindex all your data.

```
[Server]
(...)
RegenerateBitFieldIndex=True
```

You can also use the DREREGENERATE index action to regenerate the BitFieldType index while the server is running.

8. Save and close the configuration file.

9. Restart the IDOL Content component for your changes to take effect.

> **NOTE:** Each document that you store in Content must contain only one instance of any particular BitFieldType field.

Edit Set Information after Indexing

After a document is indexed into the IDOL Content component, you can edit the information in BitFieldType fields by using the DREREPLACE action with the #DREFIELDBITOR, #DREFIELDBITXOR, and #DREFIELDBITAND operators.

Related Topics

- Change Document Field Values, on page 430

Find Documents within a Set

You can query the IDOL Content component using FieldText with the BITSET field specifier to find only documents that are part of a particular set. In this case, you use a decimal value to identify the set number. For example:

```
http://localhost:9010/action=Query&FieldText=BITSET{4,18}:BitField
```

This query returns documents that are part of set 4 or set 18 for sets defined by the BitField field.

Related Topics

- BITSET, on page 256

Metadata Fields

Metadata fields are fields that the IDOL Content component creates for documents at index time to display information about the documents when they are returned as results for a query. Some document metadata fields are always displayed when Content returns a document as a query result. You can display all document metadata fields by adding XMLMeta=True to your query.

Content displays the following metadata fields for results:

- **<autn:baseid>**. If the document has multiple sections, this is the ID of the first section of the document. If the document is not sectioned, this value is the same as the document ID.

- **<autn:content>**. The text content of the document.

- **<autn:database>**. The Content database in which the document is stored.

- **<autn:date>**. The date (in epoch seconds) when the document was created. This date is read from the field that has been identified by the `DateType` parameter in the IDOL Content component configuration file. If no field has been identified, the date the document was indexed is used instead.

- **<autn:expiredate>**. The date (in epoch seconds) when the document expires. This date is read from the field that has been identified by the `ExpireDateType` parameter in the IDOL Content component configuration file. If you have set an offset in the `ExpireAfterDelay` parameter, the `<autn:expiredate>` field includes this offset to calculate the expiration date. When a document expires, it is deleted from Content or moved to a different database (depending on what you set `ExpireIntoDatabase` to in the IDOL Content component configuration file).

- **<autn:id>**. The document ID. This ID is assigned to the document at index time. If Content is compacted, the IDs of documents change.

- **<autn:language>, <autn:languageencoding>, <autn:languagetype>**. The language, encoding, and language type associated with the document. The language type is read from the field that you identified by the `LanguageType` property in the IDOL Content component configuration file. The language and encoding of the document are read from the `Language` and `Encoding` parameters set for this language type in the configuration file.

  If no field from which the language type can be read has been identified, the `DefaultLanguageType` that you set in the configuration file is used instead, unless automatic language detection is enabled, or the document has been submitted to Content with an index action that sets a specific language type for the document.

- **<autn:links>**. A list of stemmed terms that are contained both in the query and in the result document.

- **<autn:reference>**. The document reference. This is read from the field that has been identified by the `ReferenceType` parameter in the IDOL Content component configuration file. If no field has been identified, Content automatically generates a reference for the document at index time.

- **<autn:section>**. The number of sections the document has been split up into at index time. The first section is section 0.

- **<autn:title>**. The document title. This is read from the field that has been identified by the `TitleType` parameter in the IDOL Content component configuration file. If no field has been identified, the document is not given a title.

- **<autn:weight>**. The percentage relevance that the document has to the query.

When you set `XMLMeta` to **True**, Content also returns the following fields for your results set. These metadata fields are used by the DAH for sorting.

- **<autn:numericdatesort>**. The index (starting from zero) of the elements of your defined sort order that refer to a `NumericDateType` field.

- **<autn:numericsort>**. The index (starting from zero) of the elements of your defined sort order that refer to a `NumericType` field.

Change Field Values

You can use the DREREPLACE index action to change the values of fields or add fields to a document after you index content into the IDOL Content component. For more information, see Change Document Field Values, on page 430.

Chapter 5: Language Support

This section describes how the IDOL Content component supports processing in many different languages and how you configure that support.

IDOL Language Support Concepts

IDOL uses probabilistic modeling and therefore does not require any form of language-dependent parsing, dictionaries, or translation modules.

Treating words as abstract symbols of meaning allows IDOL technology to derive understanding through the context in which symbols occur rather than a rigid definition of grammar. Slang and other variations in language do not affect the software analysis.

The IDOL Content component can build up a statistical understanding of the patterns in any language. The more information Content has about a particular type of information (for example, legal terms, pharmaceutical developments, technology, and so on), the more understanding it gains of those topics.

You can think of a new language as simply another type of information, for which Content needs enough material to learn from. Therefore, it is possible to mix more than one language in Content as long as you have sufficient amounts of each language to build its understanding.

The choice of language does not compromise the accuracy of the concepts extracted by the IDOL Content component. The underlying algorithm is the same regardless of the language used.

IDOL internationalization functionality enables:

- **automatic language detection**. Content can automatically detect the language and encoding of documents that it processes. This feature allows you to set up processes that Content automatically applies to documents or document metadata if they are in a specific language. For example, if Content identifies a document as Chinese, it automatically applies the appropriate preliminary linguistic tools.

> **NOTE:** If a document contains multiple languages, Content determines which language it contains most, and processes the document according to the settings for this language.

- **cross-lingual systems**. You can set up cross-lingual systems in Content. This feature allows you to produce multilingual results for queries, or to restrict results to documents in a specific language or encoding. For example, an English query can return information both in English and Spanish.

Although IDOL technology is language independent, it can be beneficial to use language-dependent features to optimize the ability of the IDOL Content component to match concepts irrespective of their appearance in text. IDOL therefore provides the following features:

- **stop word lists**. Every language has words that do not carry much significant meaning. In grammatical terms these are normally prepositions, conjunctions, auxiliary verbs, and so on (for example, words such as *the*, *a*, and *to* in English). These words can be safely ignored when processing content.

  IDOL provides as standard a set of stop word lists for the most commonly used languages.

- **stemming**. In languages, some words have a common morphological root. IDOL provides stemming algorithms that reduce words to this form. This process allows you to match concepts regardless of the grammatical use of words. In English for example, the words *help*, *helpful*, *helping,* and *helped* can all be stripped to their stem *help* without significant loss of meaning.

  IDOL provides as standard a set of stemming algorithms for the most commonly used languages. IDOL applies stemming after it discards stop words, both at index time (when content is stored in the IDOL Content component) and at query time (IDOL removes stop words and stems query text before matching).

> **NOTE:** Content also supports per-language use of a stemming file, which you can use in conjunction with the stemming algorithms to specify stems for individual words.

- **multiple encodings**. Content supports multiple encodings for languages such as Greek and Russian. You can use different encodings interchangeably, which means that it does not matter which encoding a language is given in. For example, it is possible to query in one recognized encoding for a language and receive results that are in other encodings.

- **transliteration schemes**. Transliteration is the ability to represent letters that do not belong to the Latin alphabet or words that contain accented letters with the corresponding characters of another alphabet. This makes familiarity with the accents and special characters of different languages unnecessary.

- **canonicalization of characters**. Some encodings have more than one way to represent a character. For example, the Japanese katakana script can have full width or half width characters. Regardless of its width the character in itself carries the same meaning.

The IDOL software infrastructure uses canonicalization to ensure that it treats all character forms equally. It automatically converts to an internationally recognized canonical form.

Related Topics

- Create a Custom Stem File for a Language, on page 124

- Enable Transliteration, on page 126

Run the IDOL Content component in Multiple Languages

You can combine multiple languages in one IDOL Content component. Use the outline below to determine what you have to do.

To combine multiple languages in one IDOL Content component

1. Before you index your documents, ensure that the IDOL Content component configuration file contains the languages that you want to use. See Determine the Languages that are Enabled, on the next page.

2. If the configuration file does not contain all the languages that you want to use, add the missing languages. Set up a field process that enables Content to associate these languages with documents. See Define Language Types, on page 113 and Associate Language Types with Documents, on page 115.

3. Check the documents that you want to index into Content:

   - Content can read the language type (that is, language and encoding) from a document field. If some of your documents do not contain these fields, Content applies the default language type. See Add LanguageType Fields to Documents, on page 118 and Define a Default Language Type, on page 118.

     If you do not want to associate the default language type with your documents, enable automatic language detection. See Enable Automatic Language Detection, on page 120.

     Alternatively, you can manually index your documents into Content, adding the language type of the documents to each index action. In this case, you must index documents in batches, where each batch must have the same language type. See Index Data, on page 49.

   - Content automatically processes documents that contain fields that specify the language type. You must add any missing languages to the IDOL Content component configuration file.

Related Topics

- Languages and Language Files, on page 513

When you query Content, by default Content returns only documents that have the same language as the language type (that is, language and encoding) of the query. You can change this behavior in the following ways:

- To use a language type in the query text that is not the default language type, add the `LanguageType` parameter to your query.

- To return results in a specific encoding, add the `OutputEncoding` parameter to your query. You can return only encodings that are compatible with the query language.

- To return documents in multiple languages, add the `AnyLanguage` parameter to your query.

- To return documents in a specific language, add the `AnyLanguage` and `MatchLanguage` parameters to your query.

Related Topics

Determine the Languages that are Enabled

You can determine the languages that the IDOL Content component can process by looking at the IDOL Content component configuration file.

To check which languages are enabled in Content

1. Open the IDOL Content component configuration file in a text editor.

2. Find the `[LanguageTypes]` section.

   This section lists the languages that Content can process. For example:

   ```
   [LanguageTypes]
   DefaultLanguageType=englishUTF8
   DefaultEncoding=UTF8
   LanguageDirectory=C:\IDOLServer\IDOL\langfiles
   0=Afrikaans
   1=Albanian
   2=Arabic
   3=Armenian
   4=Azeri
   5=Basque
   6=general
   ```

   - The `DefaultLanguageType` parameter specifies the language type to apply when:

     ○ Content cannot read the language type nor encoding of a document from a specified field.

     ○ the action does not include a `LanguageType` parameter.

     ○ automatic language detection is not enabled.

- The `general` language category is for documents whose encoding is identified, but whose language is not.
- The `LanguageDirectory` parameter specifies the directory that contains resource files (such as stop word lists) that Content uses to process languages.

> **TIP:** You can also view information on licensed and configured languages and encodings on the **Languages** tab on the Status page in IDOL Admin. For more information, refer to the *IDOL Admin User Guide*.

Related Topics

Define Language Types

To run the IDOL Content component in multiple languages, specify the language types that you want Content to process. A language type is a combination of the language and encoding.

> **NOTE:** You must specify languages and language types before you index data into Content.

To specify language types

1. Open the IDOL Content component configuration file in a text editor.

2. Find the `[LanguageTypes]` section and list the languages that you want Content to process. You must use UTF-8 characters when specifying a language.

   For example:

   ```
   [LanguageTypes]
   0=English
   1=Afrikaans
   2=Albanian
   3=Arabic
   4=Armenian
   5=Azeri
   ```

3. In the `[LanguageTypes]` section, set any configuration parameters that you want to apply to all languages. For details of the configuration parameters you can use, refer to the *IDOL Server Reference*.

   > **NOTE:** As well as the general language configuration parameters, you can set any of the individual language configuration parameters in the `[LanguageTypes]` section. The value in this section sets the default value for all languages, which you can override in the individual language configuration sections.

   For example:

```
[LanguageTypes]
DefaultLanguageType=englishUTF8
DefaultEncoding=UTF8
LanguageDirectory=C:\IDOLserver\IDOL\langfiles
GenericTransliteration=True
StopWordIndex=1
ProperNames=3
TangibleCharacters=!?
```

4. For each language that you use, create a section using the name of the language.

   In this section, specify appropriate settings that determine how Content handles this language. For details on the configuration parameters you can use, refer to the *IDOL Server Reference*.

5. For each section, add the `Encodings` parameter and define the encodings and corresponding language types used by the language.

   For example:

```
[english]
Encodings=UTF8:englishUTF8
Stoplist=english.dat
IndexNumbers=1

[afrikaans]
Encodings=UTF8:afrikaansUTF8
IndexNumbers=1

[albanian]
Encodings=UTF8:albanianUTF8
IndexNumbers=1

[arabic]
Encodings=ARABIC_ISO:arabicARABIC_ISO,ARABIC:arabicARABIC,UTF8:arabicUTF8
IndexNumbers=1

[armenian]
Encodings=UTF8:armenianUTF8
IndexNumbers=1

[azeri]
Encodings=UTF8:azeriUTF8
IndexNumbers=1

[general]
Encodings=UTF8:generalUTF8,CYRILLIC:generalCYRILLIC
IndexNumbers=1
```

6. Save the configuration file.

7. You can now configure Content to associate the language types that you defined with documents.

Related Topics

- Supported Languages and Common Encodings, on page 513

- Display Online Help, on page 30

- Associate Language Types with Documents, below

Associate Language Types with Documents

After you define all the language types you want the IDOL Content component to process, set up a field process that enables Content to associate these language types with documents.

The way that you configure this field process depends on the documents that you want to index into Content:

- Documents that Contain a Language Type Field

- Documents that Contain Field Data that can Identify Language

Related Topics

- Define Language Types, on page 113

Documents that Contain a Language Type Field

Use the following procedure to set up languages if all the documents that you want to index into Content contain a field that specifies the language type.

To configure Content to read language type from a field

1. Set up a process for looking up the language of a document in the `[FieldProcessing]` section.

   For example:

   ```
   [FieldProcessing]
   0=LookForLanguage
   ```

2. Create a section for the process. Create a `Property` for the process and identify the field that you want the process to apply to.

   For example:

   ```
   [LookForLanguage]
   Property=SetLanguage
   PropertyFieldCSVs=*/DRELANGUAGE,*/myLanguageType
   ```

3. Create a section for this property. Set the `LanguageType` parameter to `True` to map the values of the `*/DRELANGUAGE` fields to the equivalent language type in the `[LanguageTypes]` section.

   For example:

```
[SetLanguage]
LanguageType=True

[LanguageTypes]
0=russian

[russian]
Encodings=CYRILLIC:russianCYRILLIC,CYRILLIC_ISO:russianCYRILLIC_
ISO,CYRILLIC_KOI8:russianCYRILLIC_KOI8,UTF8:russianUTF8
Stoplist=russian.dat
IndexNumbers=1
```

4. Save and close the configuration file.

5. Restart the IDOL Content component for your changes to take effect.

6. You can now index documents into Content.

Related Topics

- Index Data, on page 49

Documents that Contain Field Data that can Identify Language

Use the following procedure to set up languages if all the documents that you want to index into the IDOL Content component contain a field that contains data that you can use to identify the language type.

To configure the IDOL Content component to identify languages from field data

1. Use the `[FieldProcessing]` section of the IDOL Content component configuration file to define each language property that you want Content to be able to detect.

   For example:

   ```
   [FieldProcessing]
   0=DetectArabic
   1=DetectEnglish
   2=DetectChSimplified
   3=DetectChTraditional
   4=DetectFrench
   ```

2. Define a section with the name of the respective language type for each of the languages that you defined in the `[FieldProcessing]` section. In this section, specify the fields that Content must look for and the values that those fields must have to recognize the document as a particular language type.

   For example:

   ```
   [DetectArabic]
   Property=SetArabicProperty
   ```

```
PropertyFieldCSVs=*/DRELANGUAGETYPE,*/LANG
PropertyMatch=arabic

[DetectEnglish]
Property=SetEnglishProperty
PropertyFieldCSVs=*/DRELANGUAGETYPE,*/LANG
PropertyMatch=*eng*,uk,*british

[DetectChSimplified]
Property=SetChSimplifiedProperty
PropertyFieldCSVs=*/DRELANGUAGETYPE,*/LANG
PropertyMatch=*ChSimp*,ChineseSimp*

[DetectChTraditonal]
Property=SetChTraditionalProperty
PropertyFieldCSVs=*/DRELANGUAGETYPE,*/LANG
PropertyMatch=*ChTrad*,ChineseTrad*

[DetectFrench]
Property=SetFrenchProperty
PropertyFieldCSVs=*/DRELANGUAGETYPE, */DRELANGAGETYPE,*/LANG
PropertyMatch=*fre*,fran*
```

3. Define a section with the same value as the respective property for each `Property` that you defined in the `[FieldProcessing]` subsections. In this section, you can then specify the language type (which you must also list in the `[LanguageTypes]` section where you define how you want Content to handle the individual languages).

   For example:

   ```
   [SetArabicProperty]
   LanguageType=Arabic
   HiddenType=True

   [SetEnglishProperty]
   LanguageType=English
   HiddenType=True

   [SetChSimplifiedProperty]
   LanguageType=ChSimplified
   HiddenType=True

   [SetChTraditionalProperty]
   LanguageType=ChTraditional
   HiddenType=True

   [SetFrenchProperty]
   LanguageType=French
   HiddenType=True
   ```

4. Save and close the configuration file.

5. Restart the IDOL Content component for your changes to take effect.

6. You can now index documents into the IDOL Content component.

Related Topics

- Index Data, on page 49

Add LanguageType Fields to Documents

You can configure any of the IDOL connectors to add fields to documents from which the IDOL Content component can read the language type of the documents.

To add a language type field to documents in the connector

1. Open the configuration file of your IDOL connector in a text editor.

2. In the [TaskName] or [Ingestion] section, set the IngestActions parameter to META:*FieldName=FieldValue*.

   For example:

   ```
   IngestActions=META:DRELanguage=englishUTF8
   ```

   > **NOTE:** If you add this setting to a [*TaskName*] section, it applies only to the connector task defined in that section. If you add the setting to the [Ingestion] section, it applies to all connector tasks.

3. Save the configuration file.

Alternatively, you can use a Lua script to add a field in a Connector Framework Server (CFS) task. In a setup with a single CFS for multiple connectors, and documents in a single language, this method allows you to process all documents in one task, rather than setting up the ingest action for each connector.

Define a Default Language Type

You can specify a default language type (language and encoding) that the IDOL Content component uses for documents of unspecified language. It uses the default language type when:

- Content cannot read the language type of a document from a specified field.

- the action does not include a LanguageType parameter.

- automatic language detection is not enabled.

The default language type is also the default for the LanguageType parameter in actions such as Query and Summarize.

To specify a default language type

1. Open the IDOL Content component configuration file in a text editor.

2. Find the `[LanguageTypes]` section and enter the language type to associate with any document that does not contain a language type field.

   If you use automatic language detection, Content uses the detected language and encoding to determine the language type of documents, and not the default language type.

   For example:

   ```
   [LanguageTypes]
   DefaultLanguageType=englishUTF8
   LanguageDirectory=C:\HewlettPackardEnterprise\IDOLServer\common\langfiles
   ```

3. Save and close the configuration file.

4. Restart the IDOL Content component for your changes to take effect.

Define a General Language

You can specify a General language, which the IDOL Content component uses for documents with an unconfigured language, but whose encoding is identified. Content categorizes documents as the General language when:

- Content cannot read the language type nor the encoding of a document from a specified field.

- automatic language detection is enabled.

If Content detects an unconfigured language type, it indexes to the equivalent `General` language type for that encoding, if it exists. It also logs a warning message in the index log so that you can add an appropriate language type to the configuration file. Content also indexes unknown languages to the `General` language type for the encoding, if it exists. If the encoding is unknown, Content indexes the document to the default language.

If you set `DiscardUnconfiguredLanguagesAtIndex` to **True**, Content does not index documents with unconfigured languages, even if a `General` language exists for that encoding.

To specify a General language category

1. Open the IDOL Content component configuration file in a text editor.

2. Add the appropriate encodings to the `[General]` section.

3. Save and close the configuration file.

4. Restart the IDOL Content component for your changes to take effect.

If the document has no identified language or encoding, Content indexes it to the `DefaultLanguageType`.

Related Topics

-

Enable Automatic Language Detection

If your IDOL license includes automatic language detection, the IDOL Content component can automatically identify the language and encoding of a document when it is indexed. Content analyzes a certain amount of text in the document content fields (fields for which `SourceType` is set to **True** in the IDOL Content component configuration file).

To enable automatic language detection

1. Open the IDOL Content component configuration file in a text editor.

2. Find the `[Server]` section and add this setting:

   `AutoDetectLanguagesAtIndex=True`

3. Set `DiscardUnconfiguredLanguagesAtIndex` to **True** if you do not want to index documents with a language type that is not configured.

   Set `DiscardUnknownLanguagesAtIndex` to **True** if you do not want to index documents whose language Content cannot recognize. For example, it might not recognize the language because the document does not contain language, or it might not have enough text for Content to determine the language.

   By default, Content indexes the document using the default language type. It also logs a warning message in the index log, so that you can add an appropriate language type.

4. You can change the amount of text that Content analyzes to detect the language of a document. By default, Content uses only a few sentences. In some situations, increasing the amount of text to analyze can give more accurate results, such as when significant amounts of a minor second language are present.

   Add the `MaxLanguageDetectTerms` setting to the `[Server]` section, specifying the number of terms (words) that Content uses for detection. For example:

   `MaxLanguageDetectTerms=1000`

5. By default, Content detects any 7-bit ASCII characters as UTF-8. If you instead want to group these documents with documents using 8-bit ASCII, disable the `LangDetectUTF8` parameter by setting it to **False**.

   Ensure that the encoding option you want is present in the language type configuration (see Define Language Types, on page 113). If there are no compatible encodings configured for the detected language, IDOL assigns the default language type.

6. Save and close the configuration file.

7. Restart the IDOL Content component for your changes to take effect.

> **NOTE:** If you enable automatic language detection and set up a field process that reads the language of a document from one of its fields, Content uses the field process rather than autodetection to determine the document language and encoding.

Specify the Language Type of a Query

When you send a query to the IDOL Content component, by default it reads the query language type as the `DefaultLanguageType` defined in the IDOL Content component configuration file.

You must correctly set the language type of any query text so that Content can handle the query text correctly (for example, stemming correctly). If you want to send a query that does not use the default language type, you must add the `LanguageType` parameter to your query action. This parameter specifies that the query uses the language and encoding set in the IDOL Content component configuration file for the specified `LanguageType`.

For example:

The following query uses the language and encoding specified for the `DefaultLanguageType`, so you can send it to Content without adding the `LanguageType` parameter:

```
http://12.3.4.56:4000/action=Query&Text=The Bayes theory of probability
```

The following query uses the language and encoding specified for the `German` language type:

```
http://12.3.4.56:4000/action=Query&Text=Einsteins
Relativitätstheorie&LanguageType=German
```

Related Topics

Convert Results to a Specific Encoding

You can send the following types of query to the IDOL Content component:

- **Text queries**. Queries that contain some form of query text (for example, `Query`, `SuggestOnText`, `Summarize`, and so on).

- **Text-free queries**. Queries that do not contain any query text (for example, `Suggest`, `List`, `GetContent`, and so on).

Text Queries

When you send a query action to the IDOL Content component, by default it returns results that use the same language and encoding as the query text. This language type can be:

- the language and encoding specified in the `LanguageType` parameter sent with the query.

- the language type specified in the `DefaultLanguageType` configuration parameter if the query does not include a `LanguageType` parameter.

To return query results in a specific encoding, add the `OutputEncoding` parameter to your query action. This parameter converts the results of a query to any type of encoding that is compatible with the query

language. If you specify an encoding that is not compatible with the query language, Content returns an appropriate message.

You can specify a default value for the `OutputEncoding` parameter in the `DefaultEncoding` configuration parameter.

For example:

```
http://12.3.4.56:4000/action=Query&Text=Neurologia i
Neurochirurgia&LanguageType=PolishEASTERNEUROPEAN&OutputEncoding=EASTERNEUROPEAN_
ISO
```

In this example, Content converts all query results to `EASTERNEUROPEAN_ISO`.

Related Topics

- Supported Languages and Common Encodings, on page 513

Text-Free Queries

By default, query actions that do not contain any query text return results in the encoding specified in the `DefaultLanguageType` parameter. If any of the query results are not compatible with this encoding, the IDOL Content component indicates this in the results.

If you want a query action to return results in a specific encoding, you can add the `OutputEncoding` parameter to your query. Content converts all results to this encoding, as long as they are compatible with it. If any of the query results are not compatible with this encoding, Content returns an appropriate message. You can specify a default value for the `OutputEncoding` parameter in the `DefaultEncoding` configuration parameter.

For example:

```
http://12.3.4.56:4000/action=Suggest&ID=9016&OutputEncoding=EASTERNEUROPEAN_ISO
```

In this example, Content converts all query results to `EASTERNEUROPEAN_ISO`.

Return Documents in Multiple Languages

When you send a query action to the IDOL Content component, by default it returns results that use the same language and encoding as the query text. This language type can be:

- the language and encoding specified in the `LanguageType` parameter sent with the query.

- the language type specified in the `DefaultLanguageType` configuration parameter if the query does not include a `LanguageType` parameter.

To return documents in any language for your query rather than in the query language, set the `AnyLanguage` parameter to **True** to your query.

When Content receives the query, it applies the stemming algorithm and stop word list that is appropriate for the query language type. It returns only documents that contain words which match the stopped and stemmed terms in the query (that is, the words in result documents must stem to the same root as the words in the query text).

For example:

```
http://12.3.4.56:4000/action=Query&Text=Innovative internet marketing solutions in
Baghdad&AnyLanguage=True
```

In this example, Content returns documents in multiple languages that contain terms that match terms in the specified `Text`.

The query returns documents in multiple languages only if they contain terms that match terms in the query (for example, query text that contains the term *Baghdad* might return documents in English, French, German, and so on).

International Stop List

When you use the `AnyLanguage` parameter in all or most of your queries, you might want to use the international stop list. This stop list contains common stop words for all languages, but does not include terms that are stop words in one language, but useful search terms in another language.

For multilingual searching, this stop list ensures that the query includes all useful terms, while still removing all stop words that are never useful terms.

The `international.dat` stop list is included in the `langfiles` directory of a default IDOL Content component installation. You must configure each of the languages in your configuration file to use this stop list. If you have existing data in Content, you must also reindex the data.

To configure the international stop list

1. Open the IDOL Content component configuration file in a text editor.

2. In the `[Languages]` configuration section, find the configuration sections for all the languages that you want to return in `AnyLanguage` queries, including the language specified in the `DefaultLanguageType` parameter.

3. In each configuration section, set `Stoplist` to **international.dat**. For example:

   ```
   [english]
   Encodings=UTF8:generalUTF8
   Stoplist=international.dat

   [italian]
   Encodings=UTF8:italianUTF8
   Stoplist=international.dat

   [spanish]
   Encodings=UTF8:spanishUTF8
   Stoplist=international.dat
   ```

4. Save and close the configuration file.

5. Restart the IDOL Content component for your changes to take effect.

6. Reindex all your data into the IDOL Content component to apply the changes to all your documents.

Related Topics

Return Documents in a Specific Language

When you send a query action to the IDOL Content component, by default it returns results that use the same language and encoding as the query text. This language type can be:

- the language and encoding specified in the `LanguageType` parameter sent with the query.

- the language type specified in the `DefaultLanguageType` configuration parameter if the query does not include a `LanguageType` parameter.

To return documents in one more specific languages for the query, you can set the `MatchLanguage` parameter to a list of the languages that you want to match. When you set this parameter, Content applies the stemming algorithm and stop word list that is appropriate for the query language type as usual (it does not translate the query). The results include only documents in the specified languages that match the stopped and stemmed terms in the original query (that is, the words in result documents must stem to the same root as the words in the query text).

For example:

```
action=Query&Text=Birmingham&LanguageType=EnglishUTF8&MatchLanguage=Dutch+German
```

In this example Content stems the query term *Birmingham* according to the stemming rules for the `EnglishUTF8` language type. It returns any documents in Dutch or German that contain a term that has the same stem as the English stem of *Birmingham*.

> **NOTE:** If you specify `MatchLanguage`, you cannot specify `MatchLanguageType` or `MatchEncoding` in your query.

To return documents that match the query terms, regardless of the document language, you can add the `AnyLanguage` parameter to your query. See Return Documents in Multiple Languages, on page 122.

> **TIP:** You can use the **Languages** tab in the **Status** page of the IDOL Admin interface to search for all documents in a particular language. You can also modify that query in IDOL Admin. For more information, refer to the *IDOL Admin User Guide*.

Create a Custom Stem File for a Language

You can override the default stemming rules for certain words in a particular language by creating a language-specific *stemming file*. This file is a list of words and their stems. If a stemming file exists, the IDOL Content component uses it to stem the terms that it contains. Terms that are not in the file stem according to the default stemming rules.

Micro Focus recommends use of a stemming file only for unusual or specialized terms where the default rules do not generate a stem. A stemming file is not intended to be a complete replacement for the IDOL stemming algorithms.

To set up a stemming file

1. Create a text file.

2. Format the file as a stop word list. The first line is an encoding designation. Subsequent lines contain individual word pairs; a term followed by its stem. For example:

```
[UTF8]
mice mouse
mouse mouse
children child
```

The terms and stems can contain only alphanumeric characters.

> **NOTE:** To ensure that two words stem to the same value, you must add both words to the stemming file, with the appropriate stem.

3. Save the file with a name of your choice (for example, `english_stem.dat`) in the directory *installDir*`/common/langfiles`.

4. Open the IDOL Content component configuration file in a text editor.

5. In the [*MyLanguage*] section for the stemming file language, set the `StemmingFile` configuration parameter to the name of your stemming file. For example:

```
[english]
Encodings=UTF8:englishUTF8
Stoplist=engish.dat
StemmingFile=english_stem.dat
```

6. Ensure that this [*MyLanguage*] section does not set `Stemming` to **False**. The default value for `Stemming` in a language is **True**.

If you disable stemming for a language, but provide a stemming file, Content stems terms in the file, but does not stem other terms.

Decompose Compound Words

You can configure the IDOL Content component to automatically separate a compound word into root words at both index and query time. For example, the German word for *bicycle pump* is a single word *Fahrradpumpe* that can be divided into *Fahrrad* and *Pumpe*.

To specify decomposition

1. Create a text file. Use the following format:

```
[UTF8]
rollercoaster roller coaster
hemidemisemiquaver hemi demi semi quaver
```

Each line defines the decomposition for one word. The first word on a line is broken into the remaining words on the line.

2. Store the text file in the `langfiles` directory of your IDOL Content component installation.

3. Open the IDOL Content component configuration file in a text editor.

4. For each language that the decomposition file applies to, specify the file name in the `DecompositionFile` configuration parameter. For example:

```
[German]
DecompositionFile=german_decomp.txt
```

> **NOTE:** Each of the terms in the output from the decomposition is also stemmed.

Enable Transliteration

Transliteration is the process of converting accented characters such as øù into equivalent non accented characters. This process is useful in environments where accented keyboards are not available.

Enable Generic Transliteration

The default IDOL Content component configuration file uses generic transliteration. Micro Focus recommends that you use generic transliteration because it is the best way to ensure that cross-lingual search can happen.

To enable transliteration for all languages

- In the `[LanguageTypes]` configuration section, set the `GenericTransliteration` parameter to **True**.

Generic transliteration performs transliteration as described in the following table.

Language or character type	Transliteration
Symbols	All dashes and hyphens to a hyphen character.
Latin	Accented characters to non-accented characters
Spanish	Accented vowels áéíóúü to non-accented vowels
Portuguese	Accented vowels àáâãçéêíòóôõúü to non-accented vowels
Greek	Accented Greek characters to non-accented characters
Cyrillic (including Serbian extensions)	All characters mapped to A–Z
Arabic	Arabic character normalization
Japanese	Half width katakana to full width katakana Full width 0–9, A–Z, a–z to single byte 0–9, A–Z, a–z
Chinese	Full width 0–9, A–Z, a–z to single byte 0–9, A–Z, a–z

For all other languages, transliteration does not apply, except for hyphen normalization.

> **NOTE:** Languages with a sentence-breaking library might be transliterated as part of the sentence-breaking process.

When you set `GenericTransliteration` to **True**, it applies to all languages, unless you specifically disable transliteration for a language.

You can disable transliteration for an individual language by setting the `Transliteration` parameter to **False** in the individual language configuration section. This option completely disables transliteration for that language.

Enable Transliteration for Individual Languages

In cases where you need a particular transliteration, you can set `GenericTransliteration` to **False** and use the per-language transliteration schemes.

To turn transliteration on or off for an individual language

- In the `[MyLanguage]` individual language configuration sections, set the `Transliteration` parameter to **True** or **False**.

When you set `GenericTransliteration` to **False**, the IDOL Content component always performs transliteration for the languages in the following table, even if you also set `Transliteration` to **False**.

Language	Transliteration
Japanese	Half width katakana to full width katakana Full width 0–9, A–Z, a–z to single byte 0–9, A–Z, a–z
Chinese	Full width 0–9, A–Z, a–z to single byte 0–9, A–Z, a–z
Greek	Accented Greek characters to non-accented characters
Spanish	Accented vowels áéíóúü to non-accented vowels
Portuguese	Accented vowels àáâãçéêíòóôõúü to non-accented vowels
Russian	Some removal of characters
Arabic Persian Sindhi Urdu Malay Malayalam Pushto	Arabic character normalization

Transliteration is optional for the language groups in the following table.

Language group	Transliteration
Western European	àáâãä=a å=aa ç=c èéêë=e ìíîï=i òóôõö=o ø=oe ùúûü=u œ(oe)=oe æ=ae ß=ss ñ=nh ý=y ð=d þ=th
German	Same as Western European apart from: ä=ae ö=oe ü=ue
Scandinavian	Same as Western European apart from: ä=ae ö=oe ü=ue
Catalan	Same as Western European apart from: ç=sz
Cyrillic	All characters mapped to A–Z Transliteration scheme uses British Standard 2979:1958
South Slavic	For transliteration scheme, refer to *A Handbook of Bosnian, Serbian and Croatian* by Brown & Alt.

The following table describes the languages that each of these language groups contain.

Western European	Czech	Italian	Romanian
	Dutch	Maori	Slovakian
	English	Mirandese	Spanish
	French	Polish	Turkish
	Hungarian	Portuguese	
German	German		
Scandinavian	Danish	Norwegian	
	Finnish	Swedish	
	Icelandic		
Catalan	Catalan		
Cyrillic	Russian		
	Tajik		

South Slavic	Bosnian		
	Serbian		
	Croatian		

For all other languages, transliteration does not apply, except for hyphen normalization.

For full details of the transliteration of different characters in different transliteration modes, refer to *IDOL Expert*.

Related Topics

- Index Nonalphanumeric Characters, on page 61

Chapter 6: Set Up Document Tracking

Document tracking is a feature present in IDOL components that are involved in indexing. It reports upon the progress of documents as they pass through an index chain. Every time a document reaches a certain stage in the indexing process, the component commits event data to a back end, which stores the events.

The back end can be a log file, or an SQL database (this option requires an additional library, which is available in the IDOL Server installer). You retrieve the events by using an appropriate interface for your chosen back end, for example an SQL client for a database.

This section describes how to set up document tracking using either SQL or IDOL log files as a back end.

Set up Document Tracking with an SQL Back End

This section contains instructions for setting up document tracking in IDOL with an SQL back end. It includes instructions for setting up a PostgreSQL or Microsoft SQL Server database. PostgreSQL is the recommended and most fully tested database, but you can also use Microsoft SQL Server (2008 (SP3) or 2012), or Oracle.

- Set up Document Tracking with PostgreSQL
- Set up Document Tracking with Microsoft SQL Server

Set up Document Tracking with PostgreSQL

This section describes how to set up document tracking with PostgreSQL.This process includes the following steps:

- Install the PostgreSQL database.
- Create the database structure.
- Install SQL drivers on the IDOL host machines.
- Configure IDOL components.

Set up PostgreSQL to Store Tracking Information

This section describes how to set up the PostgreSQL back end.

The example actions and commands are for IDOL components running on a Microsoft Windows platform, and a PostgreSQL server running on a Linux platform. Other combinations of platforms are possible.

Install the SQL Database

The following Linux command example installs the latest stable PostgreSQL server, using the default port (5432). You can use any recent, stable version (on any port).

```
sudo apt-get install postgresql
```

You can test that your server is up with `psql`.

For Windows, you can install PostgreSQL from the following Web site:

http://www.postgresql.org/download/windows/

For more detailed installation instructions, refer to the PostgreSQL wiki.

https://wiki.postgresql.org/wiki/Main_Page

Set up the Database and Table

This section describes how to set up the database in PostgreSQL on Linux. On Windows, you can complete the tasks by using `Pgadmin`.

The PostgreSQL installation creates a user for you.

To set up the database and table on Linux

1. Create a database with an arbitrary name. For example:

   ```
   sudo -u postgres createdb mydoctrackdb
   ```

   You can test it by using the following example command:

   ```
   sudo -u postgres psql -d mydoctrackdb
   ```

2. Create the tables to store document tracking events, by running the following commands from `psql` or your SQL client:

   ```
   CREATE TABLE type(
     typeid serial PRIMARY KEY,
     type varchar(64) NOT NULL UNIQUE,
     is_error smallint,
     is_terminal smallint
   );

   -- Broken into several INSERTs here for clarity only
   INSERT INTO type(typeid, type, is_error, is_terminal)
           VALUES(0,'Unknown',0,0);
   ```

```
        INSERT INTO type(typeid, type, is_error, is_terminal)
                VALUES(10,'Committed',0,1),
                      (20,'Deleted',0,1),
                      (30,'Indexed',0,0),
                      (40,'Received',0,0);


        INSERT INTO type(typeid, type, is_error, is_terminal)
                VALUES(50,'Updated',0,1),
                      (-10,'Warning',1,0),
                      (-20,'Error',1,1),
                      (-30,'Rejected',1,1);


        INSERT INTO type(typeid, type, is_error, is_terminal)
                VALUES(90,'Added',0,0),
                      (100,'Delete received.',0,0),
                      (110,'Update received.',0,0);


        INSERT INTO type(typeid, type, is_error, is_terminal)
                VALUES(120,'Non-importing add received.',0,0),
                      (130,'Import:Queue',0,0),
                      (140,'Import:Importing',0,0);


        INSERT INTO type(typeid, type, is_error, is_terminal)
                VALUES(150,'Import:Pre',0,0),
                      (160,'Import:Post',0,0),
                      (170,'Import:Finished',0,0);


        INSERT INTO type(typeid, type, is_error, is_terminal)
                VALUES (180,'Import:Cancel',0,1);


        INSERT INTO type(typeid, type, is_error, is_terminal)
                VALUES (190,'Import:Extracting metadata',0,0);


        INSERT INTO type(typeid, type, is_error, is_terminal)
                VALUES (200,'Import:Extracting metadata finished',0,0);


        INSERT INTO type(typeid, type, is_error, is_terminal)
                VALUES (210,'Import:ExtractMetaAbort',1,1);


        INSERT INTO type(typeid, type, is_error, is_terminal)
                VALUES (220,'Import:Abort',1,1);


        INSERT INTO type(typeid, type, is_error, is_terminal)
                VALUES (230,'Replaced',0,0);

        CREATE TABLE source(
          sourceid serial PRIMARY KEY,
          source varchar(128) NOT NULL UNIQUE
        );
```

```
CREATE TABLE event(
  eventid serial PRIMARY KEY,
  docuid varchar(128) NOT NULL,
  typeid int NOT NULL,
  sourceid int NOT NULL,
  timestamp bigint NOT NULL,

  CONSTRAINT type_fk FOREIGN KEY(typeid)
     REFERENCES type(typeid)
     ON DELETE CASCADE ON UPDATE CASCADE,

  CONSTRAINT source_fk FOREIGN KEY(sourceid)
     REFERENCES source(sourceid)
     ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE metadata(
  metadataid serial PRIMARY KEY,
  key varchar(32) NOT NULL,
  value varchar(1024) NOT NULL
);

CREATE TABLE event_metadata(
  eventid int NOT NULL,
  metadataid int NOT NULL,

  CONSTRAINT event_fk FOREIGN KEY(eventid)
               REFERENCES event(eventid)
               ON DELETE CASCADE ON UPDATE CASCADE,

  CONSTRAINT metadata_fk FOREIGN KEY(metadataid)
     REFERENCES metadata(metadataid)
     ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE docuid_reference(
  docuid varchar(128) NOT NULL,
  ref varchar(4096) NOT NULL,

  UNIQUE (docuid, ref)
);

CREATE TABLE doctrack_schema_version(
  key varchar(64) NOT NULL,
  value varchar(128) NOT NULL
);

INSERT INTO doctrack_schema_version(key,value)
```

```
        VALUES('major_version', '1'),
               ('minor_version','1'),
               ('IDOL_version','10.9');

    CREATE RULE no_schema_version_insert AS
      ON INSERT TO doctrack_schema_version
      DO INSTEAD NOTHING;

    CREATE RULE no_schema_version_delete AS
      ON DELETE TO doctrack_schema_version
      DO INSTEAD NOTHING;
```

> **NOTE:** If you want to set up document tracking in an existing IDOL installation that uses the deprecated IndexTasks component, you must also add the following statements for the `type` table:
>
> ```
> INSERT INTO type(typeid, type, is_error, is_terminal)
>        VALUES(60,'IndexTasksCompleted',0,0),
>               (70,'IndexTasksStarted',0,0),
>               (80,'IndexTask',0,0);
> ```

To set up the database and table on Windows

1. Use `Pgadmin` to run the SQL **Create Table** command. Right-click **databases/mydoctrackdb** (or the name of the database that you created) and select **Create script**.

Database Access Permissions

On Windows, you can modify the configuration in `Pgadmin` by selecting the appropriate item on the left pane, and then clicking **Tools/Server Configuration**.

To modify the database access permissions

1. Find the `pg_hba.conf` host-based authentication file by inspecting the PostgreSQL configuration file. The following lines in the `postgresql.conf` file identify the location:

   ```
   hba_file = '/etc/postgresql/9.1/main/pg_hba.conf'  # host-based authentication
   file
   ```

   > **NOTE:** The location of the `postgresql.conf` file can vary, depending on your version and operating system. On Linux, you can run the following command to find the path to the configuration file:
   >
   > ```
   > ps -ef | grep postgres
   > ```

2. Modify the `pg_hba.conf` configuration file, which you located in Step 1, to allow your IDOL components to access the database. Find the following section, and add appropriate lines for your client IP addresses.

   ```
   # TYPE  DATABASE    USER          ADDRESS           METHOD
   host    all         my.user.name  10.2.123.123/32   trust
   ```

   For simplicity, you can set the USER field to **all**. Micro Focus recommends that you use a secure METHOD, such as **md5**, after you have tested the system.

3. Allow PostgreSQL to accept connections. In the `postgresql.conf` file, find the `listen_ addresses` parameter and uncomment or modify it:

   ```
   listen_addresses = '*'
   ```

4. Save and close the `pg_hba.conf` and `postgresql.conf` files.

5. Restart PostgreSQL, by using the following command (on Linux):

   ```
   sudo -u postgres /etc/init.d/postgresql restart
   ```

Set up the IDOL Host Machines

To communicate with the SQL server, you must install an SQL driver (for your server type), and you must install an ODBC driver manager on the IDOL host machines.

NOTE: To use document tracking with a PostgreSQL back end, you must have PostgreSQL ODBC client driver version 9.1.0 or later.

Install the SQL Driver and Manager for PostgreSQL

Microsoft Windows has a driver manager by default. If you require a driver manager for Linux, try `UnixODBC`, for example by using the following command:

```
sudo apt-get install unixodbc
```

You can download drivers for both Windows and Linux on the PostgreSQL Web site:

http://www.postgresql.org/ftp/odbc/versions/

Make sure that you install the correct version for your platform (for example, 64-bit).

On Linux, it might be easier to use your package manager. For PostgreSQL:

```
sudo apt-get install odbc-postgresql
```

NOTE: Make a note of the name of the SQL driver that you install, because you must reference it in a configuration parameter, or a data source name (DSN).

Check the Installed Drivers

Use the following procedures to check the installed drivers.

NOTE: You can use the Windows user interface to find the installed drivers for the Microsoft SQL Server back end.

To check the installed drivers on Linux and UNIX ODBC

- In your terminal (not in `psql`), type the following command to list the available drivers:

  ```
  odbcinst -d -q
  ```

To find a list of drivers on Windows

1. Run the following command to open the driver manager:

   `%windir%\system32\odbcad32.exe`

2. In the driver manager, review the information on the **Drivers** tab. The following driver is required:

   PostgreSQL ANSI or PostgreSQL ANSI (x64)

Configure IDOL Components

To use document tracking with an SQL back end, you must use the IDOL document tracking library, which is included in the IDOL Server installer. You can store this library in any accessible location, and then configure the location in your IDOL components.

You must then configure your IDOL components to use the SQL document tracking back end. The DIH, IDOL Content component, Connector Framework Server (CFS), and CFS Connectors support document tracking.

After configuration, an IDOL component automatically adds itself to the `Source` table on startup.

> **NOTE:** If you retire an IDOL component, Micro Focus recommends that you leave the entry in the `Source` table, because existing records might refer to the source of the retired component.

To configure an IDOL component for document tracking

1. Open the IDOL component configuration file in a text editor.

2. (DIH only) Turn on document tracking by setting the `DocumentTracking` parameter to **True** in the `[Server]` section of the DIH configuration file.

   ```
   [Server]
   DocumentTracking=True
   ```

   > **NOTE:** If you are using DIH in a unified IDOL Server configuration, set the `DocumentTracking` parameter to **True** in the `[DistributionSettings]` section.

3. Create a `[DocumentTracking]` configuration section.

4. In this `[DocumentTracking]` section, set the `Backend` parameter to **Library**.

5. Set `LibraryPath` to the absolute path to your document tracking library. In the default IDOL installation, the library is located in *InstallPath*/IDOL/modules/, and the library name is `dt_odbc.dll` (on Windows), or `libdt_odbc.so` (on UNIX).

6. Set `ConnectionString` to the connection string to use, with subparameters set for your setup. For example:

   ```
   ConnectionString=Driver=PostgreSQL ANSI(x64); Server=sql-host.mycompany.com;
   Port=5432; Database=mydoctrackdb; UID=postgres;Password=password;
   ```

   You can also use a DSN instead of subparameters. For example:

   ```
   ConnectionString=DSN=MyDSN
   ```

   Set the parameters consistent with your environment. On Linux, the connection string cannot contain spaces.

On Linux, you can set the `Driver` subparameter to the path to your SQL ODBC driver shared object (for example, `/usr/lib/x86_64-linux-gnu/odbc/psqlodbca.so` or `/usr/lib/odbc/psqlodbca.so`). Micro Focus recommends that you use the ANSI version of libraries.

> **TIP:** On Linux, if you use a DSN in your connection string, and you see `File not Found` errors in your IDOL logs, try specifying the driver explicitly in the `ConnectionString` configuration parameter:
>
> `ConnectionString=Driver=/usr/lib/psqlodbca.so;DSN=mydsn;`

7. Set any other configuration parameters for document tracking. For more information, refer to the *IDOL Server Reference*. For example:

```
[DocumentTracking]
MaxEventsPerFile=500
TimeoutSeconds=20
UIDFieldName=UID
```

> **NOTE:** Connectors generate the document ID strings, and add them to the `UIDFieldName` field, so you must not use a field that already exists for another purpose.

8. Save and close the configuration file.

    Restart the IDOL component for your changes to take effect.

Set up Document Tracking with Microsoft SQL Server

This section describes how to set up document tracking with Microsoft SQL Server 2008 (SP3). This process includes the following steps:

- Install Microsoft SQL Server.
- Create the database structure.
- Install SQL drivers on the IDOL host machines.
- Configure IDOL components.

Set up Microsoft SQL Server to Store Tracking Information

This section describes how to use Microsoft SQL Server as your SQL back end.

Configuration Example for Microsoft SQL Server

For the Microsoft SQL Server back end, you must set the `ConnectionString` configuration parameter in the `[DocumentTracking]` configuration section. This parameter must contain a valid connection string, as understood by SQL Server and your ODBC driver manager.

Micro Focus recommends that you use a DSN if you are running IDOL on a Windows operating system. This option lets the operating system save your settings. If you use Windows integrated authentication, use a connection string of the following form:

```
ConnectionString=DSN=dt;database=odbc_test_db
```

If you use SQL Authentication [user+password], add `TRUSTED_CONNECTION=yes;`. For example:

```
ConnectionString=DSN=dt;TRUSTED_CONNECTION=yes;database=odbc_test_db
```

> **NOTE:** You can set the database that you want to connect to in the DSN configuration GUI, or Microsoft SQL server. However, for SQL Server 2005, SQL Server 2008, and SQL Server 2012, Micro Focus recommends that you explicitly specify it in the connection string connection options.

You can also use a DSN on UNIX operating systems, but you might find configuration easier if you include all parameters in the `ConnectionString` parameter and omit the DSNs.

Troubleshoot Connection and Authentication Problems

The following table describes some common connection and authentication problems that you can identify in the SQL Management Studio, on the server side.

Issue	Suggestion
SQL Server is not configured to allow SQL Authentication.	In Server Properties, review the settings on the **Security** tab.
SQL Server is not configured to allow remote connections.	In Server Properties, review the settings on the **Connections** tab.
SQL Server is not mapping the user to the correct permissions for your database.	In the properties for the user, review the settings on the **User Mapping** tab.

For the client, check the following in the DSN configuration GUI:

- The driver must be `SQL Server Native Client 10.0`.

- The DSN must have **Use ANSI quoted identifiers**, **Use ANSI nulls, paddings, warnings**, and **Perform translation for character data** selected.

Initialization Commands

You can run the following commands from an SQL command-line interface, or from the GUI in Microsoft SQL Server Management Studio.

```
CREATE TABLE type(
    typeid integer identity PRIMARY KEY,
    type varchar(64) NOT NULL UNIQUE,
    is_error smallint,
    is_terminal smallint
);
SET IDENTITY_INSERT type ON
INSERT INTO type(typeid, type, is_error, is_terminal) VALUES(0,'Unknown',0,0);
INSERT INTO type(typeid, type, is_error, is_terminal) VALUES (10,'Committed',0,1);
INSERT INTO type(typeid, type, is_error, is_terminal) VALUES (20,'Deleted',0,1);
INSERT INTO type(typeid, type, is_error, is_terminal) VALUES (30,'Indexed',0,0);
INSERT INTO type(typeid, type, is_error, is_terminal) VALUES (40,'Received',0,0);
```

```
INSERT INTO type(typeid, type, is_error, is_terminal) VALUES (50,'Updated',0,1);
INSERT INTO type(typeid, type, is_error, is_terminal) VALUES (-10,'Warning',1,0)
INSERT INTO type(typeid, type, is_error, is_terminal) VALUES (-20,'Error',1,1);
INSERT INTO type(typeid, type, is_error, is_terminal) VALUES (-30,'Rejected',1,1);
INSERT INTO type(typeid, type, is_error, is_terminal) VALUES (90,'Added',0,0);
INSERT INTO type(typeid, type, is_error, is_terminal) VALUES (100,'Delete
received.',0,0);
INSERT INTO type(typeid, type, is_error, is_terminal) VALUES (110,'Update
received.',0,0);
INSERT INTO type(typeid, type, is_error, is_terminal) VALUES (120,'Non-importing
add received.',0,0);
INSERT INTO type(typeid, type, is_error, is_terminal) VALUES
(130,'Import:Queue',0,0);
INSERT INTO type(typeid, type, is_error, is_terminal) VALUES
(140,'Import:Importing',0,0);
INSERT INTO type(typeid, type, is_error, is_terminal) VALUES
(150,'Import:Pre',0,0);
INSERT INTO type(typeid, type, is_error, is_terminal) VALUES
(160,'Import:Post',0,0);
INSERT INTO type(typeid, type, is_error, is_terminal) VALUES
(170,'Import:Finished',0,0);
INSERT INTO type(typeid, type, is_error, is_terminal) VALUES
(180,'Import:Cancel',0,1);
INSERT INTO type(typeid, type, is_error, is_terminal) VALUES
(190,'Import:Extracting metadata',0,0);
INSERT INTO type(typeid, type, is_error, is_terminal) VALUES
(200,'Import:Extracting metadata finished',0,0);
INSERT INTO type(typeid, type, is_error, is_terminal) VALUES
(210,'Import:ExtractMetaAbort',1,1);
INSERT INTO type(typeid, type, is_error, is_terminal) VALUES
(220,'Import:Abort',1,1);
INSERT INTO type(typeid, type, is_error, is_terminal) VALUES (230,'Replaced',0,0);

CREATE TABLE source(
    sourceid integer identity PRIMARY KEY,
    source varchar(128) NOT NULL UNIQUE
);

CREATE TABLE event(
    eventid integer identity PRIMARY KEY,
    docuid varchar(128) NOT NULL,
    typeid int NOT NULL,
    sourceid int NOT NULL,
    timestamp bigint NOT NULL
);

CREATE TABLE metadata(
    metadataid integer identity PRIMARY KEY,
    [key] varchar(32) NOT NULL,
```

```
    value varchar(1024) NOT NULL
);

CREATE TABLE event_metadata(
  eventid int NOT NULL,
  metadataid int NOT NULL,
);

CREATE TABLE docuid_reference(
   docuid varchar(128) NOT NULL,
   ref varchar(900) NOT NULL,
);

CREATE TABLE doctrack_schema_version(
   [key] varchar(64) NOT NULL,
   value varchar(128) NOT NULL
);

INSERT INTO doctrack_schema_version([key],value) VALUES ('major_version', '1');
INSERT INTO doctrack_schema_version([key],value) VALUES ('minor_version','1');
INSERT INTO doctrack_schema_version([key],value) VALUES ('IDOL_version','10.9');
```

> **NOTE:** If you want to set up document tracking in an existing IDOL installation that uses the deprecated IndexTasks component, you must also add the following statements for the `type` table:
>
> ```
> INSERT INTO type(typeid, type, is_error, is_terminal) VALUES
> (60,'IndexTasksCompleted',0,0);
> INSERT INTO type(typeid, type, is_error, is_terminal) VALUES
> (70,'IndexTasksStarted',0,0);
> INSERT INTO type(typeid, type, is_error, is_terminal) VALUES
> (80,'IndexTask',0,0);
> ```

Set up the IDOL Host Machines

To communicate with the SQL server, you must install an SQL driver (for your server type), and you must install an ODBC driver manager on the IDOL host machines.

Install the SQL Driver and Manager for Microsoft SQL Server

Microsoft Windows comes with SQL Server drivers by default. You can use `odbcad32.exe` (in *WindowsDirectory*\system32) to verify that SQL Server Native Client 10.0 is installed.

For SQL Server 2008 SP3, you can download the driver from the Microsoft Web site:

http://www.microsoft.com/en-us/download/details.aspx?id=27596

> **NOTE:** You might need a different version of the ODBC drivers if you are using a different edition of SQL Server.

UNIX machines can use the open-source `FreeTDS` driver to connect to Microsoft SQL Server. A driver developed by Microsoft also exists, targeted at Red Hat Enterprise Linux. This example uses FreeTDS, which can be obtained from the following Web site:

http://www.freetds.org/

Download the driver distribution, and then read, edit, and run the following commands.

1. Unzip the package:

   ```
   tar –zxf freetds-stable.tgz
   ```

2. Use the following commands to build the driver:

   ```
   cd freetds-0.91

   ./configure --prefix=YourFileDirectory --with-tdsver=8.0

   make

   make install
   ```

These commands copy the drivers to your prefix directory. The driver itself is copied to

```
Prefix/lib/libtdsodbc.so
```

You can reference this driver in your connection string when configuring IDOL. For more information, see Configure IDOL Components, on the next page.

For example:

```
ConnectionString=Driver=/freetds/inst/lib/libtdsodbc.so;TDS_
Version=7.0;Server=myserverhostname\SQLEXPRESS;Port=56841;UID=idol;PWD=password;
```

Check the Installed Drivers

Use the following procedures to check the installed drivers.

> **NOTE:** You can use the Windows user interface to find the installed drivers for the Microsoft SQL Server back end.

To check the installed drivers on Linux and UNIX ODBC

- In your terminal (not in an SQL client), type the following command to list the available drivers:

  ```
  odbcinst -d -q
  ```

To find a list of drivers on Windows

1. Run the following command to open the driver manager:

   ```
   %windir%\system32\odbcad32.exe
   ```

2. In the driver manager, review the information on the **Drivers** tab. The following driver is required:

   SQL Server Native Client 10.0

Configure IDOL Components

To use document tracking with an SQL back end, you must use the IDOL document tracking library, which is included in the IDOL Server installer. You can store this library in any accessible location, and then configure the location in your IDOL components.

You must then configure your IDOL components to use the SQL document tracking back end. The DIH, IDOL Content component, Connector Framework Server (CFS), and CFS Connectors support document tracking.

After configuration, an IDOL component automatically adds itself to the `Source` table on startup.

> **NOTE:** If you retire an IDOL component, Micro Focus recommends that you leave the entry in the `Source` table, because existing records might refer to the source of the retired component.

To configure an IDOL component for document tracking

1. Open the IDOL component configuration file in a text editor.

2. (DIH only) Turn on document tracking by setting the `DocumentTracking` parameter to **True** in the `[Server]` section of the DIH configuration file.

   ```
   [Server]
   DocumentTracking=True
   ```

   > **NOTE:** If you are using DIH in a unified IDOL Server configuration, set the `DocumentTracking` parameter to **True** in the `[DistributionSettings]` section.

3. Create a `[DocumentTracking]` configuration section.

4. In this `[DocumentTracking]` section, set the `Backend` parameter to **Library**.

5. Set `LibraryPath` to the absolute path to your document tracking library. In the default IDOL installation, the library is located in *InstallPath*/IDOL/modules/, and the library name is `dt_odbc.dll` (on Windows), or `libdt_odbc.so` (on UNIX).

6. Set `ConnectionString` to the connection string to use, with subparameters set for your setup. For example:

   ```
   ConnectionString=DSN=MyDSN
   ```

   Set the parameters consistent with your environment. On Linux, the connection string cannot contain spaces.

   On Linux, you can set the `Driver` subparameter to the path to your SQL ODBC driver shared object. Micro Focus recommends that you use the ANSI version of libraries.

   > **TIP:** On Linux, if you use a DSN in your connection string, and you see `File not Found` errors in your IDOL logs, try specifying the driver explicitly in the `ConnectionString` configuration parameter:
   >
   > ```
   > ConnectionString=Driver=/freetds/inst/lib/libtdsodbc.so;DSN=mydsn
   > ```

7. Set any other configuration parameters for document tracking. For more information, refer to the *IDOL Server Reference*. For example:

    ```
    [DocumentTracking]
    MaxEventsPerFile=500
    TimeoutSeconds=20
    UIDFieldName=UID
    ```

    > **NOTE:** Connectors generate the document ID strings, and add them to the `UIDFieldName` field, so you must not use a field that already exists for another purpose.

8. Save and close the configuration file.

    Restart the IDOL component for your changes to take effect.

Verify the Setup

The following section describes how to troubleshoot and verify your setup.

Check IDOL Configuration

When you start the IDOL Content component, it logs the following message in the application log when the document tracking module loads correctly:

```
Starting document tracker
```

IDOL logs messages about document tracking to the `Index` log stream. To see these messages, set the `LogLevel` parameter to **Full** for this log stream.

Index Content

To test the document tracking setup, you can create an IDX document to add to the IDOL Content component. Make sure that you include the configured `UIDFieldName` in your IDX. Alternatively, you can use a connector to generate the IDX and index it.

The following example IDX has the UID field configured as the `UIDFieldName`.

```
dtsample.idx
#DREREFERENCE sample_idx_001
#DRETITLE A document to test document tracking
#DREDBNAME Default
#DREFIELD UID="001"
#DREFIELD Myfield1="field1"
#DREFIELD Myfield2="field2"
#DRECONTENT Did it work?
#DREENDDOC
#DREENDDATANOOP
```

Index this document into your IDOL Content component, for example by using a `DREADD` index action:

```
http://idol-server:9011/DREADD?dtsample.idx&IndexUID=test
```

Query Your Document

Run a `select` statement to query for the document in your SQL back end. For example:

```
test:~$ sudo -u postgres psql -d odbc_test
/usr/lib/postgresql/8.4/bin/psql: /usr/local/lib/libldap_r-2.4.so.2: no versi
on information available (required by /usr/lib/libpq.so.5)
psql (8.4.15)
Type "help" for help.

odbc_test=# select * from event limit 5;
 eventid | docuid | typeid | sourceid | timestamp
---------+--------+--------+----------+-----------
  114187 | 1678   |     10 |       25 | 1371487651
  114188 | 1679   |     10 |       25 | 1371487651
  114189 | 1680   |     10 |       25 | 1371487651
  114190 | 1681   |     10 |       25 | 1371487651
  114191 | 1682   |     10 |       25 | 1371487651
(5 rows)

odbc_test=#
```

You might need to wait some time (30-40 seconds) for Content to send records to your back end. After this time, if the document is not present, check the index log for SQL errors.

Clean Results

At the end of a test run, you might want to clean the database of tracking events. It is safe to delete entries from the `Event` table.

You must not clear the `Source` table while IDOL components are running. Stop the components before clearing the `Source` table or removing any entries from it.

> **NOTE:** If you retire an IDOL component, Micro Focus recommends that you leave the entry in the `Source` table, because existing records might refer to the source of the retired component.

Set up Document Tracking with a Log Back End

This section describes how to set up document tracking to store information in an IDOL log file.

To configure document tracking in logs

1. Open the IDOL component configuration file in a text editor.

2. (DIH only) Turn on document tracking by setting the `DocumentTracking` parameter to **True** in

the `[Server]` section of the DIH configuration file.

```
[Server]
DocumentTracking=True
```

> **NOTE:** If you are using DIH in a unified IDOL Server configuration, set the `DocumentTracking` parameter to **True** in the `[DistributionSettings]` section.

3. Create a `[DocumentTracking]` configuration section.

4. In this `[DocumentTracking]` section, set the `Backend` parameter to **Log**.

5. Set any other configuration parameters for document tracking. For more information, refer to the *IDOL Server Reference*. For example:

```
[DocumentTracking]
MaxEventsPerFile=500
TimeoutSeconds=20
UIDFieldName=UID
```

6. Find the `[Logging]` configuration section.

7. Configure a new log stream for document tracking, with `LogTypeCSVs` set to **doctrack**. For more information, see Customize Logging, on page 469.

    For example:

```
[Logging]
LogLevel=FULL
LogDirectory=Logs
0=ApplicationLogStream
1=ActionLogStream
2=DocumentTrackingLogStream
...

[DocumentTrackingLogStream]
LogFile=doctrack.log
LogHistorySize=50
LogTime=True
LogEcho=False
LogMaxSizeKBs=1024
LogTypeCSVs=doctrack
```

8. Save and close the configuration file.

    Restart the IDOL component for your changes to take effect.

Configure Event Storage

You can configure how document tracking stores the events by modifying the `[DocumentTracking]` configuration in your components. By default, document tracking stores all events, but you can use the

`PositiveEventFilter` and `NegativeEventFilter` configuration parameters to reduce the number of events to store.

- `PositiveEventFilter` allows you to configure a list of events that you explicitly want to store.

- `NegativeEventFilter` allows you to configure a list of events that you do not want to store.

If you configure both parameters, `PositiveEventFilter` takes precedence. For more information, refer to the *IDOL Server Reference*.

For a full list of event types that you can use in the filter parameters, see Document Tracking Event Definitions, below.

Document Tracking Event Definitions

The following table lists the event definitions that are used for document tracking.

Event type	Generated by	Description
Warning	General	An event that suggests some user attention might be required for this document.
Error	General	An event that suggests that further processing of the document might stop because of errors.
Committed	Content	A document is indexed and available to search.
Deleted	Content	A document has been deleted.
Indexed	Content	A document has been processed for indexing (but might not be searchable until flushed).
Updated	Content	A document has been updated by `DREREPLACE`.
Rejected	Content	This document has been rejected and will definitely not be indexed.
Received	DIH	A document has been received.
Added	Connectors	A document add command has been added to the connector ingest queue.
Deleted	Connectors	A document delete command has been added to the connector ingest queue.
Replaced	Connectors	A document replace command (delete and add) has been added to the connector ingest queue.
Updated	Connectors	A document update command has been added to the connector ingest queue.
Delete received.	CFS	CFS has received a document delete command.

Update received.	CFS	CFS has received a document update command.
Non-importing add received.	CFS	CFS has received a document that does not require import (for example, because it is metadata only).
Import:Queue	CFS	A document has been placed in the import queue.
Import:Importing	CFS	A document is being imported (that is, processed by KeyView).
Import:Pre	CFS	A document is being processed by the pre-import tasks.
Import:Post	CFS	A document is being processed by the post-import tasks.
Import:Finished	CFS	A document has been successfully imported.
Import:Cancel	CFS	A document import process has been cancelled.
Import:Extracting metadata	CFS	The import process is extracting metadata from a document.
Import:Extracting metadata finished	CFS	The import process has completed extracting metadata from a document.
Import:ExtractMetaAbort	CFS	The metadata extraction process was stopped for an import.
Import:Abort	CFS	The import process was stopped.

The following event definitions are generated by the deprecated IndexTasks component:

Event type	Generated by	Description
IndexTasksStarted	IndexTasks	The start of all tasks in IndexTasks.
IndexTask	IndexTasks	A task has been started in IndexTasks. The task name is available as a custom value for this event.
IndexTasksCompleted	IndexTasks	The end of all tasks in IndexTasks.

Part III: IDOL Server Operations

This section shows how you can make best use of the many information-retrieval, analysis, classification, and management capabilities of IDOL Server.

- Agents
- Categorization
- Document Classification
- Binary Categories
- AgentBoolean Agents and Categories
- Cluster Information
- Profiles

Chapter 7: Agents

This section describes how to set up and use agents.

About Agents

Agents automatically find documents for you that match your interests. A user who is interested in football and gardening could, for example, create a Real Madrid agent and a Pest Control agent.

When you create an agent, you give it training text. This training provides an example of the type of text that the agent must look for, so that an agent returns only documents, profiles, categories, or other agents that conceptually match its training.

For example, you could create a Mortgage agent and train it with text that is similar to the type of results you expect the agent to return. You can train the agent with text that you type yourself, or with documents. After you train the agent and specify details for it (such as the maximum number of results the agent returns, the minimum conceptual similarity of results and so on), you can run the agent. You can edit or retrain the agent at any time to fine-tune it.

> **NOTE:** By default agents match against all Content index databases. However, you can restrict the matching to one or more databases.

Related Topics

- AgentBoolean Agents and Categories, on page 185

Manipulate Agents

This section describes how to create, modify, view, and delete agents by using ACI actions.

> **TIP:** You can also create, modify, view, and delete agents by using the IDOL Admin interface. Refer to the *IDOL Admin User Guide* for more details.

Create an Agent

You can create agents by using the `AgentAdd` action to the IDOL Community component. For details on this action, refer to the *IDOL Server Reference*. For example:

```
http://12.3.4.56:4000/action=AgentAdd&UserName=Administrator&AgentName=Global+Warmi
ng&Training=Factors+affecting+global+warming&FieldMinScore=60
```

This action uses port 4000 to create an agent called `Global Warming` for the `Administrator` user, in the IDOL Community component, which is on a machine with the IP address `12.3.4.56`.

The IDOL Community component creates and stores the agent in the agent index (IDOL Agentstore component). The agent is trained to find documents whose concept matches the concept of the text `Factors affecting global warming`. Only documents that have a conceptual relevance of at least 60 percent to this text can return as results.

Related Topics

- Create AgentBoolean Agents and Categories, on page 188
- Optimize AgentBoolean Matching, on page 191

Edit an Agent

You can edit agents by using the `AgentEdit` action. For details on this action, refer to the *IDOL Server Reference*. For example:

```
http://12.3.4.56:4000/action=AgentEdit&UserName=Administrator&AgentName=Global+Warm
ing&FieldMinScore=75
```

Retrain an Agent

You can retrain agents by using the `AgentRetrain` action. For details on this action, refer to the *IDOL Server Reference*. Retraining the agent modifies the concepts of its training with the concepts of the text that you use for retraining. For example:

```
http://12.3.4.56:4000/action=AgentRetrain&UserName=Administrator&AgentName=Global+W
arming&PositiveDocs=534+352+4534
```

This action uses port 4000 to retrain the `Global Warming` agent for the `Administrator` user with the documents that have the IDs `534`, `352`, and `4534`.

Copy an Agent

You can copy an agent by using the `AgentCopy` action. For details on this action, refer to the *IDOL Server Reference*. You can copy an agent to use it as a template.You can copy the agent and then modify the copy. For example:

```
http://12.3.4.56:4000/action=AgentCopy&UserName=Administrator&AgentName=Global+Warm
ing&DestinationUserName=JSmith&DestinationAgentName=Environment
```

This action uses port 4000 to copy the `Global Warming` agent details from the `Administrator` user to the `Environment` agent for the user `JSmith`.

View Agent Details

You can view the details of an agent by using the `AgentRead` action. For details on this action, refer to the *IDOL Server Reference*. For example:

```
http://12.3.4.56:4000/action=AgentRead&UserName=Administrator&AgentName=Global+Warm
ing
```

This action requests the details of the `Global Warming` agent for the `Administrator` user from IDOL Community component.

Delete an Agent

You can delete an agent from the IDOL agent index by sending the `AgentDelete` action to the IDOL Community component. For details on this action, refer to the *IDOL Server Reference*. For example:

```
http://12.3.4.56:4000/action=AgentDelete&UserName=Administrator&AgentName=Global+Wa
rming
```

This action deletes the `Global Warming` agent for the `Administrator` user from the IDOL Community component.

Related Topics

- Display Online Help, on page 30

Query with Agents

You can query with an agent by using the `AgentGetResults` action. For details on this action, refer to the *IDOL Server Reference*.

> **NOTE:** When you match an agent against the IDOL Content component databases, all the agent terms are internally postfixed with a tilde (~) to indicate that the terms are stemmed and must not be stemmed again.

For example:

```
http://12.3.4.56:4000/action=AgentGetResults&UserName=Administrator&AgentName=Globa
l+Warming&DREDatabaseMatch=News,Archive
```

This action uses port 4000 to request the results of the `Global Warming` agent for the `Administrator` user from the IDOL Community component, which is located on a machine with the IP address `12.3.4.56`.

It matches the `Global Warming` agent against the IDOL Content component `News` and `Archive` databases.

Modify Document References for an Agent

You can specify a set of documents to mark as read or unread for an agent by using the `AgentChangeDocsReadStatus` action. For details on this action, refer to the *IDOL Server Reference*. For example:

```
http://12.3.4.56:9030/action=AgentChangeDocsReadStatus&UserName=Administrator&Agent
Name=Global+Warming&DocRefs=1234,5678,2953
```

This action modifies the agent called `Global Warming` for the `Administrator` user, and marks the documents with the IDs `1234`, `5678`, and `2953` as read, so that they do not return in the agent results.

Related Topics

- Display Online Help, on page 30

Collaboration and Expertise with Agents

You can use agents in IDOL to collaborate with other users or to locate experts in your field of interest.

Collaboration

The IDOL Community component automatically matches users with common explicit interest agents or similar implicit profiles. You can use this information to create virtual expert knowledge groups.

You can use the `Community` action to find agents or profiles in the community that match the agents or the profiles of a specific user. For example:

```
http://Communityhost:port
/action=Community&UserName=JSmith&Agents=True&Profiles=True&AgentsFindProfiles=True
&ProfilesFindAgents=True
```

This action instructs the IDOL Community component to find agents and profiles in the user community that match both the agents and the profiles of the user `JSmith`.

Expertise

The IDOL Community component accepts a natural language or Boolean search string and returns users who own matching agents or profiles. This action allows instant identification of experts in any subjects at hand, eliminating time-consuming searches for specialists, and unnecessary researching of subjects for which expert knowledge is already available

You can use the `Community` action to find agents or profiles in the community that match a natural language or Boolean search string. For example:

```
http://IDOLhost:port/action=Community&Text=how does the cost of funds, such as the
costs of performing a credit evaluation on the business requesting a loan,
```

```
determine the spread between the federal funds rate and the prime
rate&AgentsFindProfiles=True&ProfilesFindAgents=True
```

This action instructs the IDOL Community component to find agents and profiles in the user community that match the specified text.

Chapter 8: Categorization

The IDOL categorization capability allows you to create and administer categories, and to use them for categorization, suggesting, sentiment analysis, and matching.

Introduction to Categorization

The IDOL Category component automatically organizes text documents of any type into predefined categories. These sections describe how to adapt the categorization process to obtain the best possible performance.

For example, start with a data set of 1,000 news stories and a list of categories such as Sports, Politics, Entertainment, Science, and Business. The categorization process has two principal stages: training and testing. Divide the data set into training data, which might consist of 800 of the stories, and test data, which contains the remaining 200.

In the training stage, Category uses the training data to build the agents that it uses later to categorize the test data. A human expert sorts the data into the categories, by reading each news story and deciding which category it belongs to. More than one category might be appropriate for some stories. For example, you could place a story about patenting the human genome in both Science and Business. Other stories might not have an appropriate category, so they are discarded. After the training data has been sorted manually, you train the agents by running the training sections of the Categorizer.

After training, you are ready to categorize additional documents. You enter the test documents into Category, which automatically places them into the category or categories that its mathematical rules decide are most appropriate. Similar to the expert sorting the training data, Category might place a particular document in more than one category, or in no category at all.

The human expert must also sort the 200 test documents. You can then examine the performance of Category and determine how well it categorizes. If it is categorizing optimally, you can then add any future news stories into Category to categorize automatically. If required, you can add the original 200 test documents to the training set.

Related Topics

Create a Hierarchical Category Structure

The IDOL Category component provides a single category, the *root* category, which you cannot delete or modify. The root category serves as a base for the hierarchical category structure that you create. You can create categories under the root category:

- from scratch
- from clusters
- from legacy topic sets
- by copying categories
- by generating a taxonomy
- from XML

All categories are stored on disk, and become available for querying only if they are indexed into the IDOL category index (IDOL Agentstore component).

After you create categories, you can:

- train the categories
- retrain the categories
- move the categories

Create Categories from Scratch

You can create categories from scratch.

To create categories from scratch

1. Create the category by using the `CategoryCreate` action. For example:

   ```
   action=CategoryCreate&Name=Botanics&Category=1
   ```

   From this action, IDOL Server creates the `Botanics` category with an ID of `1`. The new category is a child of the `root` category, which has the ID `0`. Categories that you create from scratch are by default stored as child categories of the `root` category. However, you can specify an alternative parent category when you create a category. The `Category` parameter is optional. If you do not specify an ID, the system automatically generates a random ID.

2. Create a child category.

   The IDOL Category component returns an ID for the category that it creates. You can use this ID to identify the category, for example, to add a child category to it:

   ```
   action=CategoryCreate&Name=Perennials&Parent=1
   ```

In this example, Category creates the `Perennials` category. The new category is a child of the category with the ID `1` (in this case, the `Botanics` category).

3. Train the new category. You can set training action parameters for the `CategoryCreate` action to train a category when you create it.

4. Optionally, move categories to create a hierarchical structure or to modify their position in the category hierarchy.

Related Topics

Create Categories from Clusters

You can use the `CategoryImportFromCluster` action to create categories from clusters that the IDOL Category component has previously created. It imports these categories with training that is generated from the cluster concepts.

Category stores the categories that it imports from clusters in the `root` category, unless you specify a parent category for them. For example:

`action=CategoryImportFromCluster&SourceJobName=Job1&BuildNow=True`

In this example, Category imports all the clusters in the `Job1` cluster source job to categories in the `root` category. The `BuildNow` parameter instructs Category to build the categories immediately, so that they become active. You can also activate the category at a later point by using the `CategoryBuild` action.

Related Topics

Create Categories from Legacy Topic Sets

You can use the `CategoryImportFromTopic` action to import categories from existing legacy topic sets. The IDOL Category component creates one category for each topic set. When you import a topic set, you can specify whether you want to maintain the original Boolean rules of the topic, or import the topic as an IDOL concept matching agent.

All categories that you import from legacy topic sets are child categories of the `root` category. You can move them to create a hierarchical structure. For example:

`action=CategoryImportFromTopic&Topic=MyTopicFile.otl&BuildNow=True`

In this example, Category imports the topic sets that are stored in the `MyTopicFile.otl` to categories in the `root` category. The `BuildNow` parameter instructs Category to build the categories immediately, so that they become active. You can also activate the category at a later point by using the `CategoryBuild` action.

Related Topics

- Build Categories, on page 165

Create Categories by Copying Categories

You can create a category by copying an existing category and retraining or editing it. The IDOL Category component stores the new category in the same position in the `root` category as the original category, unless you specify a parent category for it. For example:

```
action=CategoryCopy&Category=123456789012345&Name=BotanicsCopy&Parent=9876543210987
6&BuildNow=True
```

In this example, Category copies the category with the ID `123456789012345`. It calls the new category `BotanicsCopy` and stores it as a child category of the category with the ID `98765432109876`. The `BuildNow` parameter instructs Category to build the categories immediately, so that they become active. You can also activate the category at a later point by using the `CategoryBuild` action.

Related Topics

- Build Categories, on page 165

Create Categories when you Generate a Taxonomy

You can use the `TaxonomyGenerate` action to generate a taxonomy to build categories from clusters or query results. The IDOL Category component stores the imported categories in the `root` category, in a hierarchical structure that reflects the hierarchical structure of the taxonomy.

Related Topics

- Generate a Taxonomy from Clusters, on page 169
- Generate a Taxonomy from Query Results, on page 169

Create Categories from XML

The `CategoryImportFromXML` action allows you to create categories by importing category information from an XML file. This file can be a third-party category XML hierarchy provided that it follows the IDOL category XML format. The categories are imported with the training set in the XML file.

Related Topics

- Category XML Format, on page 569

Create Categories from Partitions

The `CategoryPartition` action divides up a specified set of documents into several partitions. The IDOL Category component assigns each document to a partition, and generates a title for each partition. This option is similar to clustering, but it places all documents in a query set into partitions. For example:

```
action=CategoryPartition&DREQuery=child+benefit+cuts&NumResults=500&NumPartitions=6
```

In this example, Category retrieves all the documents that match the specified query terms up to a maximum of 500 results, splits them into six partitions, and returns the results as an ACI response.

To create categories from the partitions, add the `CreateCategories` action parameter to the `CategoryPartition` action.

For more information, refer to the *IDOL Server Reference*.

Related Topics

- Cluster Information, on page 197

Create Categories for Sentiment Analysis

The `CategorySetupSentimentAnalysisCats` action allows you to set up and train positive, negative, and neutral sentiment categories to use to perform sentiment analysis.

After you run the `CategorySetupSentimentAnalysisCats` action, you can perform sentiment analysis without having to perform any other actions.

For example:

```
action=CategorySetupSentimentAnalysisCats&PositiveCatName=PositiveFeedback&Positive
CatTraining=the new procedure for routing service calls has really improved our
customer satisfaction&Parent=71056
```

In this example, the IDOL Category component sets up a category, `PositiveFeedback`, as a child of the category with the ID 71056, and trains that category with the specified text.

Related Topics

- Suggest Categories with Confidence Values, on page 168

Train Categories

You can use the `CategorySetTraining` action to train a category. Category training can consist of text, documents, a Boolean expression, category content, or a combination of these. These elements identify text, documents, agents, profiles, and other categories that match the category.

For example:

```
action=CategorySetTraining&Category=323499876022105571056&DocID=238,785,9912&BuildN
ow=True
```

In this example, the IDOL Category component trains the category with the ID `323499876022105571056` using the content of the documents with the IDs `238`, `785`, and `9912`. The `BuildNow` parameter instructs Category to build the categories immediately, so that they become active. You can also activate the category at a later point by using the `CategoryBuild` action.

You can also train categories when you create them by assigning training action parameters to the `CategoryCreate` action.

> **NOTE:** Categories that you create or import by using an action other than `CategoryCreate` are already trained. However, you can retrain them.

Related Topics

Retrain Categories

Use the `CategorySetTraining` action to retrain a category. You can use text, documents, a Boolean expression, and category content or a combination of these to retrain a category. When you retrain a category, its original training merges with the new training. For example:

```
action=CategorySetTraining&Category=323499876022105571056&Boolean=dog AND NOT
cat&BuildNow=True
```

In this example, the IDOL Category component retrains the category with the ID `323499876022105571056` using the Boolean expression `dog AND NOT cat`. The `BuildNow` parameter instructs Category to build the categories immediately, so that they become active. You can also activate the category at a later point by using the `CategoryBuild` action.

Related Topics

Move Categories

You can use the `CategoryMove` action to move individual categories in the category hierarchy. For example:

```
action=CategoryMove&Category=124365780934532&Parent=12398234987345876
```

In this example, the IDOL Category component moves the category that has the ID `124365780934532` to the category with the ID `123098234987345876` (to make category `12398234987345876` the new parent of category `124365780934532`).

View and Administer Categories

The IDOL Category component allows you to do these tasks to maintain your category hierarchy:

- view category details
- view category hierarchy details
- view category terms and weights
- view category training
- change category fields
- change category term weights
- replace categories
- activate categories

- build categories

- delete categories

- delete category training

- export categories to XML

- sync the IDOL category index with the categories stored on disk

View Category Details

Use the `CategoryGetDetails` action to view category fields. For example:

`action=CategoryGetDetails&Category=124365780934532`

In this example, the IDOL Category component returns all fields in the category with the ID `124365780934532`.

View Category Hierarchy Details

Use the `CategoryGetHierDetails` action to view hierarchy details for a category. For example:

`action=CategoryGetHierDetails&Category=124365780934532`

In this example, IDOL Server returns the hierarchy details for the category with the ID `124365780934532`.

View Category Terms and Weights

Use the `CategoryGetTNW` action to view category stemmed terms and their weights. For example:

`action=CategoryGetTNW&Category=124365780934532`

In this example, the IDOL Category component returns the terms and weights of the category with the ID `124365780934532`.

View Category Training

Use the `CategoryGetTraining` action to view category training. For example:

`action=CategoryGetTraining&Category=124365780934532`

In this example, the IDOL Category component returns the training of the category with the ID `124365780934532`.

Change Category Fields

Use the `CategorySetDetails` action to set the value of one or more category fields, or to create new fields in a category.

By default each category has a threshold of 0 and is set to return six results. Use the CategorySetDetails action fields THRESHOLD and NUMRESULTS to set the threshold and the number of results that a category can return. For example:

```
action=CategorySetDetails&Category=124365780934532&Fields=THRESHOLD,NUMRESULTS&Valu
es=60,10&BuildNow=True
```

In this example, the IDOL Category component sets the THRESHOLD field of the category with the ID 124365780934532 to 60 and its NUMRESULTS to 10. The BuildNow parameter instructs Category to build the categories immediately, so that they become active. You can also activate the category at a later point by using the CategoryBuild action.

Related Topics

- Build Categories, on the next page

Reset Category Fields

Use the CategoryResetDetails action to reset the value of the category fields to their default values.

Use the CategoryResetDetails action to reset the values of any of these fields: DATABASES, FIELDTEXT, NUMRESULTS, THRESHOLD, TAXONOMYROOT, SIMPLECATDEFAULTCAT, SIMPLECATPARAM, and FIELD. The action also removes values associated with the fields. For example:

```
action=CategoryResetDetails&Category=324987602&Params=NUMRESULTS,THRESHOLD
```

In this example, the IDOL Category component resets the NUMRESULTS and THRESHOLD fields of the category with ID 324987602 to their default values of 6 and 0.

Change Category Term Weights

You can use the CategorySetTNW action to change the weights of terms in the category that you believe are weighted inappropriately. For example:

```
action=CategorySetTNW&Category=124365780934532&Terms=tax,monei,budget&Weights=2353,
1223,1023&BuildNow=True
```

In this example, the IDOL Category component sets the weight of the term tax to 2353, the weight of the term monei to 1223 and the weight of the term budget to 1023 (tax, monei and budget are what IDOL stems the words *Tax*, *Money* and *Budget* to).

The BuildNow parameter instructs Category to build the categories immediately, so that they become active. You can also activate the category at a later point by using the CategoryBuild action.

Related Topics

- Build Categories, on the next page

Remove Category Term Weights

You can use the CategorySetTNW action to remove any modifications that you made to the weights of terms in a category. For example:

```
action=CategorySetTNW&Category=124365780934532&BuildNow=True
```

In this example, IDOL Server removes all previous changes to the weights of all terms in the category with the ID `124365780934532`. The `BuildNow` parameter instructs Category to build the categories immediately, so that they become active. You can also activate the category at a later point by using the `CategoryBuild` action.

Related Topics

- Build Categories, below

Replace Categories

You can use the `CategoryReplace` action to replace a category with another category. For example:

```
action=CategoryReplace&FromCategory=123456789012345&ToCategory=98765432109876&Build
Now=True
```

In this example, the IDOL Category component replaces the `98765432109876` category with the `123456789012345` category. The `BuildNow` parameter instructs Category to build the categories immediately, so that they become active. You can also activate the category at a later point by using the `CategoryBuild` action.

Related Topics

- Build Categories, below

Activate or Deactivate Categories

You can use the `CategoryActivate` action to activate or deactivate a category. You cannot query inactive categories or return them as results. By default the IDOL Category component activates a category when it builds it. For example:

```
action=CategoryActivate&Category=3234998760221&Active=True
```

In this example, the IDOL Category component activates the category with the ID `3234998760221`.

Use the `CategoryGetHierDetails` action to find out whether categories are active.

Related Topics

- View Category Hierarchy Details, on page 163

Build Categories

You can use the `CategoryBuild` action to build a category. You must build a category after you create it and train it, as well as every time that you retrain it. Building a category identifies the concepts of the training for the category and indexes the category into the IDOL category index (IDOL Agentstore component). For example:

```
action=CategoryBuild&Category=32349987602210557106
```

In this example, the IDOL Category component builds the category with the ID `32349987602210557106`.

> **NOTE:** If you train or retrain a category by using the `CategorySetTraining` action with `BuildNow` set to **True**, you do not have to run a `CategoryBuild` action, because the category was built immediately after it was trained.
>
> The IDOL Category component also builds categories immediately if you assign training parameters to the `CategoryCreate` action to train a category when you create it.

Delete Categories

You can use the `CategoryDelete` action to delete a category. Deleting a category removes the category from disk and from the IDOL category index. For example:

`action=CategoryDelete&Category=32349987602210557106`

In this example, the IDOL Category component deletes the category with the ID 32349987602210557106.

Delete Category Training

You can use the `CategoryDeleteTraining` action to delete all or part of the training for a category. Deleting a category removes the training from disk and from the IDOL category index. For example:

`action=CategoryDeleteTraining&Category=32349987602210557106`

In this example, the IDOL Category component deletes the training for the category with the ID 32349987602210557106.

Export Categories to XML

You can use the `CategoryExportToXML` action to export a category to XML format, including its descendants, training documents, and terms and weights. For example:

`action=CategoryExportToXML`

In this example, the IDOL Category component exports the entire category structure to XML.

Synchronize the Category Index with Stored Categories

You can use the `CategorySyncCatDRE` action to synchronize the IDOL category index with the categories stored on disk. `CategorySyncCatDRE` deletes the current contents of the category index, and overwrites it with the category information stored on disk. For example:

`action=CategorySyncCatDRE`

In this example, the IDOL Category component synchronizes its category index with the categories stored on disk.

Categorize Data

You can configure IDOL to automatically categorize data before you index it.

To automatically categorize documents before storing them in the IDOL Content component index, set up a preprocessing task in Connector Framework Server (CFS). The IDOL Category component matches incoming documents against categories that its category index contains and returns matching categories. CFS then tags the incoming documents according to the categories that they match.

For more information, refer to the *Connector Framework Server Administration Guide*.

Related Topics

-

Suggest Categories

The IDOL Category component can suggest conceptually similar categories for:

- documents

- text

- categories

Suggest Categories for Documents

You can use the `CategorySuggestFromDocument` action to suggest categories from the IDOL category index that are conceptually similar to a specified document. For example:

```
action=CategorySuggestFromDocument&DocID=125
```

In this example, the IDOL Category component returns categories that are conceptually similar to the document with the ID `125`.

Suggest Categories for Text

You can use the `CategorySuggestFromText` action to suggest categories from the IDOL category index that are conceptually similar to specified text. For example:

```
action=CategorySuggestFromText&QueryText=Caring for passiflora incarnata
```

In this example, the IDOL Category component returns categories that are conceptually similar to the text `Caring for passiflora incarnata`.

It is possible to configure a spell checking engine to detect spelling mistakes in the input for this action and use the corrected terms to suggest categories. `TextParse` input is also spellchecked and

categories are suggested based on the corrected terms, without changing the spelling in the document being ingested.

For configuration details refer to *IDOL Server Reference.*

Suggest Categories for Categories

You can use the `CategorySuggestFromCategory` action to suggest categories from the IDOL category index that are conceptually similar to a specified category. For example:

```
action=CategorySuggestFromCategory&Category=32349987602210557106
```

In this example, the IDOL Category component returns categories that are conceptually similar to the category with the ID `3234998760221055`.

Suggest Categories with Confidence Values

You can use the `CategorySimpleCategorize` action to suggest categories from the IDOL category index that are conceptually similar to a specified document, piece of text, or file content, and to return terms and weights for each category.

You can also specify cluster job names and cluster numbers to import into categories when you use the `CategorySimpleCategorize` action. This enables you to perform sentiment analysis on clusters.

Related Topics

- Create Categories for Sentiment Analysis, on page 161

Match Categories

You can use the `CategoryQuery` action to match categories against data, agents, profiles, and other categories. For example:

```
action=CategoryQuery&Category=32349987602210557106
```

In this example, the IDOL Category component matches the category with the ID `32349987602210557106` against all its databases and returns conceptually similar data, agents, profiles, and categories.

Create Taxonomies

The IDOL Category component taxonomy generation feature allows you to automatically create hierarchical contextual taxonomies of clusters or other information. It provides you with an overview of the information landscape and an insight into specific areas of the information.

You can also create a taxonomy manually and name it by the category that is its root node.

Generate Taxonomies Automatically

The `TaxonomyGenerate` action allows you to generate a hierarchical taxonomy from one or more clusters or query results.

The taxonomy generator adapts the Bayesian and information theoretic methods to concept selection. It applies Bayesian algorithms to identify statistical relationships between concepts and sets of concepts (at the document and document set level). It then filters them to form the hierarchical structure of the final taxonomy.

You can write the taxonomy to disk as a directory structure, or import the taxonomy into the category hierarchy.

> **NOTE:** Before you create a taxonomy, ensure that your IDOL Content component index does not contain duplicate documents or text that is repeated in multiple documents (for example, document headers). Ensure that these are stripped out at the import stage to gain optimal results.

You can set up a schedule that runs the `TaxonomyGenerate` action at regular intervals.
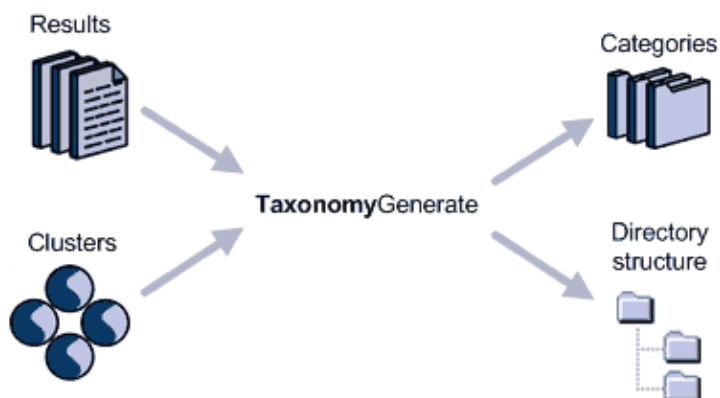
Related Topics

- Cluster Information, on page 197

Generate a Taxonomy from Clusters

Use the `TaxonomyGenerate` action with the `SourceJobName` and `Cluster` parameters to generate a taxonomy from one or more clusters. For example:

```
action=TaxonomyGenerate&SourceJobName=Taxonomy1&Cluster=0,1
```

In this example, the IDOL Category component generates a taxonomy from the **Taxonomy1** cluster.

Generate a Taxonomy from Query Results

Use the `TaxonomyGenerate` action with the `DREQuery` parameter to generate a taxonomy from a query. For example:

```
action=TaxonomyGenerate&DREQuery=new+tax+cuts
```

In this example, the IDOL Category component generates a taxonomy from the results that it returns from its data index for the query `new tax cuts`.

Schedule Taxonomy Generation

You can set up a schedule to run the `TaxonomyGenerate` action at regular intervals.

Related Topics

- Set up Schedules, on page 205

Create Named Taxonomies

You can take advantage of the hierarchical nature of IDOL categorization to manually create and store named taxonomies in the IDOL Category component. You can then use those taxonomies to investigate searches or suggestions.

If you set up a hierarchy of categories using the `CategoryCreate` or `CategoryImportFromXML` actions, you can then run `CategorySetDetails` for the topmost category with `TaxonomyRoot` set to **True**. That category becomes the root of the new taxonomy and its name is the new taxonomy name.

You can also generate a named taxonomy automatically by running the `TaxonomyGenerate` action and setting `TaxonomyRoot` to **True**.

You can subsequently use the taxonomy name as a parameter in actions such as `CategoryFind` and `CategorySuggestFromDocument`, to restrict the scope of the action results.

Categorization Example

This section contains an example of how you might set up categorization in the IDOL Category component. Each step provides several examples of possible actions for that step.

To use Categorization

1. Create a few categories.

   ```
   action=CategoryCreate&Name=Animals&Category=10
   ```

   ```
   action=CategoryCreate&Name=Cats&Parent=10&Category=11
   ```

   ```
   action=CategoryCreate&Name=Dogs&Parent=10&Category=12
   ```

2. Train the new categories (from a selection of indexed documents, free text, and Boolean rules).

   ```
   action=CategorySetTraining&Category=10&DocRef=http://foo.com/animals&DocID=1
   09,178
   ```

   ```
   action=CategorySetTraining&Category=11&Training=Cats and kittens can be a
   variety of colours, such as tabby or tortoiseshell
   ```

   ```
   action=CategorySetTraining&Category=12&Boolean=dog OR hound
   ```

3. Build the categories (this processes the training into terms and weights, enabling `CategoryQuery` and `CategorySuggest`).

    ```
    action=CategoryBuild&Category=10&Recurse=True
    ```

4. Manually adjust the terms and weights for one of the categories, and rebuild it.

    ```
    action=CategorySetTnW&Category=11&Terms=cat,kitten&Weights=500,400
    ```

    ```
    action=CategoryBuild&Category=11
    ```

5. Query for documents matching a particular category, with optional parameters.

    ```
    action=CategoryQuery&Category=11
    ```

    ```
    action=CategoryQuery&Category=12&DatabaseMatch=News&NumResults=10
    ```

6. Categorize already-indexed documents or free text.

    ```
    action=categorySuggestfromDocument&DocID=1465
    ```

    ```
    action=CategorySuggestFromText&QueryText=the cat in the hat
    ```

Chapter 9: Document Classification

IDOL document classification allows you to create classifiers, which use the random forest algorithm to classify documents into a set of classes that you define.

Introduction to Document Classification

Document classification allows you to automatically assign documents to classes according to values that occur in a set of fields that you specify. Classification uses the random forest algorithm, and you can use it as an alternative to the IDOL conceptual categorization (see Categorization, on page 157).

Classification works by analyzing the contents of various feature fields in the documents. You choose feature fields that contain useful information for classifying the documents. Typically, feature fields contain small snippets of information, rather than large portions of text. For example, you might use a name, or color field, rather than the document content. The feature fields that you use depend on the classifier and classes that you want to create.

To use classification, you create one or more *classifiers*. A classifier contains a set of *classes* (similar to categories), which represent the topics that you want to assign documents to. You train each class with a set of documents that represent the kind of content that you want the class to match.

After you create and train a classifier, you can query the classifier with new documents, and the classifier returns the details of the class that each new document belongs to.

Use Document Classification

The following sections describe how to create and train a classifier, and query the classifier with new documents.

For more information about the classification actions, refer to the *IDOL Server Reference*.

Choose Feature Fields

Before you create a classifier, you must choose the fields in your documents that you want to use to classify documents. These are the feature fields for the classifier.

Feature fields generally contain short pieces of information, such as a name or a very brief description. A good choice of feature field is similar to a good choice of `ParametricType` field. For example, if you

want to create a food classifier, you might use a field that stores ingredients, or a meal name, rather than a field that contains a recipe procedure or a detailed description of a type of food.

The feature fields must contain information that describes features of the different classes that you want to create for your classifier. For example, to classify meals as vegetarian or meat-based, you must find feature fields that describe features of vegetarian or meat-based meals.

The exact choice of feature field also depends on the contents of your documents.

For example, the following IDX document describes part of a recipe for soup:

```
#DREREFERENCE Food/Carrot and Coriander Soup
#DRETITLE Carrot and Coriander Soup
#DRESECTION 0
#DREFIELD Ingredient="carrots"
#DREFIELD Ingredient="onion"
#DREFIELD Ingredient="potato"
#DREFIELD Herbs="coriander"
#DREFIELD Seasoning="vegetable stock"
#DREFIELD Meal="soup"
#DREFIELD Equipment="food processor"
#DREFIELD PreparationTime="20 minutes"
#DREFIELD CookingTime="1 hour"
#DREFIELD Description"This easy recipe makes a tasty carrot and coriander soup"
#DRECONTENT
Example soup recipe
#DREENDDOC
```

- For a classifier that distinguishes between vegetarian meals and meat-based meals, you might choose the `Ingredient` field.

- For a classifier that distinguishes between savory recipes and desserts, you might choose the `Meal` and `Ingredient` fields.

- For a classifier that distinguishes between quick recipes and meals that take longer, you might choose the `PreparationTime` and `CookingTime` fields.

You can choose more than one feature field for a classifier. The classifier does not distinguish between data from different feature fields. It extracts the content from all the available feature fields from a document, and uses all the content to train the classifier (or classify a document).

For example, if your document had the fields:

```
#DREFIELD Ingredient1="carrots"
#DREFIELD Ingredient2="onion"
#DREFIELD Ingredient3="potato"
```

You can set `Ingredient1`, `Ingredient2`, and `Ingredient3` as feature fields. If you use this document for classification, it gives the same results as if you used a document with the following fields:

```
#DREFIELD Ingredient1="onion"
#DREFIELD Ingredient2="potato"
#DREFIELD Ingredient3="carrots"
```

Create a Classifier

You create a classifier with a unique name and a set of feature fields.

To create a classifier

- Send a `ClassifierCreate` action to the IDOL Category component, with the following parameters:

  - `ClassifierName` set to the name of the new classifier. This name must be unique in the IDOL Category component.

  - `ClassifierType` set to **RandomForest**.

  - `FeatureFields` set to a comma-separated list of the feature fields that you want to use for the classifier.

For example:

```
action=ClassifierCreate&ClassifierName=food&FeatureFields=Ingredient,Herbs,Seasoning
```

This action creates a `food` classifier, which uses the `Ingredient`, `Herbs`, and `Seasoning` fields to classify documents.

Create and Train Classes

After you create the classifier, you create and assign training to the classes. You can either create the classes and assign training in a single action, or you can create the classes and train them later.

The documents that you use to train the class must exist in the IDOL data index (IDOL Content component). You provide training in the form of a state token, which you create by using the `Query` action with the `StoreState` parameter set to **True**. See Choose Training Documents for Classes, on the next page.

To create a class

- Send a `ClassifierAddClass` action to the IDOL Category component, with the following parameters:

  - `ClassifierName` set to the name of the classifier.

  - `ClassName` set to the name of the new class.

  - (Optional) `StateID` set to a state token that lists the documents that you want to use to train the class.

For example:

```
action=ClassifierAddClass&ClassifierName=food&ClassName=vegetarian&StateID=B8UGIK95
FKJG-23
```

This action creates a `vegetarian` class in the `food` classifier. It assigns the documents from the state token `B8UGIK95FKJG-23` as training for the new class.

If you do not train the class when you create it, you can add training by using the `ClassifierSetClassTraining` action. You can also use this action to retrain a class. For more information, see .

You must run the `ClassifierAddClass` action for each class that you want to create in the classifier.

Choose Training Documents for Classes

When you create a classifier, you must train each of the classes with content that represents the classes that you want to define. The content must exist in your IDOL data index, and the content must contain the feature fields that you have defined for the classifier.

You provide training to the classes as a state token. You create state tokens by sending the `Query` action with the `StoreState` parameter set to **True**. Therefore, to train a class, you must have a single query that returns the documents that define that class.

For some classifications, you might be able to perform a complex query that returns enough documents to train your classifier. However, the best way to find training is usually to manually categorize a set of documents, and add a field that labels the document with its class. You can then use a simple `FieldText` query to find all documents with a particular label.

For example, if you label a set of documents with a `MealType` field, with a value of *savory* or *dessert*, you can use the following query to find and save the results to use as training for the `savory` class:

```
action=Query&FieldText=MATCH{savory}:MealType&MaxResults=1000&StoreState=True
```

You can use the resulting state token that this query returns to train the class. You can also create similar queries to train your other classes.

After you have trained the classifier, you can classify any new documents, and automatically add the label field to those documents.

> **NOTE:** To get the best results out of your classifiers, use as many training documents as possible. Micro Focus recommends that you use a minimum of 200 to 300 training documents for each class.

Train the Classifier

You must train the classifier before you can use it to classify documents. During this stage, the IDOL Category component retrieves all the training documents from the index, and extracts the feature fields. It uses the content to train each class in the classifier.

For Category to successfully train the classifier, it must have at least two classes, each of which must have training assigned.

> **NOTE:** When Category trains the classifier, it ignores any very rare features.

To train a classifier

- Send a `ClassifierTrain` action to the IDOL Category component, with the `ClassifierName` parameter set to the name of the classifier.

For example:

`action=ClassifierTrain&ClassifierName=food`

This action trains the `food` classifier.

> **NOTE:** The action returns an error if IDOL Category component could not extract any features from the training documents (for example, because none of the training documents contained the feature fields for the classifier).

Classify Documents

You can use a trained classifier to classify documents, by using the `ClassifierQuery` action.

The document can either be:

- the document reference for a document that exists in the IDOL Content component index.
- a percent-encoded IDX or XML document.

In both cases, the IDOL Category component extracts the classifier feature fields from the query document, and compares the values in these feature fields against the trained classes in the classifier. The action returns the class that the document matches most closely.

To classify a document that exists in the index

- Send the `ClassifierQuery` action with the following parameters.
  - `ClassifierName` set to the name of the classifier to use to classify the document.
  - `DocRef` set to the IDOL reference of the document to classify.

For example:

`action=ClassifierQuery&ClassifierName=food&DocRef=http://www.example.com/documents/carrots`

To classify a document that does not exist in the index

- Send the `ClassifierQuery` action to the IDOL Category component with the following parameters.
  - `ClassifierName` set to the name of the classifier to use to classify the document.
  - `QueryText` set to the percent-encoded IDX or XML document (Category detects the correct format automatically).

For example:

```
action=ClassifierQuery&ClassifierName=food&QueryText=%23DREREFERENCE%20Food%2FLeek%
20and%20Potato%20Pie%0D%0A%23DRETITLE%20Leek%20and%20Potato%20Pie%0D%0A%23DRESECTIO
N%200%0D%0A%23DREFIELD%20Ingredient%3D%22leeks%22%0D%0A%23DREFIELD%20Ingredient%3D%
22potatoes%22%0D%0A%23DREFIELD%20Ingredient%3D%22cheese%22%0D%0A%23DREFIELD%20Ingre
dient%3D%22pastry%22%0D%0A%23DREFIELD%20Ingredient%3D%22butter%22%0D%0A%23DREFIELD%
20Ingredient%3D%22egg%22%0D%0A%23DREFIELD%20Herbs%3D%22rosemary%22%0D%0A%23DREFIELD
%20Herbs%3D%22thyme%22%0D%0A%23DREFIELD%20Meal%3D%22pie%22%0D%0A%23DREFIELD%20Equip
ment%3D%22pie%20dish%22%0D%0A%23DREFIELD%20PreparationTime%3D%2210%20minutes%22%0D%
0A%23DREFIELD%20CookingTime%3D%221%20hour%22%0D%0A%23DREFIELD%20Description%22This%
20easy%20recipe%20makes%20a%20tasty%20leek%20and%20potato%20pie%22%0D%0A%23DRECONTE
NT%0D%0APie%20recipe%0D%0A%23DREENDDOC
```

This action classifies the following document:

```
#DREREFERENCE Food/Leek and Potato Pie
#DRETITLE Leek and Potato Pie
#DRESECTION 0
#DREFIELD Ingredient="leek"
#DREFIELD Ingredient="potato"
#DREFIELD Ingredient="cheese"
#DREFIELD Ingredient="shortcrust pastry"
#DREFIELD Ingredient="butter"
#DREFIELD Ingredient="egg"
#DREFIELD Herbs="rosemary"
#DREFIELD Herbs="thyme"
#DREFIELD Meal="pie"
#DREFIELD Equipment="pie dish"
#DREFIELD PreparationTime="10 minutes"
#DREFIELD CookingTime="1 hour"
#DREFIELD Description"This easy recipe makes a tasty leek and potato pie"
#DRECONTENT
Pie recipe
#DREENDDOC
```

View and Administer Classifiers

After you have set up classification, you can list and view classifiers, retrain classes, and delete classes and classifiers.

List and View Classifiers

The `ClassifierList` and `ClassifierGetInfo` actions allow you to view information about the classifiers you have created. `ClassifierList` returns information for all classifiers, and `ClassifierGetInfo` returns information for a single classifier that you specify. Both actions return the number of classes in the classifier, the feature fields that the classifier uses, and whether the classifier has been trained.

For example:

```
action=ClassifierList
```

This action returns the names of all your classifiers, and the information for each classifier.

```
action=ClassifierGetInfo&ClassifierName=food
```

This action returns information for the `food` classifier.

Retrain a Class

You can change the training documents associated with a class in a classifier by using the `ClassifierSetClassTraining` action. This action overwrites any existing training for the class with the new training. After you retrain a class, you must retrain the classifier.

To retrain a classifier

1. Send the `ClassifierSetClassTraining` action with the following parameters:

   - `ClassifierName` set to the name of the classifier.

   - `ClassName` set to the name of the class to retrain.

   - `StateID` set to a state token that lists the documents that you want to use to train the class.

   For example:

   ```
   action=ClassifierSetClassTraining&ClassifierName=food&ClassName=vegetarian&Stat
   eID=G7KPID13APWM-15
   ```

   This action updates the training for the `vegetarian` class in the `food` classifier to use the documents listed in the state token `G7KPID13APWM-15`.

2. Send a `ClassifierTrain` action to IDOL Server, with the `ClassifierName` parameter set to the name of the classifier.

   For example:

   ```
   action=ClassifierTrain&ClassifierName=food
   ```

   This action trains the `food` classifier, and updates the training for the retrained classes.

Delete a Class

You can delete a class from a classifier by using the `ClassifierDeleteClass` action. After you send this action, IDOL Server automatically retrains the classifier.

For example:

```
action=ClassifierDeleteClass&ClassifierName=food&ClassName=vegetarian
```

This action deletes the `vegetarian` class from the `food` classifier and retrains the classifier.

Delete a Classifier

You can delete a classifier that you no longer need by using the `ClassifierDelete` action. This action deletes the classifier and all associated classes.

For example:

`action=ClassifierDelete&ClassifierName=food`

This action deletes the `food` classifier.

Chapter 10: Binary Categories

This section describes how to set up and use binary categories.

About Binary Categories

A binary category is a special kind of category, designed to answer yes/no questions about documents, files, and text, like *Is this document spam?*, *Does this violate company policy?,* or *Is this work-related?*.

After you create a binary category, you train it. Unlike regular categories, which receive only positive training, binary categories can receive both positive and negative training. You provide the binary category with documents or text that result in a *yes* (POSITIVE) answer when querying with the binary category. You also provide documents or text that result in a *no* (NEGATIVE) answer to the same query.

After training, the binary category determines whether a piece of text, a file on disk, or a document in IDOL results in a *POSITIVE* or a *NEGATIVE* result. IDOL also provides a score (0–1) for the result, which indicates the confidence.

Create and Administer Binary Categories

The following section describes how to create, train, delete, change, and view binary categories.

Related Topics

- Display Online Help, on page 30

Create a Binary Category

You can use the `BinaryCatCreate` action to create binary categories. For details on this action, refer to the *IDOL Server Reference*. For example:

```
http://12.3.4.56:9020/action=BinaryCatCreate&Name=spam_binarycat
```

This action uses port `9020` to instruct the IDOL Category component, which is located on a machine with the IP address `12.3.4.56`, to create a new binary category named `spam_binarycat`.

Train a Binary Category

You can use the `BinaryCatTrain` action to train a binary category. Unlike normal categories, which have only positive training, binary categories can have both positive and negative training.

If the binary category has existing training, `BinaryCatTrain` adds the new training to it. If you want to replace the training for a binary category, you must first use the `BinaryCatDeleteTraining` action to delete the existing training. For example:

```
http://12.3.4.56:9020/action=BinaryCatTrain&Name=spam_
binarycat&PositiveDocID=123,456&NegativeDocID=789,890
```

This action uses port `9020` to instruct the IDOL Category component, which is located on a machine with the IP address `12.3.4.56`, to train the binary category named `spam_binarycat`. It uses the documents with IDs `123` and `456` for positive training, and the documents with IDs `789` and `890` for negative training.

Delete Training From a Binary Category

You can use the `BinaryCatDeleteTraining` action to remove the existing training from a binary category. Using the `BinaryCatTrain` action on a binary category with existing training adds the new training to the existing training, unless you use the `BinaryCatDeleteTraining` action first. For details on this action, refer to the *IDOL Server Reference*. For example:

```
http://12.3.4.56:9020/action=BinaryCatDeleteTraining&Name=spam_binarycat
```

This action uses port `9020` to instruct the IDOL Category component, which is located on a machine with the IP address `12.3.4.56`, to delete the training of the binary category named `spam_binarycat`.

Change Binary Category Details

You can use the `BinaryCatSetDetails` action to change the fields for a binary category from their default values. For details on this action, refer to the *IDOL Server Reference*. For example:

```
http://12.3.4.56:9020/action=BinaryCatSetDetails&Name=spam_
binarycat&MinDocOccs=12&TestTermsPerDoc=20
```

This action instructs the IDOL Category component to change the value of the parameter `MinDocOccs` to `12`, and the value of the parameter `TestTermsPerDoc` to `20`, for the binary category named `spam_binarycat`.

View Binary Category Details

You can use the `BinaryCatGetDetails` action to view the details of a binary category. For details on this action, refer to the *IDOL Server Reference*. For example:

```
http://12.3.4.56:9020/action=BinaryCatGetDetails&Name=spam_binarycat
```

This action uses port `9020` to instruct the IDOL Category component, which is located on a machine with the IP address `12.3.4.56`, to display parameter values of the binary category named `spam_binarycat`.

List Binary Categories

You can use the `BinaryCatList` action to view a list of all the binary categories in the system. For details on this action, refer to the *IDOL Server Reference*. For example:

`http://12.3.4.56:9020/action=BinaryCatList`

This action uses port `9020` to instruct the IDOL Category component, which is located on a machine with the IP address `12.3.4.56`, to list all the binary categories in the system.

Delete a Binary Category

You can use the `BinaryCatDelete` action to delete a binary category from the IDOL Category component. For details on this action, refer to the *IDOL Server Reference*. For example:

`http://12.3.4.56:9020/action=BinaryCatDelete&Name=spam_binarycat`

This action uses port `9020` to instruct the IDOL Category component, which is located on a machine with the IP address `12.3.4.56`, to delete the binary category named `spam_binarycat`.

Query with Binary Categories

After you train the binary category, you can use the `BinaryCatQuery` action to determine whether a piece of text, a file on disk, or a document in IDOL results in a "yes" (POSITIVE) or a "no" (NEGATIVE) answer for the question that the binary category asks. For example, your binary category might ask the question "Is this spam?" and could filter emails to ignore spam emails. For details on the `BinaryCatQuery` action, refer to the *IDOL Server Reference*.

For example:

`http://12.3.4.56:9020/action=BinaryCatQuery&Name=spam_binarycat&QueryText=How to become an instant millionaire`

This action uses port `9020` to instruct the IDOL Category component, which is located on a machine with the IP address `12.3.4.56`, to check whether the text `How to become an instant millionaire` results in a "POSITIVE" or a "NEGATIVE" result for the question posed by the binary category `spam_binarycat`.

Related Topics

- Display Online Help, on page 30

Binary Category Example

This section contains a step-by-step scenario for how you might use a binary category. After the initial creation of the binary category, each step provides more than one example of possible actions.

1. Create a binary category.

   ```
   action=BinaryCatCreate&Name=spam_binarycat
   ```

2. Train the new binary category (from a selection of indexed documents, free text, and files).

   ```
   action=BinaryCatTrain&Name=spam_
   binarycat&PositiveDocID=123,456&NegativeDocID=789,890
   ```

   ```
   action=BinaryCatCreate&Name=spam_binarycat&PositiveTraining=Get rich quick,
   join today&NegativeTraining=We should discuss the Jones file
   ```

   ```
   action=BinaryCatCreate&Name=spam_
   binarycat&PositiveDirectory=C:\spampositive&NegativeDirectory=C:\spamnegative
   ```

3. Query some data using the binary category.

   ```
   action=BinaryCatQuery&Name=spam_binarycat&QueryFile=C:\unknown_email.txt
   ```

   ```
   action=BinaryCatQuery&Name=spam_binarycat&QueryText=Limited space available,
   apply today
   ```

The following sample shows a `POSITIVE` result, with a score of `0.99664` from a binary category query:

```
<autnresponse xmlns:autn="'http://schemas.autonomy.com/aci/'">
    <action>
        BINARYCATQUERY
    </action>
    <response>
        SUCCESS
    </response>
    <responsedata>
        <autn:queryresult>
            <autn:result>
                POSITIVE
            </autn:result>
            <autn:score>
                0.99664
            </autn:score>
        <autn:queryresult>
    </responsedata>
</autnresponse>
```

Chapter 11: AgentBoolean Agents and Categories

This section describes how to set up and use AgentBoolean agents and categories in IDOL.

AgentBoolean Agents and Categories

You can create agents and categories that use keywords, conceptual information, a Boolean or proximity expression, or a FieldText expression to match documents. The fields in the agents or categories that contain these expressions are called *AgentBoolean* fields.

The following sections describe how to set up and use AgentBoolean agents and categories.

To use AgentBoolean and FieldText fields

1. Configure the fields that the IDOL Content component uses to match against AgentBoolean agents and categories.

2. Create agents and categories that contain Boolean or FieldText matching expressions.

3. Send queries to the IDOL Content component to match against the categories and agents.

After you set up AgentBoolean matching, you can optimize the system to provide the most efficient matching.

Related Topics

- Configure IDOL Server for Text Parse Queries, on page 187

- Create AgentBoolean Agents and Categories, on page 188

- Perform AgentBoolean Queries, on page 189

- Optimize AgentBoolean Matching, on page 191

Examples

IDOL stores agents and categories in the IDOL Agentstore component in the same way as the IDOL Content component stores documents. You can create agents and categories by using IDOL actions (in the IDOL Community component or IDOL Category component), or you can index an IDX or XML agent or category into the Agentstore. For example:

```
#DREREFERENCE 947344A0
#DRETITLE
My Cat and Dog Agent
#DREFIELD MyABField="cat AND dog"
#DREFIELD FieldTextField="MATCH{cat}:Animal"
#DRECONTENT
cat
#DREENDDOC
```

Similarly, you can search for agents and categories in the IDOL Agentstore component in the same way that you search for documents in the IDOL Content component.

For example, you can find agents and categories that match a document. This process allows you to categorize documents, or alert users when a new document matches their agent.

In this case, AgentBoolean expressions can improve the performance and accuracy for matching documents. It also provides extra functionality that you cannot easily achieve with simple conceptual agents.

Match Specific Concepts

If you have an Apollo category, it matches documents that contain the concepts *Apollo space program* and *Greek god Apollo*. You can use a Boolean expression to specifically restrict results to one or other of the concepts. For example:

```
"Space Program" NOT "Greek god"
```

Use Field Restrictions

You can use field restrictions in your AgentBoolean expressions, to match only the most relevant documents. For example:

```
"New Zealand":COUNTRY AND wine.
```

This expression matches documents that contain the phrase `New Zealand` in the `COUNTRY` field, and contain the term `wine`.

Related Topics

- Simple Field Restricted Search, on page 240

Use Term Occurrence Restrictions

You can use term occurrence restrictions in your agents to ensure that only the most relevant documents return. For example:

```
"Gene Therapy"[10:]
```

This expression matches documents in which the phrase `Gene Therapy` occurs ten or more times.

Categorize Documents before Indexing

You can use the Connector Framework Server (CFS) to match documents against categories before you index them, and to tag the document with the appropriate category. You can use AgentBoolean categories for more specific categorization. This method speeds up future searches for documents that match a category, because the document is already tagged.

You can also use this method to prevent documents from being indexed, based on the category data. For example, you can automatically prevent the IDOL Content component from indexing a document that contains sensitive or restricted material.

Related Topics

- Categorization, on page 157

Alert Users to Documents that Match Their Agents

Connector Framework Server (CFS) allows you to alert users to documents that match their agents before you index the documents. In this case, CFS can send a `TextParse` query to the IDOL Agentstore component to find all agents that match the document, and then email the users who own those agents.

`TextParse` queries allow you to send a whole IDX or XML document to Agentstore in a query. Agentstore extracts fields that you configure as `TextParseIndexType` from the document and uses the contents of these fields as the query text.

AgentBoolean rules improve the speed and accuracy of this agent matching procedure.

Related Topics

- Agents, on page 151

Configure IDOL Server for Text Parse Queries

IDOL Server can categorize documents or alert users to new content that matches their agents before it indexes the documents. This process includes matching against AgentBoolean rules.

You can configure your pre-indexing tasks to send the percent-encoded IDX or XML document as query text, with the `TextParse` action parameter set to **True**.

When you use an IDX or an XML document as query text, you must configure as `TextParseIndexType` all the fields that you use to match agents and categories.

IDOL stores agents and categories in the IDOL Agentstore component. Agentstore has its own configuration file, which by default is stored in the following location:

installDir\agentstore

To configure TextParse fields

1. Open the IDX or XML document that you want to match against the AgentBoolean categories in a text editor.

2. Decide which fields in the document contains the content that you want to match against the AgentBoolean categories (for example, the `DRECONTENT` and the `DRETITLE` field).

3. Open the IDOL Agentstore component configuration file in a text editor.

4. In the `[FieldProcessing]` section, add a new process to the list of processes. This process identifies the fields in the IDX or XML document that you want to match against the AgentBoolean categories. For example:

   ```
   [FieldProcessing]
   0=SetIndexFields
   ...
   15=SetAgentBooleanFields
   ```

5. Create a new section for the process that you added. In this section, create a property for the process (you define the property later, by setting one or more applicable configuration parameters). Set `PropertyFieldCSVs` to a list of fields to associate with the process. If you are not sure which fields to use, type */* to use all fields. For example:

   ```
   [SetAgentBooleanFields]
   Property=AgentBooleanFields
   PropertyFieldCSVs=*/DRECONTENT,*DRETITLE
   ```

6. Create a new section for the property that you added, in which you set `TextParseIndexType` to **True**. This property indicates that Agentstore must use the associated fields as query text in TextParse queries. For example:

   ```
   [AgentBooleanFields]
   TextParseIndexType=True
   ```

7. Save and close the configuration file.

8. Restart the IDOL Agentstore component for your changes to take effect.

Related Topics

- Fields, on page 81

Create AgentBoolean Agents and Categories

You can create `AgentBoolean` or `FieldText` agents and categories in IDOL Server in the same way as you create other agents. For example, you can send an `AgentAdd` action to the IDOL Community component, and add the `BooleanRestriction` or `FieldTextRestriction` parameter.

Alternatively, you can manually create an IDX document that contains agents or categories, and index it into the IDOL Agentstore component. For example:

```
#DREREFERENCE 947344A0
DRETITLE
```

```
Cat
#DREFIELD MyABField="(cat AND mat) AND ("furry kitten")"
#DREFIELD FieldTextField="MATCH{cat}:Animal"
#DRECONTENT
cat
#DREENDDOC
```

Related Topics

-

Perform AgentBoolean Queries

After you create or index AgentBoolean categories and agents, you can start querying them.

To match documents that already exist in the IDOL Content component against the categories and agents, use the following actions:

- `AgentGetResults`. Returns results for a particular agent.

- `CategorySuggestFromDocument`. Returns categories that match a particular document in your IDOL Content component index.

To match IDX or XML documents that do not exist in the IDOL Content component, you can use the following actions with the `TextParse` parameter:

- `CategorySuggestOnText`. Returns categories that match the text you provide.

- `Query`. Returns documents that match the text you provide.

  NOTE: You must send the `Query` action to the IDOL Agentstore component to return agents or categories.

To perform a TextParse query

1. Percent-encode the content of the IDX or XML document that you want to match against the AgentBoolean agents or categories. For example:

   ```
   #DREREFERENCE http://www.catdog.com
   #DRETITLE Cats and Dogs
   #DREFIELD Animal10="dog"
   #DREFIELD Animal11="cat"
   #DRECONTENT
   The organisation takes care of homeless cats and dogs
   #DREENDDOC
   ```

   Percent-encoding turns this IDX into this string:

   ```
   %23DREREFERENCE%20http%3A%2F%2Fwww%2Ecatdog%2Ecom%0D%0A
   %23DRETITLE%20Cats%20and%20Dogs%0D%0A
   %23DREFIELD%20Animal10%3D%22dog%22%0D%0A
   %23DREFIELD%20Animal11%3D%22cat%22%0D%0A
   ```

```
The%20organisation%20takes%20care%20of%20homeless%20cats%20and%20dogs%0D%0A
%23DREENDDOC
```

2. Copy the percent-encoded content string.

3. Send a `Query` action to the IDOL Agentstore component with the following parameters:

   - `Text`. Paste the percent-encoded content of the IDX or XML document to match against the AgentBoolean categories.

   - `TextParse`. Set this parameter to **True** to indicate that the specified `Text` is a percent-encoded document in IDX or XML format (it automatically detects the correct format).

   - `AgentBooleanField`. Set this parameter to the name of the AgentBoolean field to match against.

   - `DatabaseMatch`. Set this parameter to the database that contains agents or categories in the IDOL Agentstore component. By default, Agentstore databases are internal, so you must specify them explicitly.

   For example:

   ```
   action=Query&TextParse=True&AgentBooleanField=myABfield&DatabaseMatch=Activa
   ted
   &Text=%23DREREFERENCE%20http%3A%2F%2Fwww%2Ecatdog%2Ecom%0D%0A
   %23DRETITLE%20Cats%20and%20Dogs%0D%0A
   %23DREFIELD%20Animal10%3D%22dog%22%0D%0A
   %23DREFIELD%20Animal11%3D%22cat%22%0D%0A
   %23DRECONTENT%0D%0A
   The%20organisation%20takes%20care%20of%20homeless%20cats%20and%20dogs%0D%0A
   %23DREENDDOC
   ```

This query finds the categories that conceptually match the query text in the `Activated` Agentstore database. It then checks which of these categories contain a Boolean expression in their `myAbfield` that the fields in the percent-encoded document match.

> **NOTE:** Agentstore also returns agents and categories that match the query text and do not contain the AgentBoolean or FieldText field.

Agentstore returns only categories that match the document conceptually and contain a Boolean expression that matches the document fields. For example:

- Agentstore returns a category that conceptually matches the document if its `myABfield` contains, for example, one of these Boolean expressions:

  ```
  cat AND dog
  ```

  ```
  cat:DRETITLE AND dog
  ```

- Agentstore does not return a category that conceptually matches the document if its `myABfield` contains, for example, one of these Boolean expressions:

  ```
  cat AND mat
  ```

  ```
  cat AND dog:Animal10
  ```

  (because `Animal10` is not configured as a `TextParseIndexType` field).

Related Topics

Optimize AgentBoolean Matching

After you set up AgentBoolean and FieldText matching, you can optimize the performance.

Configure AgentBoolean Cache Fields

When you create categories and agents in IDOL Server, the Category and Community components send these agents and categories to the IDOL Agentstore component.

If you manually create agents and categories and want to use a custom field for Boolean and FieldText restrictions, you can configure these fields in the IDOL Agentstore component. This configuration optimizes AgentBoolean and FieldText matching for those fields.

To configure AgentBoolean cache fields

1. Open the IDX or XML agents or categories that you want to index into IDOL Server.

2. Note the names of fields that contain the Boolean or FieldText restrictions.

3. Open the IDOL Agentstore component configuration file in a text editor.

4. In the `[Server]` section, set `AgentBooleanCacheField` to the name of the field that contains the Boolean restrictions.

5. Set `FieldTextCacheField` to the name of the field that contains the FieldText restrictions.

   > **NOTE:** You can have only one field in the `AgentBooleanCacheField` and `FieldTextCacheField` parameters.

6. Save and close the configuration file.

7. Restart the IDOL Agentstore component for your changes to take effect.

Index a Dummy IDX

If your AgentBoolean expressions contain field restrictions, you might need to index a dummy IDX document to ensure that the IDOL Agentstore component contains all the fields that you use for field restrictions in your agents and categories.

For example, if your agents and categories contain the FieldText expression `MATCH{dog}:Animal10`, Agentstore must include the `Animal10` field to optimize restrictions.

Determine Whether a Dummy IDX is Required

When you index an agent document, Agentstore attempts to automatically create the fields that you use in your field restrictions in the configured `FieldTextCacheField` and `AgentBooleanCacheField`.

For simple restrictions that do not include wildcards (for example `dog:Animal10`), or that include only a leading `*/` wildcard, Agentstore can create the required fields and you do not need to use a dummy document.

However, if you use wildcards that can expand to different field names (for example `Animal*`), you must use a dummy document to add all the relevant fields to your index.

> **IMPORTANT:** Agentstore creates new fields under any document root fields that exist in the agent index (that is, the current document root and any fields that match `DocumentDelimiterCSVs`).
>
> Agentstore does not add the field to every possible XML field path. For example, if your field restriction contains the `NAME` field, Agentstore does not match a value in `XML/DOCUMENT/USER/NAME`. In this case you can:
>
> - use `USER/NAME` in your restriction (and configure `XML/DOCUMENT` as a document root, if it not already)
>
> - index an XML dummy document that includes `USER/NAME` as a field.

Create and Index the Dummy IDX

A dummy IDX document contains all fields that you use in field restrictions, with empty values. For example, create a copy of your alert documents, and remove the values from each field. For example:

```
#DREREFERENCE dummy document
#DRETITLE DummyDocument
#DREFIELD Animal10=""
#DREFIELD Animal11=""
#DRECONTENT
#DREENDDOC
```

To ensure that Agentstore does not subsequently return the dummy document in queries, index it into the `Deactivated` Agentstore database. You can delete the document after indexing.

> **NOTE:** You must index the dummy IDX before you index or create the agents.

Related Topics

-

Customize Agent Content

In large systems that experience heavy load, you can optimize the performance of AgentBoolean queries by creating optimized agents or categories. This optimization is most useful when you use IDOL Server to:

- Return agents that match a non-indexed document, for example to alert users to new content that matches their agents.

- Return categories that match a non-indexed document, for example to categorize a document and add the category information to a field before indexing.

When Agentstore matches query text against agents or categories, it uses the following matching order:

1. It matches the query text against the `Index` fields in the agents or categories (for example the `TRAINING` or `DRECONTENT` fields).

2. For agents that match in Step 1, it matches the query text against any Boolean restrictions in the agents or categories.

3. For agents that match in Step 2, it matches the query text against any `FieldText` restrictions in the agents or categories.

Agentstore checks the Boolean restriction only if the agent or category content matches the document.

To optimize performance, choose your category or agent content carefully to ensure that it matches only query text that is also likely to match the Boolean expression.

Use a Minimal List of Rare Terms

For many Boolean restrictions, you can select one or two terms that queries must contain to match the Boolean expression. If you select the rarest of these combinations, fewer documents match the agent content. This process reduces the number of Boolean expressions that Agentstore must match.

For example, if your Boolean restriction is:

`"New Zealand" AND "French Champagne"`

you can set the agent content to:

`Zealand`

In this Boolean expression, the query text must contain all four terms, so you can choose any of them as the agent content. `Zealand` is the rarest term, so fewer documents match this agent.

For other expressions, you can often similarly choose a minimal list of terms to reduce the number of documents that match against the Boolean restriction.

To find the minimal list of terms

- Send the `DocumentStats` action with the `QueryStats` parameter set to **True**. For example:

  `http://localhost:9050/action=DocumentStats&Text="New Zealand" AND "French Champagne"&QueryStats=True`

This action returns a minimal list of terms that text must contain to match the Boolean expression. It provides the rarest terms where possible. The action returns the following information:

```
<autnresponse>
   <action>DOCUMENTSTATS</action>
   <response>SUCCESS</response>
   <responsedata>
      <autn:wildrequired>false</autn:wildrequired>
      <autn:numberrequired>1</autn:numberrequired>
      <autn:required>ZEALAND</autn:required>
   </responsedata>
</autnresponse>
```

Using this method to choose the agent or category content can improve performance, especially in systems that receive a large number of queries.

Use AlwaysMatchType Fields

For some Boolean expressions that include Wildcards and `DREFUZZY` or `SOUNDEX` expressions, the minimal list of terms must include a Wildcard value.

For these expressions, the `DocumentStats` action with `QueryStats` set to **True** returns the following line:

`<autn:wildrequired>`**true**`</autn:wildrequired>`

The agent or category content cannot contain a Wildcard value, so you must use a different approach to ensure that Agentstore checks the Boolean expression.

You can ensure that Agentstore checks any documents that contain one of these expressions by configuring an `AlwaysMatchType` field in the Agentstore component and adding it to the agents.

Agentstore always matches the content of an agent or a category that contains an `AlwaysMatchType` field. It then attempts to match the query text against the Boolean restriction, and returns any agents or categories that match at this stage.

To configure AlwaysMatchType fields

1. Open the IDX or XML files that contain your AgentBoolean documents.

2. Decide which fields you want to always match. For a document to always match, the `AlwaysMatchType` field must have a non-empty string.

   > **NOTE:** For IDOL Server categories, you must use the `NOCONTENTCAT` field.

3. Open the IDOL Agentstore component configuration file in a text editor.

4. In the `[FieldProcessing]` section, add a new process to the list of processes. This process identifies the fields in the IDX or XML document that you want to always match. For example:

   ```
   [FieldProcessing]
   0=SetIndexFields
   ...
   15=SetAlwaysMatchFields
   ```

5. Create a new section for the process that you added. In this section, create a property for the process (you define the property, later by setting one or more applicable configuration parameters). Set `PropertyFieldCSVs` to a list of fields to associate with the process. For example:

   ```
   [SetAgentBooleanFields]
   Property=AlwaysMatchFields
   PropertyFieldCSVs=*/MARKERFIELD,*/NOCONTENTCAT
   ```

6. Create a new section for the property that you added, in which you set `AlwaysMatchType` to **True**. This property indicates that Agentstore must always match the associated fields in queries. For example:

```
[AlwaysMatchFields]
AlwaysMatchType=True
```

7. Save and close the configuration file.

8. Restart the IDOL Agentstore component for your changes to take effect.

After you configure the `AlwaysMatchType` field, you must manually create agents as IDX or XML documents.

For each agent, send the Boolean restriction in a `DocumentStats` action to IDOL Server to return the minimal list of terms.

- If the result does not contain a Wildcard value, you can create agents in the normal way.

- If the result contains a Wildcard, add the `AlwaysMatchType` field to the agent IDX. For example:

```
#DREREFERENCE WineAgent
#DRETITLE My Wine Agent
#DREFIELD BOOLEANRESTRICTION=""Champ*" AND "Merlot""
#DREFIELD MARKERFIELD="1"
#DRECONTENT
#DREENDDOC
```

You can index your agent IDX or XML document into the Agentstore component agent database by using a `DREADD` or `DREADDDATA` index action. For example:

```
http://localhost:9051/DREADD?C:\data\Agents.idx&DREDBName=Agent
```

> **NOTE:** Micro Focus recommends that you do not use `AwaysMatchType` fields without also specifying either the Boolean or FieldText restriction. If you add an `AlwaysMatchType` field to the agent or category without any other restriction, it returns for every query.
>
> If you do have agents or categories of this kind, you can use `FieldText` in your agent or category query to filter them out. For example:
>
> ```
> FieldText=EXISTS{}:BOOLEANRESTRICTION OR EXISTS{}:FIELDTEXTRESTRICTION
> ```
>
> This example restricts the results of the query to agents and categories that have one of the restriction fields.

Related Topics

- Fields, on page 81

Chapter 12: Cluster Information

The IDOL Category component can automatically cluster information to make trends and developments in this information visible. Clustering is the process of taking a large repository of unstructured data and automatically partitioning it, so that similar information is clustered together. Each cluster represents a concept area in the knowledge base, and contains a set of items with common properties.

> **NOTE:** The IDOL Category component provides several methods of clustering, including dynamic clustering and index tagging. See Cluster Results, on page 327, and Tag Documents into Clusters, on page 77.
>
> For details of all the available types of clustering, and information about which one to choose, refer to *IDOL Expert*.

Generate Snapshots

To cluster information, take a snapshot of data in your IDOL data index. You can then automatically cluster the data in this snapshot (you do not need to set up an initial taxonomy).
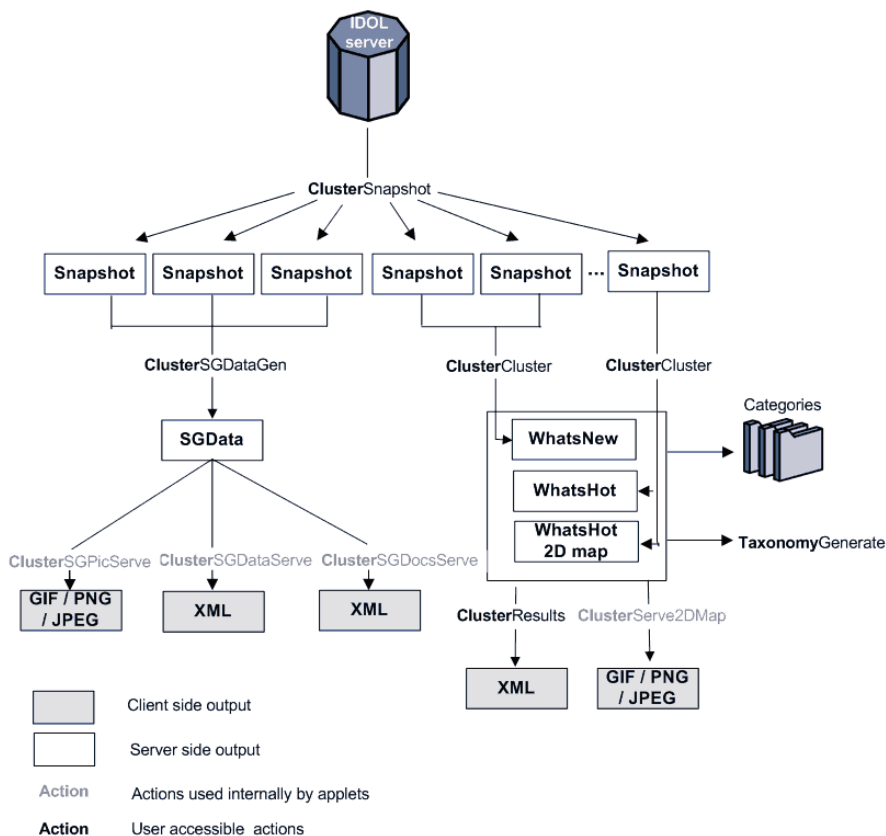
The IDOL Category component takes a snapshot of the data in your IDOL Content component and, based on these snapshots, clusters related information together. Each cluster represents a concept area that contains a set of items, which share common properties.
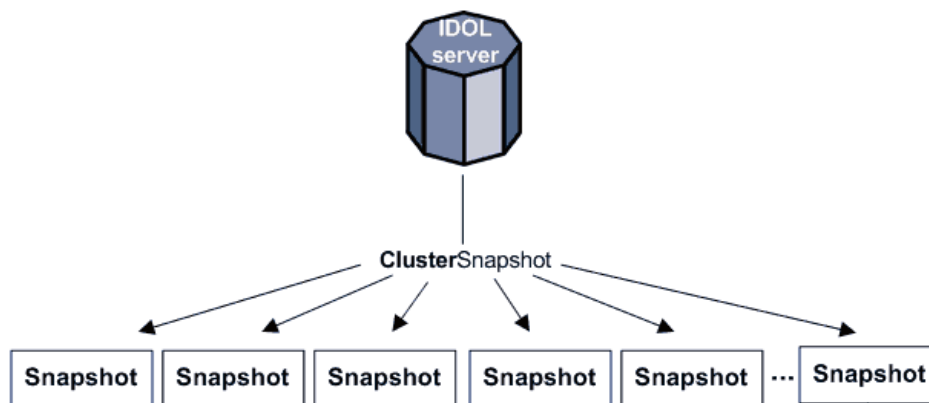
The `ClusterSnapshot` action allows you to take a snapshot of the data stored in the IDOL Content component data index. By default this includes data for the Content databases `News` and `Archive`. A snapshot represents the content of the data index at a particular time, and enables you to generate cluster information and spectrographs at a later point, even if the data index has changed. You can use a single snapshot to generate both cluster information and spectrograph data to save processing time.

The action adds a timestamp to each snapshot (with the `AUTNDATE`) and stores it in binary `.cls` format in the `Snapshots` subdirectory of the `Cluster` directory in your IDOL Category component installation directory. This process allows you to have several snapshots with the same name (for example, of one

particular IDOL Content component) and snapshots with different names (for example, of different data sets).

The results of `ClusterSnapshot` are saved as a named *snapshot job*. You can specify that job name when taking other actions on the snapshot data. You can also set up a schedule that runs the `ClusterSnapshot` action at regular intervals.

> **NOTE:** The Content data index that you take a snapshot of must ideally contain at least several thousand documents with good quality content (that is, relevant text for various topics).

Related Topics

Generate Spectrograph Data

The `ClusterSGDataGen` action allows you to generate spectrograph data from a set of snapshots that you took using the `ClusterSnapshot` action.
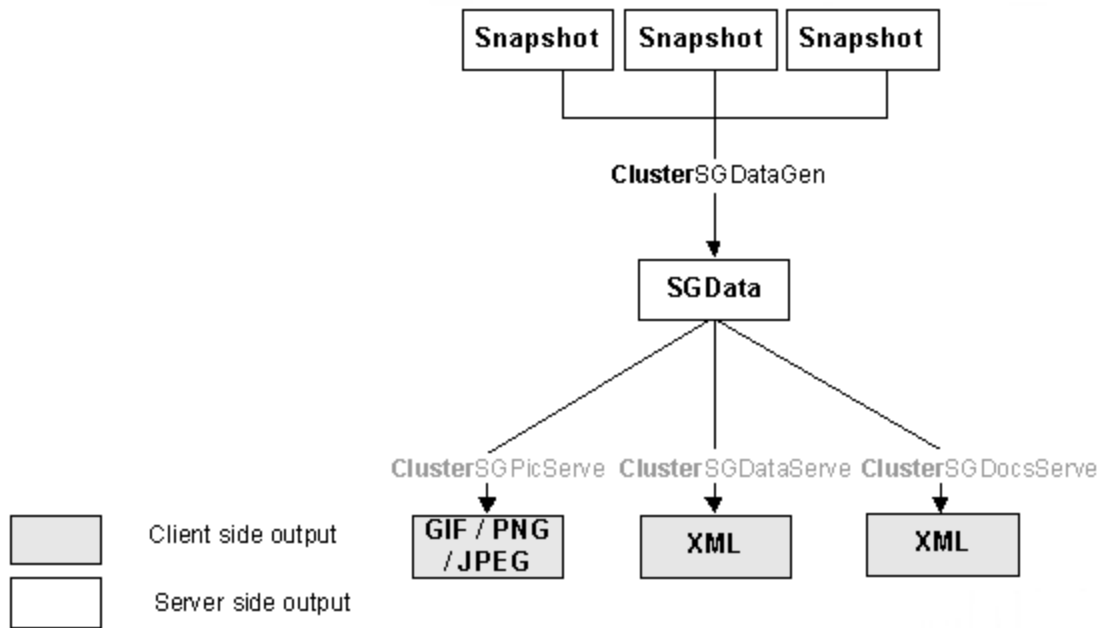
Each spectrograph dataset takes a succession of clusters from different time periods, calculates cluster similarity measures across days, and applies a graph theoretic matching algorithm. The IDOL Category component makes calculations about the conceptual spread of a cluster and its general quality. The spectrograph uses lines to represent the size (number of documents in a cluster) and quality of a cluster. The brighter a spectrograph line is, the more documents the cluster contains, and the thicker the line is, the higher the cluster quality.

All spectrograph data sets that you generate are stored in the `Sgdata` subdirectory of the `Cluster` directory in your IDOL Category component installation directory.

You can set up a schedule that runs the `ClusterSGDataGen` action at regular intervals.

You can retrieve the spectrograph image, data, or documents by using the `ClusterSGPicServe`, `ClusterSGDataServe`, and `ClusterSGDocsServe` actions.

For more information about the `ClusterSGDataGen` action and the results that it returns, refer to the *IDOL Server Reference*.

Related Topics

Generate WhatsNew and WhatsHot Information

The `ClusterCluster` action allows you to analyze clusters in a snapshot that you took using the `ClusterSnapshot` action.
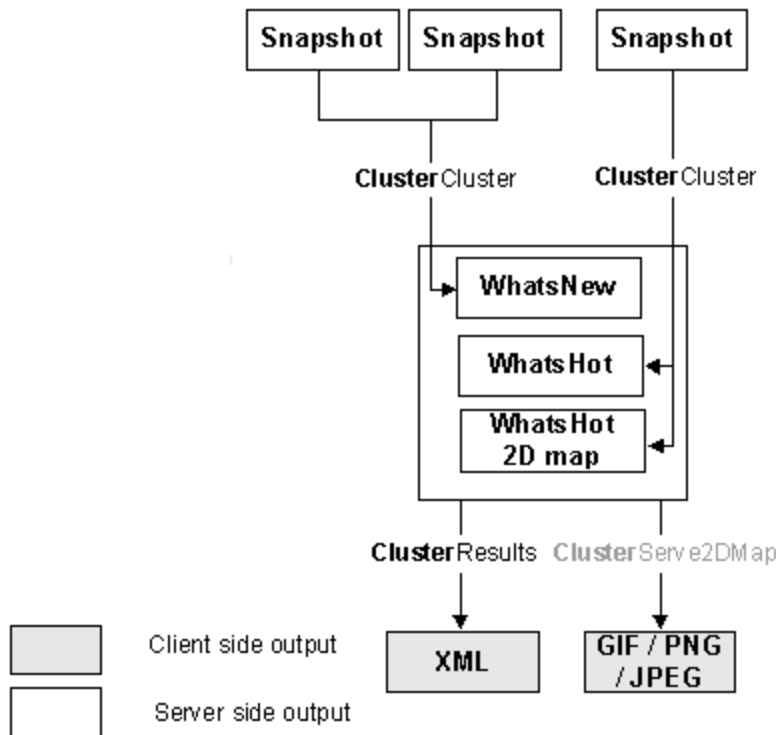
Clustering is a multistage hybrid algorithm. After the IDOL Adaptive Probabilistic Concept Modelling (APCM) technology identifies similar documents, a hierarchical agglomerative clustering algorithm groups documents into conceptually similar areas. Dynamic binding and fixating produces the required clusters. The title is generated automatically by cross-correlating important concepts in a cluster with concepts in the titles of documents in that cluster.

The IDOL Category component saves the results of clustering as a named *cluster job*. You can specify that job name when taking other actions on the clustered data. You can also set up a schedule that runs the `ClusterCluster` action at regular intervals.

Depending on which parameters you combine the action with, you can generate WhatsNew or WhatsHot information.

Related Topics

- Set up Schedules, on page 205

WhatsNew Information

WhatsNew information is the latest information that is available for the clusters that Category identifies in your snapshot.

You can generate WhatsNew information by comparing two snapshots (that have the same name or different names).

Category saves the results of the `ClusterCluster` action in configuration files (`.cfg`) in the `Clusters` subdirectory of the `Cluster` directory of your IDOL Category component installation. You can use the `ClusterResults` action to retrieve them in XML format.

If you configured the `ClusterCluster` action to generate a 2-D map of `WhatsHot` cluster information, you can use the `ClusterServe2DMap` action to return this map in one of the supported image formats (that is, GIF, PNG or JPEG).

WhatsHot Information

WhatsHot information is the most relevant information that is available for the clusters that the IDOL Category component identifies in your snapshot. Unlike WhatsNew information, this is not restricted to new information, which means that it can follow the progress of particular news items over time.

You can cluster WhatsHot information from a snapshot display this information in your user interface. You can also generate a 2-D map from WhatsHot information.

The 2-D map gives a visualization of the similarities and difference between clusters. Category uses a dimensionality reduction algorithm to maintain intercluster similarity measures so that similar clusters are close together and very different clusters are not close together. Category uses the distribution of documents throughout the space, along with nonlinear remapping, to create the landscape.

Generate a Cluster Map after You Cluster

Typically, to create a 2-D map from your clustered data, you send a `ClusterCluster` action with the `DoMapping` parameter set to **True**. You then send the `ClusterServe2DMap` action.

If you clustered your data with `DoMapping` set to **False**, you can still generate a cluster map of the data by using the `ClusterMapFromResults` action.

You pass the job name from your `ClusterCluster` action as the `SourceJobName` parameter to `ClusterMapFromResults`, and it returns a 2-D cluster map as binary image data.

Related Topics

- Create a Cluster Map, on page 341

Configure Clusters

You can take a snapshot of the data content that IDOL Server stores. This snapshot identifies clusters of conceptually similar documents, which enables you to generate a view of trends in the data. You do not need to generate an initial taxonomy to take a snapshot.

A set of data can contain a few large clusters or many small clusters, as well as several outliers that are not part of any cluster. Clusters can consist of highly similar documents or of less closely related ones. What constitutes optimal clustering depends on how you intend to use your clusters. However, the aim of clustering is always to generate an accurate characterization of the data content in your IDOL Server.

By default IDOL Server uses internal settings to produce clusters. You do not usually need to change these default settings. However, in some cases you might require more or less detail in your clusters, or the amount and nature of your data might mean that default clustering is not satisfactory.

You can adjust the size of the units on which to base clusters, the degree of conceptual similarity that documents within clusters must have, or the number of clusters to create.

Change the Number and Size of Clusters

There are two main stages to the clustering process:

- Build Seeds
- Group Seeds into Clusters

Build Seeds

The IDOL Category component builds seeds when you send the `ClusterSnapshot` action. Category takes a sample of the documents that it stores, and tries to associate individual documents with each other, based on the similarity of the concepts that the documents contain. Each group produced at this stage, containing a sample document and similar documents, is a seed.

Category stops trying to build a seed when the seed meets the requirements that `SeedSize` specifies or when there are no more documents that meet the similarity requirement that `SeedBindLevel` specifies (whichever condition is reached first). Category discards any seeds that do not reach the required size.

The number of clusters that you specify with `NumClusters` affects the number of sample documents that Category tries to create seeds from. You can adjust the relationship between the number you specify here and the size of the sample used by changing the value of `StartingSuggestOverrideFactor`.

Group Seeds into Clusters

Category groups seeds into clusters when you send the `ClusterSGDataGen` or `ClusterCluster` actions. Category tries to create clusters by grouping seeds together. The grouping is based on the similarity of the concepts that the seeds or clusters contain.

Clustering is complete when one of the following conditions is met:

- Category creates the number of clusters specified by `NumClusters`.

- Category cannot create any more clusters that meet the similarity requirement specified by `BindLevel`.

Category discards clusters that do not meet the quality requirement set by `BindLevel` or the size requirement set by `MinClusterDocs`.

For details of the clustering actions, and the settings you can make to generate the clusters from your data, refer to the *IDOL Server Reference*.

Related Topics

- [Display Online Help, on page 30](#)

Configuration Parameters

The ideal values for the parameters that affect clustering depend on the nature and amount of data in your IDOL Content component data index. You can use the `SentientClustering` parameter for the `ClusterSnapshot` action to automatically determine the correct values for `SeedSize` and `SeedBindLevel`.

This section makes general recommendations about how to manually alter these parameters according to your data. Parameters are closely interdependent, so make these changes in combination with each other (rather than just changing one of the settings). Change values in small steps.

Although you can make many changes to clustering, the number and size of clusters that the IDOL Category componentcan identify depends ultimately on the data content that it contains. You can:

- cluster a small amount of data.

- cluster a large amount of data.

- cluster very similar data.

- cluster very different data.

- change the data view.

Cluster a Small Amount of Data

If your IDOL Content component has a small amount of data, it is likely to identify fewer clusters, because it is less likely that your data contains a lot of similar documents for several different topics. You can edit the following parameters to change clustering in this situation.

> **NOTE:** Ideally, your IDOL Content component must contain at least 500 documents.

`SeedSize`	Decrease `SeedSize` (by three to four points at a time). This option reduces the size that seeds must reach, which means that more seeds are likely to be successfully created.
`MinClusterDocs`	Decrease `MinClusterDocs` so that clusters that contain fewer documents are not discarded.

StartingSuggestOverrideFactor	Increase StartingSuggestOverrideFactor (by one or two points only). This increases the number of sample documents from which Category creates seeds, which in some cases increases the possibility of finding clusters in the data.
SeedBindLevel	Decrease SeedBindLevel (by one point at a time) to reduce the similarity threshold for clusters. Do not change this value until you try changing SeedSize, because lowering SeedBindLevel is more likely to allow less-relevant documents into clusters.

Cluster a Large Amount of Data

If your IDOL Content component has a large amount of data, you probably do not need to edit any clustering parameters, because this is the situation in which clustering is most successful. In some cases (for example, if Content contains more than a million documents), it can be beneficial to alter the following parameter.

StartingSuggestOverrideFactor	Increase the value of this parameter to increase the number of sample documents from which Category creates seeds. This is sometimes necessary to allow a broader section of the data content to be represented by the clusters that are created.

Cluster Very Similar Data

If the documents in your IDOL Content component contain highly similar concepts, the IDOL Category component might identify a small number of large clusters. For example, if your IDOL Content component contains mostly documents about sports, then you might get one large *sports* cluster. This situation is a realistic characterization of the data in your data index, but in many circumstances is not useful. You can edit the following parameters to generate smaller, more specific clusters (for example, breaking *sports* into *football*, *tennis*, *golf*, and so on).

SeedBindLevel	Increase SeedBindLevel to require greater similarity between the documents that form a seed, which can reduce the breadth of topics covered by the concepts in the seed documents.
	NOTE: Increase SeedBindLevel one point at a time. Increasing by too much can result in seeds being discarded because they do not contain enough documents.
BindLevel	Increase BindLevel to require greater similarity between the concepts in seeds or clusters that merge to create a cluster. This change can decrease the size of clusters, as well as increase the number of clusters identified, because merging seeds and clusters together stops at an earlier stage.

Cluster Very Different Data

If the documents in your IDOL Content component contain a wide variety of concepts, there might not be enough similar documents for Category to create seeds or clusters that characterize the data that it stores. You can lower the similarity requirement with the following parameters.

SeedBindLevel	Decrease SeedBindLevel to reduce the similarity requirement between the documents that form a seed, which can increase the breadth of topics covered by the concepts in the seed documents. **NOTE:** Decrease SeedBindLevel one point at a time. Decreasing by too much can result in seeds and clusters that contain documents that are less relevant, because the similarity requirement is too low.
BindLevel	Decrease BindLevel to reduce the similarity requirement between the concepts in seeds or clusters that merge to create a cluster. This change can increase the size of clusters, as well as increase the number of clusters identified (because fewer are discarded for not meeting the quality requirement).

Change the Data View

It might be the case that although Category identifies clusters that characterize your data successfully, you want to change the view of the data that clustering creates. The following parameters enable you to change the data view that clusters generate.

NumClusters	Increase NumClusters to obtain a more low-level view of your data by identifying more clusters. Decrease NumClusters to obtain a more high-level view by identifying fewer clusters.
MinClusterDocs	Decrease MinClusterDocs to reduce the number of clusters that are discarded. This option allows you to identify smaller clusters. Increase MinClusterDocs to increase the number of clusters that it discards. Only larger clusters are kept.
BindLevel	Decrease BindLevel to reduce the similarity requirement between the concepts in seeds or clusters that merge to create a cluster. This option can increase the size of clusters, as well as increase the number of clusters identified (because it discards fewer clusters for not meeting the quality requirement). Increase BindLevel to increase the similarity requirement between the concepts in seeds or clusters that merge to create a cluster. This can decrease the size of clusters, as well as increase the number of clusters identified, because merging seeds and clusters together stops at an earlier stage.

Set up Schedules

You can set up to 1,024 schedules, which allow you to run the following actions at regular intervals.

- ClusterSnapshot
- ClusterCluster
- ClusterSGDataGen
- TaxonomyGenerate

For details on the settings that each `[AnalysisSchedule]` section can contain and on how you can configure them, refer to the *IDOL Server Reference*.

Related Topics

To set up schedules

1. Open the IDOL Category component configuration file in a text editor.

2. Create an `[AnalysisScheduleN]` section for each schedule that you want to run. Start numbering the `[AnalysisScheduleN]` sections from zero (so that the first schedule section is `[AnalysisSchedule0]`). For example:

```
[AnalysisSchedule0]
[AnalysisSchedule1]
[AnalysisSchedule2]
[AnalysisSchedule3]
[AnalysisSchedule4]
[AnalysisSchedule5]
```

In this example six schedules were created. Note that the schedules are listed in consecutive order, starting from zero.

3. Specify the settings for each schedule in the appropriate section. You can specify the action to schedule, the interval in which each schedule runs, the number of times each schedule runs, what job name to give the action, and so on. For example:

```
[AnalysisSchedule0]
ScheduleStartTime=now
ScheduleInterval=1 day
ScheduleCycles=1
ScheduleAction=ClusterSnapshot
TargetJobName=myjob

[AnalysisSchedule1]
ScheduleStartTime=now
ScheduleInterval=1 day
ScheduleCycles=1
ScheduleAction=ClusterCluster
SourceJobName=myjob
TargetJobName=myjob_clusters
DoMapping=True

[AnalysisSchedule2]
ScheduleStartTime=now
ScheduleInterval=1 day
ScheduleCycles=1
ScheduleAction=ClusterCluster
SourceJobName=myjob
TargetJobName=myjob_clusters_new
WhatsNew=True
```

```
        Interval=86400

        [AnalysisSchedule3]
        ScheduleStartTime=now
        ScheduleInterval=1 day
        ScheduleCycles=1
        ScheduleAction=ClusterSGDataGen
        Interval=604800
        SourceJobName=myjob
        TargetJobName=myjob_sg

        [AnalysisSchedule4]
        ScheduleStartTime=now
        ScheduleInterval=1 day
        ScheduleCycles=1
        ScheduleAction=ClusterSGDataGen
        Interval=86400
        SourceJobName=myjob_content
        TargetJobName=compare_snapshots_sg

        [AnalysisSchedule5]
        ScheduleStartTime=now
        ScheduleInterval=1 day
        ScheduleCycles=1
        ScheduleAction=TaxonomyGenerate
        Cluster=0,1,2,3,4,5,6,7,8,9
        SourceJobName=myjob_clusters
        TargetJobName=myjob_taxonomy
        WriteTaxonomy=true
        NumResults=25
```

4. Save and close the configuration file.

5. Restart the IDOL Category component for your changes to take effect.

> **TIP:** As an alternative to setting up schedules in the configuration file, you can also use the
> **Category Queue** page in the Control section of IDOL Admin to schedule `ClusterSnapshot`,
> `ClusterCluster`, `ClusterSGDataGen`, and `TaxonomyGenerate` actions. For more information, refer
> to the *IDOL Admin User Guide*.

Partition Documents into Clusters

Partition clustering is a different method of clustering that allows you to create a specified number of
clusters, and to place all documents into a cluster. This method is faster than using `ClusterSnapshot`.

This method adds all documents into a cluster, which means that the clusters generated are relatively
broad and might represent several topics. You might want to use this method if you want to cluster the
results of a query into a predefined number of clusters.

To partition documents into clusters

- Send the `ClusterPartition` action to the IDOL Category component. Set the following action parameters:

DREQuery	The query that you want to cluster the results for.
NumResults	The number of results from the specified query that you want to return and cluster.
NumClusters	The number of clusters to create.

For more information about the `ClusterPartition` action, refer to the *IDOL Server Reference*.

Chapter 13: Profiles

This section describes how to create interest and expertise profiles for users for collaboration and locating experts.

About Profiles

IDOL Server automatically creates interest and expertise profiles for users in real time. You can configure the IDOL Community component to create up to four different profile types. By default, Community creates an interest and an expertise profile for each user.

You can create an interest profile by tracking the content that a user views and extracting a conceptual understanding of it. Community then uses this understanding to keep the user interest profile up to date. You can use interest profiles to target information at users, recommend content to users, alert users to the existence of content, and put users in touch with other users with similar interests.

You can create an expertise profile by tracking the content that a user creates and extracting a conceptual understanding of it. Community then uses this understanding to keep the expertise profile for that user current. You can use expertise profiles to trace users who are experts in particular subject areas.

Profile a User

You can use the `ProfileUser` action to profile a user. For details on this action, refer to the *IDOL Server Reference*.

Related Topics

- Display Online Help, on page 30

Create an Interest Profile for a User

Run the `ProfileUser` action when a user views a document. The IDOL Community component analyzes the document that the user is viewing, and determines if it is similar to any of the concepts in their interest profile (using `MatchThreshold`).

If the content of the viewed document is similar to an existing interest profile concept, Community updates the existing concept with the new document. If several concepts are similar, only the most

similar one updates. If the document content is not similar to an existing interest profile concept, Community creates a new concept in the interest profile.

> **NOTE:** Community uses only the five strongest concepts in a user interest profile for recommendations, alerting, and similar user matching.

For example:

```
http://12.3.4.56:4000/action=ProfileUser&UserName=Administrator&Document=3422+5776&
MatchThreshold=60&NamedArea=Interest
```

This action instructs the IDOL Community component to analyze the content in the 3422 and 5776 documents. If the content has a conceptual relevance of at least 60 percent to a concept in the interest profile of the Administrator user, Community uses it to update the matching interest profile concept (if several concepts are similar, only the most similar one updates). If the document content does not have a conceptual relevance of at least 60 percent to an existing interest profile concept, Community creates a new interest profile concept from it.

Create an Expertise Profile for a User

Run the ProfileUser action when a user creates text (for example, a document in the IDOL Content component that was authored by a user, or text that a user enters in a help desk environment). The IDOL Community component analyzes the text that the user created, and determines if it is similar to any concepts in the existing expertise profile (using MatchThreshold).

If the content of the viewed text is similar to an existing expertise profile concept, Community updates the existing concept with the text (if several concepts are similar, only the most similar one updates). If the text is not similar to an existing expertise profile concept, Community creates a new concept in the expertise profile.

> **NOTE:** Community uses only the five strongest concepts in a user expertise profile for expertise matching.

For example:

```
http://12.3.4.56:4000/action=ProfileUser&UserName=Administrator&Document=The
chemical structure of everyone's DNA is the same. The only difference between
people (or any animal) is the order of the base pairs&
MatchThreshold=60&NamedArea=Expertise
```

This action instructs the IDOL Community component to analyze the specified text. If the text has a conceptual relevance of at least 60 percent to any concept in the Administrator user expertise profiles, Community uses it to update the matching expertise profile concept (if several concepts are similar, only the most similar one updates). If the text does not have a conceptual relevance of at least 60 percent to an existing expertise profile concept, Community creates a new expertise profile concept from it.

Manipulate Profiles

Manipulating profiles consists of editing, querying, viewing, and deleting them.

Related Topics

-

Edit a Profile

The IDOL Community component stores interest and expertise profiles in the IDOL Agentstore component in the form of terms and weights. You can use the `ProfileEdit` action to edit profile terms and weights. For details on this action, refer to the *IDOL Server Reference*. For example:

`action=ProfileEdit&PID=1-P2.3&TermCOLOR=2322`

This action changes the weight of the COLOR term in the `1-P2.3` profile to `2322`.

Query with a Profile

You can use the `ProfileGetResults` action to query with a profile. For details on this action, refer to the *IDOL Server Reference*. When you query with a profile, by default it matches against the IDOL Content component data index.

View Profile Details

You can use the `ProfileRead` action to view profile details. For details on this action, refer to the *IDOL Server Reference*. For example:

`action=ProfileRead&UserName=Administrator&PID=3422`

This action requests the details of the `3422` profile for the `Administrator` user.

Delete a Profile

You can use the `ProfileClear` action to delete a profile from the IDOL Server profile index (in the IDOL Agentstore component). For details on this action, refer to the *IDOL Server Reference*. For example:

`action=ProfileClear&UserName=Administrator&PID=450-P0.1`

This action deletes the `450-P0.1` profile for the `Administrator` user.

Collaboration and Expertise with Profiles

You can use IDOL Server profiles to collaborate with other users or to locate experts in your field of interest.

Collaboration

The IDOL Community component automatically matches users with common explicit interest agents or similar implicit profiles. You can use this information to create virtual expert knowledge groups.

You can use the `Community` action to find agents or profiles in the community that match the agents or the profiles of a specific user. For example:

```
action=Community&UserName=JSmith&Agents=True&Profiles=True&AgentsFindProfiles=True&
ProfilesFindAgents=True
```

This action finds agents and profiles in the user community that match both the agents and the profiles of the user `JSmith`.

Expertise

The IDOL Community component accepts a natural language or Boolean search string and returns users who own matching agents or profiles. This process allows you to identify experts in any subject, eliminating time-consuming searches for specialists and unnecessary researching of subjects for which expert knowledge is already available.

You can use the `Community` action to find agents or profiles in the community that match a natural language or Boolean search string.

For example:

```
action=Community&Text=how does the cost of funds, such as the costs of performing a
credit evaluation on the business requesting a loan, determine the spread between
the federal funds rate and the prime
rate&AgentsFindProfiles=True&ProfilesFindAgents=True
```

This action finds agents and profiles in the user community that match the specified text.

Part IV: Results

This section describes the processes of querying IDOL Server, retrieving results, and displaying those results to users.

- Search and Retrieve
- Customize Results
- Manipulate Result Relevance
- Manipulate the Results Set
- View Documents

Chapter 14: Search and Retrieve

You can search the IDOL Content component with actions by using a Web browser, an Micro Focus interface application (for example, IDOL Admin), or a third-party user interface application.

Actions

You query the IDOL Content component by using actions. The following actions are available to all users that have permission to query Content (users that belong to an authorization role with the `Query` standard role. See [AuthorizationRoles] Section, on page 482).

> **TIP:** You can view information on the permissions that are available on the **Overview** tab in the Status section of IDOL Admin, or by using the `ShowPermissions` ACI action.

`GetContent`	Displays the content of one or more specified documents.
`GetQueryTagValues`	Returns the values of parametric fields in query results.
`GetTagNames`	Returns all fields of a specified type.

GetTagValues	Performs a parametric search.
Highlight	Highlights link terms in text.
Query	Submits different query types to the IDOL Content component.
Suggest	Retrieves documents that are conceptually similar to one or more specified documents.
SuggestOnText	Retrieves documents that are conceptually similar to the terms with the highest weighting in specified text.
Summarize	Generates a summary for documents or text.
TermGetBest	Lists the conceptually most important terms in one or more specified documents.
TermGetInfo	Returns the weight and other available information for specified terms.

In addition, the following actions are available to administrative users of the IDOL Content component (users that belong to an authorization role with the Admin standard role. See [AuthorizationRoles] Section, on page 482).

DetectLanguage	Determines the language of a piece of text.
GetStatus	Displays configured details about the IDOL Content component setup.
IndexerGetStatus	Displays the status of index actions in the IDOL Content component index queue.
List	Lists all documents that are stored in the IDOL Content component or any of its databases.
TermGetAll	Lists all terms that are stored in the IDOL Content component .

Related Topics

- Display Online Help, on page 30
- Send Actions to IDOL Server, on page 30

Asynchronous Actions

By default, the IDOL Content component processes all actions synchronously. In this case, Content does not respond to the action until it has completed the request. The result of the action is in the response to the request.

With additional configuration, you can also process query actions asynchronously. In this case, Content responds to the action immediately. The request is added to a queue of actions. The response to the request contains a token. You can use this token to determine whether the request has finished, and obtain the results of the action. To do this, use the QueueInfo action.

You might want to use asynchronous actions to run operations that take a long time, and that do not require an immediate response. For example, you might have synchronous queries to return results to users, and then use an asynchronous action to run a query summary.

Configure Asynchronous Actions

You must configure the IDOL Content component to allow asynchronous actions.

To configure asynchronous functionality

1. Open the IDOL Content component configuration file in a text editor.

2. Create an `[Actions]` configuration section.

3. In the `[Actions]` section, add the `Async` parameter, and set it to **AsyncActions**.

4. Create an `[AsyncActions]` section.

5. In the `[AsyncActions]` section, set the `Threads` parameter to the number of threads that you want to use for asynchronous actions.

6. Save and close the configuration file.

7. Restart the IDOL Content component for your changes to take effect.

For example:

```
[Actions]
Async=AsyncActions

[AsyncActions]
Threads=2
```

Send Asynchronous Actions

To send an asynchronous action, you use the same action syntax as normal, but add the `Synchronous` parameter, set to **False**.

You can send the following actions asynchronously:

DetectLanguage	GetTagValues	Summarize
DocumentStats	Highlight	TermExpand
GetAllRefs	List	TermGetAll
GetContent	Query	TermGetBest
GetQueryTagValues	Suggest	TermGetInfo
GetTagNames	SuggestOnText	

Retrieve Results for Asynchronous Actions

The `QueueInfo` action allows you to retrieve the status and results of asynchronous actions.

To return the results of all asynchronous actions, send a `QueueInfo` action, with the `QueueName` parameter set to **Async**, and the `QueueAction` parameter set to **GetStatus**. You can set the `Token` parameter to the token for a specific action to retrieve results for only that action.

> **NOTE:** If you do not specify the `Token` parameter, all results for the server return. Micro Focus does not recommend that you return all results for a server that has been running for a long time, because the response might time out.

> **NOTE:** There are no user restrictions for viewing the queue list, so all users can see all results that the original action returns. Micro Focus does not recommend that you use asynchronous actions if you have strict document security requirements.

> **TIP:** You can use the `QueueInfo` action to return information about the queue, and to cancel or change the priority of actions. For more information, refer to the *IDOL Server Reference*.

You can use the Asynchronous Queues page in the Monitor section of IDOL Admin to view details of the asynchronous actions that the server is processing, view the results of those actions, and cancel actions. For more information, refer to the *IDOL Admin User Guide*.

Conceptual Matches

You can use `Query`, `Suggest`, and `SuggestOnText` actions to perform conceptual searches. The IDOL Content component uses advanced pattern-matching technology to conceptually match the data that you query with (using actions) against the content it holds.

Types of Matches

- **Content searches**. You can submit natural language text or a piece of content to the IDOL Content component. It returns references to conceptually related documents ranked by relevance or contextual distance.

  Natural language queries make it possible for users to find results without having to be familiar with search algorithms or syntax. Online shoppers, for example, can find specific items without knowing the exact product or brand name.

- **Community searches.** You can create agents from natural language and then match them conceptually. You can also submit profiles or natural language text to the IDOL Agentstore component. It returns agents ranked by conceptual similarity. This process determines which users have similar interests, which promotes collaboration, and identifies experts in a field.

- **Category searches**. You can submit a piece of content to the IDOL Agentstore component, for which it returns categories ranked by conceptual similarity. This process determines which categories the piece of content is most appropriate for, so that IDOL Server can tag, route, or file the piece of content accordingly.

- **Clusters**. You can use the IDOL Content component to organize large volumes of content or large numbers of profiles into self-consistent clusters. Clustering is an automatic agglomerative

technique that allows the IDOL Content component to partition data by grouping together information that contains similar concepts.

Example Queries

This section shows some examples of the different types of queries.

Agent or Category Query

You can add agent (or category) weights to terms in your query. The IDOL Agentstore component returns documents that contain highly weighted terms with a higher relevance than documents that contain only terms with lower weights. Use the `TermGetInfo` action to find out the weight of the term. Specify weights as a number in brackets next to the term. The maximum weight is `4095`, but Micro Focus recommends that you use a maximum weight of `511`.

For example:

```
http://localhost:9050/action=Query&Text=Cat[50] OR Dog
```

This query returns documents that contain the term *cat* or the term *dog*. If the term *dog* has a higher weight than `50` in Agentstore, documents that contain the term *dog* return with higher relevance than documents that contain only the term *cat*. If the term *dog* has a lower weight than `50`, documents that contain the term *cat* return with higher relevance.

```
http://localhost:9050/action=Query&Text=Cat[30] OR Dog[10]
```

This query returns documents that contain the term *cat* or the term *dog*. Documents that contain only the term *cat* return with three times the relevance than documents that contain only the term *dog*.

You can apply term weights to phrases. For example:

```
http://localhost:9050/action=Query&Text="cats and dogs"[100]
```

This query returns documents that contain the phrase `"cats and dogs"` and applies a weight of `100` to this phrase.

You can apply weights to terms in parentheses. In this case, the weight applies to all terms within the parentheses. For example:

```
http://localhost:9050/action=Query&Text=(cat OR dog)[100] AND (fish OR dolphin)
[150]
```

This query returns documents that contain at least one of the terms *cat* or *dog*, and at least one of the terms *fish* or *dolphin*. It assigns the terms *cat* and *dog* a weight of `100`, and it assigns the terms *fish* and *dolphin* a weight of `150`.

You can use multipliers to multiply the original term weight. Enter a term as *N*, where *N* is the amount to multiply by. You can use up to two decimal places. For example:

```
http://localhost:9050/action=Query&Text=cat[*2.25] OR dog[*0.5]
```

This query returns documents that contain the term *cat* or the term *dog*. The term *cat* has `2.25` times its normal relevance. The term *dog* has `0.5` times its normal relevance.

You can also use any combination of these examples to apply term weights to complex phrases or expressions. For example:

```
http://localhost:9050/action=Query&Text="cats and dogs"[*3] OR (fish[100]+dolphin
[150])[*1.5]
```

This query applies 3 times the normal weight to the phrase "cats and dogs". It assigns a weight of 100 to the term *fish* and a weight of 150 to the term *dolphin*. It then multiplies the weights for *fish* and *dolphin* by 1.5.

Profile Query

```
http://localhost:9050/action=Query&Text= CHAMPIONLEAGU~[551] EVERTON~[407] BAYERN~
[402] UEFA~[391] PREMIERSHIP~[383] FIFA~[257] STRIKER~[226] WORLDCUP~[215] EURO~
[124] SOCCER~[114] CUP~[66]
```

This action sends a profile query to the IDOL Agentstore component. The query contains the terms that the profile training contains, and the weight of each of the terms. Agentstore can return agents, profiles, categories, or documents that conceptually match the terms of the query.

Text Query

```
http://localhost:9010/action=Query&Text=Gene analysis discovered methods to
determine the exact sequence of nucleotides that compose a specific gene.
```

This action sends a text query to the IDOL Content component. Content can return agents, profiles, categories, or documents that conceptually match the query text.

Suggest Query

```
http://localhost:9010/action=Suggest&ID=10
```

This action sends a Suggest query to the IDOL Content component. Content can return agents, profiles, categories, or documents that conceptually match the specified document (that is, the document with the ID 10).

SuggestOnText Query

```
http://localhost:9010/action=SuggestOnText&Text=Gene analysis discovered methods to
determine the exact sequence of nucleotides that compose a specific gene
```

This action sends a SuggestOnText query to IDOL Content component. Content can return agents, profiles, categories, or documents that conceptually match the terms with the highest weighting in the query text.

Keyword Search

By default, the IDOL Content component conceptually matches queries that consist of a single keyword. It stems the keyword, and then it finds documents that contain words that have the same stem as the keyword.

For example, if you query Content with the word *lovely*, it stems the word to *love* and finds documents that contain words that also stem to *love*, for example, *lovely*, *love*, *loved*, *loving*, and so on.

Keyword Occurrence Search

You can restrict the documents that match a query term to documents in which the number of occurrences of the term or phrase is within a specified range. Specify the range with a colon-separated pair of numbers. The maximum value that you can specify in the range is `32000`.

For example:

```
action=Query&Text=Gene[3:7]
```

This query returns only documents in which *Gene* appears between three and seven times (inclusively).

```
action=Query&Text="Gene Therapy"[5:8]
```

This query returns only documents in which the phrase *Gene therapy* appears between five and eight times (inclusively).

```
action=Query&Text=Gen*[2:4]
```

This query returns only documents in which a term that starts with *Gen* (such as *Gene* or *Generation*) appears between two and four times, inclusively.

```
action=Query&Text=(Gene OR DNA)[4:8] AND Therapy
```

This query returns only documents in which the term *Gene* or the term *DNA* appear between 4 and 8 times (inclusively), and the term *Therapy* occurs.

You can specify open-ended ranges. For example:

```
action=Query&Text=Gene[10:]
```

This query returns only documents in which *Gene* appears 10 or more times.

You can also specify term weights and field restrictions. For example:

```
action=Query&Text=cat[2:4]:DRETITLE + AND + cat[100][2:4]
```

Exact Keyword Search

To find documents that contain exact matches of a keyword, enable the `AdvancedSearch` setting before you index content into the IDOL Content component.

If you set `AdvancedSearch` to **True** in the `[Server]` section of the IDOL Content component configuration file before you index content, you can query for exact matches of keywords by putting the keyword in quotation marks when you perform the query (this setting also switches Content to an advanced weighting algorithm which improves conceptual querying).

For example, if you query Content with **"lovely"**, it finds only documents that contain the exact term *lovely*.

If you do not put the word in quotation marks, Content matches it conceptually; that is, the query *lovely* matches documents that stem to the same value as *lovely* (for example, it matches documents that contain, *lovely*, *love*, *loved*, *loving*, and so on).

You can also suffix a keyword with a tilde (~) to indicate that the term is already stemmed. In this case, Content does not stem the query term. However, unlike an exact keyword search using quotation marks, this query might return documents that contain other terms that stem to your query term. For example, searching for `Love~` returns documents that contain the term *love*, but also other terms that stem to *love*, for example, *lovely*, *love*, *loved*, and *loving*.

> **NOTE:** Using the tilde suffix to search for a term that is not its own stem does not return documents that contain the original term. For example, searching for `loving~` matches terms that stem to *loving*. The term *loving* stems to *love*, so this search does not return documents that contain the term *loving*.
>
> You can use the `TermGetInfo` action to find the stem of a term.

Case-Sensitive Exact Keyword Search

To find documents that contain case-sensitive exact matches of a keyword, enable the `AdvancedCaseSearch` setting before you index content into the IDOL Content component:

If you set `AdvancedCaseSearch` to **True** in the IDOL Content component configuration file `[Server]` section before you index content, you can query for case-sensitive exact matches of keywords by prefixing the keyword with a tilde (~) and putting it in quotation marks when you perform the query.

For example, if you query Content with **"~Lovely"**, it does not stem the word, and finds only documents that contain *Lovely*.

If you put a word into quotation marks but do not prefix it with a tilde (~), Content matches it exactly but not case-sensitively (that is, the query **"Lovely"** matches documents that contain, for example, *Lovely*, *lovely*, *lOveLy*, and so on).

If you prefix a word with a tilde (~) and do not put it into quotation marks, Content matches it conceptually and case-sensitively (that is, the query **Lovely** matches documents that contain, for example, *Lovely*, *Love*, *Loved*, *Loving*, and so on).

For more information about case-sensitive searching, refer to *IDOL Expert*.

Paragraph and Sentence Keyword Search

To find documents that contain groups of keywords within the same sentence or paragraph, enable the `AdvancedPlus` setting before you index content into the IDOL Content component. This parameter also enables the functionality of `AdvancedSearch` and `AdvancedCaseSearch`.

To query for keywords in the same paragraph or sentence by using the `PARAGRAPH` and `SENTENCE` operators, set `AdvancedPlus` to **True** in the `[Server]` section of the IDOL Content component configuration file before you index content.

For example, if you query Content with **cat SENTENCE dog**, it returns only documents that contain the word *cat* and the word *dog* in the same sentence. If you query Content with **cat PARAGRAPH dog**, it returns only documents that contain the word *cat* and the word *dog* in the same paragraph.

Keyword Search Examples

The word *lovely* stems to *love*. The following words also stem to *love*.

> **TIP:** To determine the stem of a word, use the `TermGetInfo` action. To determine the words to which a stem expands, use the `TermExpand` action.

LOVE	LOVELY	LOVING
LOVED	LOVES	LOVINGLY
LOVELIES	LOVEST	LOVINGS

Example 1 (Conceptual)

`action=Query&Text=Lovely`

- Default matching

- `AdvancedSearch=True`

- `AdvancedCaseSearch=True`

Content matches this query conceptually, in the default configuration or when you set `AdvancedSearch` or `AdvancedCaseSearch` to **True**. The query word is neither in quotation marks nor prefixed with a tilde (~), so Content finds documents that contain words that have the same stem as *lovely* (ignoring their case).

Example matching words	Example nonmatching words
lovely	lover
love	lovelorn
loved	loveless
loving	

Example 2 (Conceptual or Exact)

`action=Query&Text="Lovely"`

- Default matching

  In the Content default configuration, Content ignores the quotation marks and matches the word conceptually. It finds documents that contain words that have the same stem as *lovely* (ignoring their case).

Example matching words	Example nonmatching words
lovely	lover
love	lovelorn
loved	loveless
loving	

- `AdvancedSearch=True` or `AdvancedCaseSearch=True`

  If you set `AdvancedSearch` or `AdvancedCaseSearch` to **True**, Content matches the exact form of the term, because the query word is in quotation marks. It finds only documents that contain the word *lovely*. Because the word is not prefixed with a tilde (~), Content ignores its case.

Example matching words	Example nonmatching words
lovely	love
	loved
	loving
	lover

Example 3 (Conceptual)

`action=Query&Text=~Lovely`

- Default matching

  In the default configuration, Content ignores the tilde (~) and matches the word conceptually. It finds documents that contain words that have the same stem as *lovely* (ignoring their case).

Example matching words	Example nonmatching words
lovely	lover
love	lovelorn
loved	loveless
loving	

- `AdvancedSearch=True`

  If you set `AdvancedSearch` to **True**, Content ignores the tilde (~) and matches the word conceptually, because the query word is not in quotation marks. It finds documents that contain words that have the same stem as *lovely* (ignoring their case).

Example matching words	Example nonmatching words
lovely	lover
love	lovelorn
loved	loveless
loving	

- `AdvancedCaseSearch=True`

  If you set `AdvancedCaseSearch` to **True**, Content matches conceptually and case-sensitively, because the query word is prefixed with a tilde (~) but not in quotation marks. It finds documents that contain words that have the same stem as *Lovely*.

Example matching words	Example nonmatching words
Lovely	lovely
Love	love
Loved	loved
Loving	loving

Example 4 (Conceptual or Exact)

`action=Query&Text="~Lovely"`

- Default matching

  In the default configuration, Content ignores the tilde (~) and the quotation marks, and matches the word conceptually. It finds documents that contain words that have the same stem as *lovely* (ignoring their case).

Example matching words	Example nonmatching words
lovely	lover
love	lovelorn
loved	loveless
loving	

- `AdvancedSearch=True`

  If you set `AdvancedSearch` to **True**, Content ignores the tilde (~) and matches the exact form of the term. It finds only documents that contain the word *lovely* (ignoring its case).

Example matching words	Example nonmatching words
lovely	love
Lovely	loved
	loving

- `AdvancedCaseSearch=True`

  If you set `AdvancedCaseSearch` to **True**, Content matches it exactly and case-sensitively because the query word is prefixed with a tilde (~) and in quotation marks. It finds only documents that contain the word *Lovely*.

Example matching words	Example nonmatching words
Lovely	lovely
	love
	loved
	loving

Phrase Search

The IDOL Content component provides these phrase searching options:

- **Phrase occurrence search**. Use a phrase occurrence search to find documents that contain a certain number of occurrences of a phrase.

- **Default phrase search**. Put your search string in quotation marks to treat the string as a phrase and return only documents in which a matching phrase occurs (a phrase in a document qualifies as a match if it stems the same way as the query phrase).

- **Exact phrase search**. Enable `AdvancedSearch` before you index content into the IDOL Content component to find exact matches for terms or phrases. In this case, if you query Content with a term or a phrase in quotation marks, it matches them in their exact (unstemmed) form.

- **Case-sensitive exact phrase search**. Enable `AdvancedCaseSearch` before you index content into the IDOL Content component to find case-sensitive exact matches for terms or phrases. You can then query Content with a term or a phrase in quotation marks, to match them in their exact (unstemmed) form, or prefix a term with a tilde (~) to find only case-sensitive matches. For more information about case-sensitive searching, refer to *IDOL Expert*.

Phrase Occurrence Search

You can restrict the documents that match a query phrase to documents in which the number of occurrences of the phrase is within a specified range. Specify the range with a colon-separated pair of

numbers. You can use a maximum value of 32000. For example:

```
action=Query&Text="Gene therapy"[5:8]
```

This query returns only documents in which the phrase *Gene therapy* appears between five and eight times (inclusive). This also applies for Wildcard terms.

You can specify open-ended ranges. For example:

```
action=Query&Text="Gene therapy"[10:]
```

This query returns only documents in which the phrase *Gene therapy* appears 10 or more times.

Default Phrase Search

If you query Content with a phrase in quotation marks, by default it returns only documents in which a matching phrase occurs (a phrase in a document qualifies as a match if it stems the same way as the query phrase).

For example:

```
action=Query&Text="fresh and lovely"
```

Content removes any stop words that the query contains (the example query above contains the stop word *and*) and applies stemming. It is as if the query were this:

```
action=Query&Text="fresh love"
```

When it matches the query, Content returns only documents that contain a phrase that stems the same way as the phrase in the query string. The query **"fresh and lovely"** can return only documents that contain a phrase that stems to *fresh love* (for example, *fresh love*, *freshest loving*, *fresh and lovely*, *freshly loved*, and so on).

Exact Phrase Search

If you enable AdvancedSearch before you index content into the IDOL Content component, and submit a phrase in quotation marks, Content matches the phrase in its exact (unstemmed) form (this setting also switches Content to an advanced weighting algorithm that improves conceptual querying).

For example:

```
action=Query&Text="fresh and lovely"
```

Content removes any stop words that the query contains (the example query above contains the stop word *and*) but matches only documents that contain the exact (unstemmed) form of the terms. It is as if the query were:

```
action=Query&Text="fresh lovely"
```

When it matches the query, Content returns only documents that contain a phrase that matches the phrase in the query string. The query **"fresh and lovely"** returns only documents that contain a phrase that matches the phrase *fresh lovely* (for example, *fresh lovely*, *fresh and lovely*, *fresh or lovely*, and so on).

> **NOTE:** During the exact phrase search process, IDOL Server stems the terms, matches

> documents that contain the stem, and then finds occurrences that match your exact unstemmed form. This stemming step means that an exact phrase search might not match a document in a different language that contains the exact unstemmed term, because the stemming rules are different. To match these documents, you must use `GenericStemming`.

Case-Sensitive Exact Phrase Search

If you enable `AdvancedCaseSearch` before you index content into the IDOL Content component, and submit a phrase in quotation marks, Content matches the exact unstemmed phrase (the same way it matches if `AdvancedSearch` is enabled). If you prefix a term with a tilde (~), Content matches the term case-sensitively.

For example:

```
action=Query&Text="fresh and ~Lovely"
```

Content removes any stop words that the query contains (the example query above contains the stop word *and*) but matches only documents that contain the exact unstemmed form of the terms. It is as if the query were:

```
action=Query&Text="fresh ~Lovely"
```

When it matches the query, Content returns only documents that contain a phrase that matches the phrase in the query string. The query **"fresh and ~Lovely"** can return only documents that contain a phrase which matches the phrase *fresh Lovely* (for example, *fresh Lovely*, *fresh and Lovely*, *Fresh or Lovely*, and so on).

For more information about case-sensitive searching, refer to *IDOL Expert*.

Phrase Search Examples

The word *fresh* stems to *fresh*, and the word *lovely* stems to *love*. The following words also stem to *fresh* and *love*.

> **TIP:** To determine the stem of a word, use the `TermGetInfo` action. To determine the words to which a stem expands, use the `TermExpand` action.

FRESH	FRESHEST	FRESHLY
FRESHES	FRESHING	FRESHNESS
LOVE	LOVELY	LOVING
LOVED	LOVES	LOVINGLY
LOVELIES	LOVEST	LOVINGS

Example 1 (Conceptual)

```
action=Query&Text=fresh and Lovely
```

- Default matching

- `AdvancedSearch=True`

- `AdvancedCaseSearch=True`

The IDOL Content component matches this query conceptually in all configurations, because the phrase is not in quotation marks and none of its words are prefixed with a tilde (~).

Content first finds documents that contain both words that have the same stem as *fresh* and words that have the same stem as *lovely* (ignoring their case), and then documents that contain either words that have the same stem as *fresh* or words that have the same stem as *lovely* (ignoring their case).

Example matching words	Example nonmatching words	Example matching words	Example nonmatching words
fresh	fresher	lovely	lover
freshness	freshman	love	lovelorn
freshly	freshwater	loved	loveless
freshest		loving	

Example 2 (Conceptual or Exact)

`action=Query&Text="fresh and Lovely"`

- Default matching

    In the default configuration, the quotation marks indicate that this is a phrase search. Content removes any stop words in the phrase, and applies stemming. It finds only documents in which a word whose stem matches the stem of *fresh* occurs immediately before a word that matches the stem of *lovely* (ignoring their case).

Example matching phrases	Example nonmatching phrases
fresh lovely	freshman lover
freshest love	fresher lovelorn
freshly loved	lovely and fresh
fresh and lovingly	

- `AdvancedSearch=True` or `AdvancedCaseSearch=True`

    If you set `AdvancedSearch` or `AdvancedCaseSearch` to **True**, Content removes any stop words that the phrase contains, and then matches the exact terms, because the phrase is in quotation marks. Because none of the words are prefixed with a tilde (~), it finds only documents that contain the phrase *fresh lovely* (ignoring its case).

Example matching phrases	Example nonmatching phrases
fresh lovely	freshest love
fresh and lovely	freshly loved
fresh or lovely	fresh and lovingly
	lovely and fresh

Example 3 (Conceptual or Exact)

```
action=Query&Text="fresh and ~Lovely"
```

- Default matching

  In the default configuration, the quotation marks indicate that this is a phrase search. Content ignores the tilde (~). It removes any stop words that the phrase contains, and applies stemming. It finds only documents in which a word whose stem matches the stem of *fresh* occurs immediately before a word that matches the stem of *lovely* (ignoring their case).

Example matching phrases	Example nonmatching phrases
fresh lovely	freshman lover
freshest love	fresher lovelorn
freshly loved	lovely and fresh
fresh and lovingly	

- `AdvancedSearch=True`

  If you set `AdvancedSearch` to **True**, Content removes any stop words that the phrase contains and then matches the exact terms, because the phrase is in quotation marks. It ignores the tilde (~). Content finds only documents that contain the phrase *fresh lovely* (ignoring its case).

Example matching phrases	Example nonmatching phrases
fresh lovely	freshest love
fresh and lovely	freshly loved
fresh or lovely	fresh and lovingly
	lovely and fresh

- `AdvancedCaseSearch=True`

  If you set `AdvancedCaseSearch` to **True**, Content removes any stop words that the phrase contains and then matches the exact terms, because the phrase is in quotation marks. Because

Lovely is prefixed with a tilde (~), it is matched case-sensitively. Content finds only documents that contain the phrase *fresh Lovely*.

Example matching phrases	Example nonmatching phrases
fresh Lovely	fresh lovely
fresh and Lovely	fresh and lovely
Fresh and Lovely	fresh or lovely
Fresh or Lovely	Lovely fresh

Boolean and Proximity Search

You can use the `Query` action to submit standard Boolean searches to the IDOL Content component, and to submit proximity searches, which allow you to give words that appear close together in the search string a higher weighting.

NOTE: You must specify all operators in capital letters.

Boolean Search Operators

The following operators allow you to manipulate a query by applying them to words, exact phrases, or other Boolean expressions. The IDOL Content component uses APCM (Adaptive Probabilistic Concept Modeling) to rank the results that match the Boolean query.

Boolean search operators

Operator	Explanation
AND	Binary operator. Ensures that every document that returns contains both terms. For example: `action=Query&Text=cat+AND+dog` This query returns only documents that contain both *cat* and *dog*.
NOT	Unary operator. Ensures that the term following NOT is excluded from all the returned documents. For example: `action=Query&Text=cat+NOT+dog` This query returns only documents that contain *cat* and not *dog*. **NOTE:** NOT applies only to the term that immediately follows it. To exclude multiple terms, place them in brackets. To exclude a phrase, put the phrase in quotation marks and in brackets. For example:

Boolean search operators, continued

Operator	Explanation

	Doc 1: *I went to the city for the New Year*
	
	Doc 2: *I went to New York City for the New Year*
	
	The following query does not match either of these documents:
	
	`action=Query&Text=city NOT (New York)`
	
	The following query matches the first document but not the second:
	
	`action=Query&Text=city NOT ("New York")`

OR	Binary operator. One or both terms must appear for the document to return. This is the default behavior if no explicit operator is given between two terms. For example: `action=Query&Text=cat+OR+dog` This query returns only documents that contain either *cat*, *dog*, or both terms.
EOR or XOR	Binary operator. Logical exclusive OR. Only one of the terms is permitted to appear for the document to return. This operator is rarely used. For example: `action=Query&Text=cat+XOR+dog` This query returns only documents that contain either the term *cat* or the term *dog*. Documents that contain both *cat* and *dog* do not return.
( )	Bracketed expressions. These expressions are evaluated left to right and can be nested. They dictate the precedence and behavior of combined operator statements. For example: `action=Query&Text=(fish EOR pie) AND (chips EOR mash)` This query returns only documents that contain one of these combinations: *fish* and *chips* *fish* and *mash* *pie* and *chips* *pie* and *mash*

Proximity Search Operators

You can apply proximity operators to words, exact phrases, or Boolean expressions to perform a proximity search. Note the following details:

- If the two specified words are adjacent to each other, their proximity is 1. If one word separates them, their distance is 2, and so on.

- Proximity operators do not count stop words. For example, because *and* is a stop word, the terms

cat and *dog* have the proximity 1 in the text:

- ○ *catdog*

- ○ *cat and dog*.

- The IDOL Content component uses APCM (Adaptive Probabilistic Concept Modeling) to rank results.

- Proximity operators work recursively so that nested Boolean queries can have proximity operators apply to brackets or phrases. For example, in the expression

```
(term1) NEAR10 ((term2) DNEAR2 (term3))
```

the `NEAR10` operator ensures that *term1* is in proximity to an occurrence of *term2* within two of *term3*.

Proximity search operators

Operator	Explanation
NEAR*N*	Returns only documents in which the second term is within *N* words of the first term—that is, the terms are *N* or fewer words apart. If you do not specify *N*, `NEAR` defaults to 5. For example: `action=Query&Text=red+NEAR1+green` This query returns only documents in which the term *red* is adjacent to the term *green*. For example, documents that contain *red green* or *green red* return. Documents that contain *red orange green* do not return (because the terms are not close enough to each other).
DNEAR*N*	Directed `NEAR`. Returns only documents in which the second term is within *N* words of the first term, in the specified order. If you do not specify *N*, `DNEAR` defaults to 5. For example: `action=Query&Text=red+DNEAR2+green` This query returns only documents in which the term *green* follows the term *red*, and is no more than `two` words away from the term *red*. For example, documents that contain *red orange green* return, but documents that contain *green orange red* or *red orange blue green* do not return.
WNEAR*N*	Weighted `NEAR` (with OR operation). This proximity operator returns documents that contain either of the two terms. It promotes relevance when the terms are *N* or fewer words apart (closer together implies higher relevance). If you do not specify *N*, `WNEAR` defaults to 5. For example: `action=Query&Text=dog+WNEAR7+cat` This query returns documents that contain either *dog* or *cat*. It gives extra relevance to documents in which *dog* and *cat* appear seven or fewer words apart in a piece of text. This weight increases as the terms get closer to each other. Documents in which the terms occur more than seven words apart, or in which only one term occurs, return with normal relevance.

Proximity search operators, continued

Operator	Explanation
YNEAR*N*	Weighted NEAR (with AND operation). This proximity operator returns documents that contain both of the terms. It promotes relevance when the terms are *N* or fewer words apart (closer together implies higher relevance). If you do not specify *N*, YNEAR defaults to 5. For example: `action=Query&Text=dog+YNEAR7+cat` This query returns documents that contain both *dog* and *cat*. It gives extra relevance to documents in which *dog* and *cat* appear seven or fewer words apart in a piece of text. This weight increases as the terms get closer to each other. Documents in which the terms occur more than seven words apart return with the normal relevance.
BEFORE	Returns only documents in which the first term precedes the second one. For example: `action=Query&Text=red+BEFORE+green` This query returns only documents in which the term *green* appears later than the term *red*. You can also use BEFORE for FieldText queries. For a FieldText query to successfully compare two occurrences of the same field, you must set the XMLFullStructure configuration parameter to **True** in the [Server] section of the IDOL Content component configuration file.
AFTER	Returns only documents in which the first term appears later than the second one. For example: `action=Query&Text=red+AFTER+green` This query returns only documents in which the term *red* appears later than the term *green*. You can also use AFTER for FieldText queries. For a FieldText query to successfully compare two occurrences of the same field, you must set the XMLFullStructure configuration parameter to **True** in the [Server] section of the IDOL Content component configuration file.
XNEAR	Returns only documents in which the second term is exactly *N* words from the first term. For example: `action=Query&Text=cats+XNEAR2+dogs` This query returns only documents in which the term *dogs* follows the term *cats* and is exactly two words away from the term *cats*. This means that documents which contain *cats and dogs* return, but documents that contain *dogs and cats* or *cats, dogs* do not return.
SENTENCE	Returns only documents in which the second term is in the same sentence as the first term.

Proximity search operators, continued

Operator	Explanation
	For example:
	`action=Query&Text=cats+SENTENCE+dogs`
	This query returns only documents in which the term *dogs* appears in the same sentence as the word *cats*.
	The IDOL Content component breaks the document into sentences by using a number of criteria. The most important criteria is the detection of an end of sentence marker, which includes a period (.), question mark (?), or exclamation point (!), as well as their multibyte variants. However, the presence of one of these characters is not always sufficient to mark the end of a sentence, because these characters are often used in abbreviations, names, and other items for purposes other than the end of a sentence. To locate a more accurate sentence boundary, Content also uses characteristics such as capitalization and syntactic observations.
SENTENCE *NN*	Returns only documents in which the second term is in the same sentence as the first term, and they are within *NN* words of each other. For example:
	`action=Query&Text=cats+SENTENCE10+dogs`
	This query returns only documents in which the term *dogs* occurs in the same sentence as, and within 10 words of, the word *cats*.
	NOTE: `SENTENCE0` has the same behavior as `SENTENCE`.
DSENTENCE	Returns only documents in which the second term occurs later than the first term, in the same sentence. For example:
	`action=Query&Text=cats+DSENTENCE+dogs`
	This query returns only documents in which the term *dogs* occurs later than the word *cats*, in the same sentence.
DSENTENCE *NN*	Returns only documents in which the second term occurs later than the first term, and within *NN* words in the same sentence. For example:
	`action=Query&Text=cats+DSENTENCE10+dogs`
	This query returns only documents in which the term *dogs* occurs later in the same sentence than, and within 10 words of, the word *cats*.
	NOTE: `DSENTENCE0` has the same behavior as `DSENTENCE`.
PARAGRAPH	Returns only documents in which the second term is in the same paragraph as the first term.
	For example:
	`action=Query&Text=red+PARAGRAPH+green`

Proximity search operators, continued

Operator	Explanation
	This query returns only documents in which the term *green* appears in the same paragraph as the word *red*. The words do not have to be in the same sentence in the paragraph.
PARAGRAPH *NN*	Returns only documents in which the second term is in the same paragraph as the first term, and they are within *NN* words of each other. For example: `action=Query&Text=cats+PARAGRAPH20+dogs` This query returns only documents in which the term *dogs* occurs in the same paragraph as, and within 20 words of, the word *cats*. **NOTE:** PARAGRAPH0 has the same behavior as PARAGRAPH.

WHEN and NOTWHEN Search Operators

If you set XMLFullStructure to **True** in the configuration file, you can use the WHEN and NOTWHEN search operators to return documents in which XML fields occur with a common parent field. When you set XMLFullStructure to **True**, The IDOL Content component gives each occurrence of the same XML field name a different field code, so that it can identify each one uniquely.

To return only XML documents in which fields or attributes that have a specific value occur together, apply the WHEN operator to words or phrases. You can use the WHEN operator to return only those XML documents in which:

- two fields with the same parent field contain specified terms or phrases. See Example 1, below.

- two attributes that occur in the same field contain specified terms or phrases. See Example 2, on the next page.

- a field contains a specified term or phrase, and has an attribute with a specific value. See Example 3, on page 238.

To return only XML documents in which one field or attribute that has a specific value does not occur in the same parent field as another specified field or attribute and value, use the NOTWHEN operator. You can use the NOTWHEN operator in the same way as the WHEN operator. See Example 4, on page 238.

Example 1

You can use WHEN to return only XML documents in which two fields with the same parent field contain specified terms or phrases.

For example:

`action=Query&Text=audi:make+WHEN+red:color`

This query returns only XML documents in which the make and color fields are children of the same parent field, and contain the values *audi* and *red* respectively.

For example, this query returns the following document:

```
<XML>
    <car>
        <make>audi</make>
        <color>red</color>
    </car>
    <car>
>        <make>mercedes</make>
        <color>silver</color>
    </car>
    </DOC>
</XML>
```

It does not return the following document:

```
<XML>
    <DOC>
        <car>
            <make>audi</make>
            <color>silver</color>
        </car>
        <car>
            <make>mercedes</make>
            <color>red</color>
        </car>
    </DOC>
</XML>
```

You can also use complex nested expressions with the WHEN operator in query text. For example:

```
action=Query&Text=(London:CITY WHEN English:LANG) WHEN ("United Kingdom":COUNTRY)
```

Example 2

You can use WHEN to return only XML documents in which two attributes that occur in the same field contain specified terms or phrases. For example:

```
action=Query&Text=English:_ATTR_LANG WHEN "Cape Town":_ATTR_CAPITAL
```

This query returns only XML documents in which the LANG and CAPITAL attributes occur in the same field, and contain the values *English* (in the LANG attribute) and *Cape Town* (in the CAPITAL attribute).

For example, this query returns the following document:

```
<XML>
    <DOC>
        <COUNTRY CAPITAL="Cape Town" LANG="English" POP="44">South Africa</COUNTRY>
        <COUNTRY CAPITAL="Berlin" LANG="German" POP="80">Germany</COUNTRY>
    </DOC>
</XML>
```

It does not return the following document:

```
<XML>
    <DOC>
        <COUNTRY CAPITAL="Cape Town" LANG="Afrikaans" POP="10">South Africa</COUNTRY>
        <COUNTRY CAPITAL="London" LANG="English" POP="80">England</COUNTRY>
    </DOC>
</XML>
```

Example 3

You can also use WHEN to return only XML documents in which a field contains a specified term or phrase, and has an attribute that has a specific value. For example:

```
action=Query&Text=Fr.html:_ATTR_HREF WHEN France:A
```

This query returns only XML documents in which an A field contains the value *France*, and has an HREF attributes with the value *Fr.html*.

For example, this query returns the following document:

```
<XML>
    <DOC>
        <A HREF="Fr.html">France</A>
    </DOC>
</XML>
```

Example 4

You can use NOTWHEN to return XML documents that contain one field with a specified term and phrase, but that do not contain another field and value in the same parent field.

```
action=Query&Text=audi:make+NOTWHEN+red:color
```

This query returns only XML documents in which the make field contains the value audi, under a parent field that does not also have a color field with the value red.

For example, this query returns the following document:

```
<XML>
    <DOC>
        <car>
            <make>audi</make>
            <color>silver</color>
        </car>
        <car>
            <make>mercedes</make>
            <color>red</color>
        </car>
    </DOC>
</XML>
```

It does not return the following document:

```
<XML>
    <DOC>
        <car>
            <make>audi</make>
            <color>red</color>
        </car>
        <car>
            <make>mercedes</make>
            <color>silver</color>
        </car>
    </DOC>
</XML>
```

Specify the Number of Levels from the XML Root

You can add a numeric value to the WHEN operator to indicate the number of hierarchical levels from the XML root from which to match the terms or phrases.

For example, if you have the following XML document:

```
<?xml version="1.0"?>
<XML>
    <DOCUMENT>
        <DREREFERENCE>Reference_1</DREREFERENCE>
        <car>
            <make>ford</make>
            <color>
                <exterior>blue</exterior>
                <interior>black</interior>
            </color>
        </car>
    </DOCUMENT>
    <DOCUMENT>
        <DREREFERENCE>Reference_2</DREREFERENCE>
        <car>
            <make>ferrari</make>
            <color>
                <exterior>blue</exterior>
                <interior>brown</interior>
            </color>
        </car>
        <car>
            <make>ford</make>
            <color>
                <exterior>yellow</exterior>
                <interior>blue</interior>
            </color>
        </car>
```

```
    </DOCUMENT>
</XML>
```

The following query returns both documents, because the `exterior` and `make` fields share the same parent field (`<DOCUMENT>`), which is two levels from the root, and contain *blue* and *ford* respectively.

```
action=Query&Text=blue:EXTERIOR+WHEN2+ford:MAKE
```

The following query returns only `Reference_1`, because the `exterior` and `make` fields share the same parent field (`<car>`), which is three levels from the root, and contain *blue* and *ford* respectively.

```
action=Query&Text=blue:EXTERIOR+WHEN3+ford:MAKE
```

The following query does not return either document, because the `exterior` and `make` fields do not share the same parent field four levels from the root.

```
action=Query&Text=blue:EXTERIOR+WHEN4+ford:MAKE
```

Precedence of Search Operators

Boolean and proximity operators have the following precedence:

```
Highest:      NOT
                NEAR; DNEAR; XNEAR; YNEAR
                AND; BEFORE; AFTER; WHEN; SENTENCE; DSENTENCE; PARAGRAPH
Lowest:       OR; WNEAR; EOR
```

Operators that have the same level of precedence have neither left or right associativity. You can use brackets to bind terms together as appropriate. Proximity operators must have terms on either side, and cannot be adjacent to brackets.

Simple Field Restricted Search

You can use simple field restrictions within a `Query` action `Text` parameter to return results that contain specific values in specific fields. You can also combine query text with a field restriction to increase the relevance of results that contain specific values in specific fields. For these field restrictions:

- you can use only fields that the IDOL Content component stores as `Index` fields.

- you can use Wildcards.

- you cannot match more than one value, or a value that contains spaces or punctuation.

- each individual field restriction must contain fewer than 512 characters.

Related Topics

- Set up the Field Index Process, on page 43

Example Queries

```
action=Query&Text=cat:DRETITLE
```

This query returns only documents that contain the value *cat* in their `DRETITLE` field.

```
action=Query&Text=cat dog:DRETITLE
```

This query returns documents that contain the term *cat* in any field and the term *dog* in their `DRETITLE` field. Documents that contain either *cat* (in any field) or *dog* in their `DRETITLE` field also return, but with a lower relevance.

```
action=Query&Text=cat:CREATURE:FAUNA dog:ANIMAL
```

This query returns only documents that contain the value *cat* in their `CREATURE` or `FAUNA` field and the value *dog* in their `ANIMAL` field. Documents that contain either *cat* in their `CREATURE` or `FAUNA` field or *dog* in their `ANIMAL` field also return, but with a lower relevance.

```
action=Query&Text=engin*:Title
```

This query returns only documents whose title field contains the specified string (for example, *engineer*, *engineering* and so on). Note that Content expands Wildcards before it stems terms.

FieldText Search

FieldText queries are `Query`, `Suggest`, or `SuggestOnText` actions that include the `FieldText` parameter. You can combine this parameter with the `Text` parameter to restrict the results that your query returns to documents that contain a specific value in a specified field. You can also use the `FieldText` parameter on its own to query for documents that contain specific values in specific fields. Micro Focus does not recommend this type of query because it slows the IDOL Content component processing speed.

To specify how documents must match fields and field values to return as results, the `FieldText` parameter uses field specifiers which identify the pattern of values in fields. These field specifiers fall into three groups, described in these sections:

- Field Specifiers for Common Restrictions, on page 243
- Field Specifiers for Advanced Restrictions, on page 251
- Field Specifiers to Bias Result Scores, on page 278

> **NOTE:** FieldText queries that include commas and braces within the query have specific percent-encoding requirements. For information about percent-encoding, see FieldText, on page 310.

FieldText Query Guidelines

- In addition to document fields, you can also apply FieldText restrictions to metafields, such as `autn_date`, `autn_database`, `autn_section`, `autn_langtype`, and `autn_language`. For example:

  ```
  FieldText=STRING{Archiv}:autn_database
  ```

In this example, the `autn_database` metafield must contain the substring *Archiv* for this document to return. For example, if the `autn_database` metafield has the value *Archive* or *Archives*, this document returns.

```
FieldText=WILD{eng*}:autn_langtype
```

In this example, the `autn_langtype` metafield must contain a value that starts with *eng* (for example, *englishASCII* or *English_UTF8*) for this document to return as a result.

- You can use multiple field specifiers simultaneously by combining them with the Boolean operators `AND`, `NOT`, and `OR`. For example:

```
FieldText=GREATER{6.95}:PRICE:PREIS+AND+MATCH{Brown}:AUTHOR:AUTOR
```

In this example, the `PRICE` or `PREIS` field must contain a number greater than 6.95, and the `AUTHOR` or `AUTOR` field must have the value *Brown*, for the document to return as a result.

```
FieldText=(LESS{10}:PRICE+AND+MATCH{Penguin}:PUBLISHER)+OR+(NRANGE
{20,30}:PRICE+AND+MATCH{George Orwell}:AUTHOR)
```

In this example, the documents must meet one of these two criteria:

  ○ the `PRICE` field contains a number smaller than 10, and its `PUBLISHER` field has the value *Penguin*.

  ○ the `PRICE` field contains a number between 20 and 30, and its `AUTHOR` field has the value *George Orwell*.

- You can use the proximity operators `BEFORE` and `AFTER` to specify the order in which certain fields must occur in a document. For example:

```
FieldText=FieldText=MATCH{Penguin}:PUBLISHER BEFORE MATCH{George Orwell}:AUTHOR
```

This example returns only documents where the `PUBLISHER` field contains the value *Penguin* earlier in the document than the `AUTHOR` field that contains the value *George Orwell*.

> **NOTE:** For a `FieldText` query to successfully compare two occurrences of the same field, you must set the `XMLFullStructure` configuration parameter to **True** in the `[Server]` section of the configuration file.

```
FieldText=MATCH{George Orwell}:AUTHOR AFTER MATCH{Thomas Brown}:AUTHOR
```

This example returns only documents where one occurrence of the `AUTHOR` field contains the value *George Orwell* later in the document than another occurrence of the `AUTHOR` field that contains the value *Thomas Brown*. This query returns results only if you have set `XMLFullStructure` to **True**.

- To identify the fields, use the format `/FieldName` to match root-level fields, `*/FieldName` to match all fields except root-level, or `/Path/FieldName` to match fields that the specified path points to.

  To identify XML attributes, use the format `<tag name>/_ATTR_<attribute name>`, for example, `FARM/_ATTR_ANIMAL`. You can also use Wildcards when identifying fields, for example, `/Fi*d1` or `/Field*`.

- All string matching is case insensitive, unless you set the `CaseSensitive` parameter to **True** (this parameter does not apply for `TERM`, `TERMPHRASE`, and `TERMALL`, which are always case

insensitive).

Strings can contain punctuation, except braces ({ }). You must percent-encode ampersands (&) in strings as `%26`. To distinguish commas within strings from separator commas, percent-encode commas twice within strings (`%252C`).

You must not percent-encode commas that separate multiple items, and braces that start and end lists.

For example, to match the two strings `hello,world` and `goodbye,again`:

```
FieldText=MATCH{hello%252Cworld,goodbye%252Cagain}:FIELD
```

Related Topics

- Metadata Fields, on page 105

Field Specifiers for Common Restrictions

Specifier	Finds documents in which a specified field contains...
`MATCH`	a value that exactly matches one or more specified strings
`EQUAL`, `GREATER`, `LESS`, `NOTEQUAL`, `NRANGE`	a number
`GTNOW`, `LTNOW`, or `RANGE`	a date
`WILD`	a value that matches a specified Wildcard string

Fields whose Value Exactly Matches One or More Strings

You can use the following field specifier to return documents with fields that contain a specified string.

MATCH

The `MATCH` field specifier (case sensitive) finds documents in which a specified field contains a value that exactly matches a specified string.

```
FieldText=MATCH{yourStrings}:yourFields
```

where:

yourStrings	is one or more strings. A document returns only if one of these strings is matched by one of *yourFields* exactly.
	The matching is case insensitive.
	FieldText queries which include commas and braces within the query have specific percent-encoding requirements. For information about percent-encoding, see FieldText, on page 310.

yourFields	is one or more fields. A document returns only if it contains one of these fields, and if the value in this field exactly matches one of *yourStrings*. To specify multiple fields, separate them with colons (there must be no space before or after a colon).

Examples:

`FieldText=MATCH{Archive,Web,docs}:DB:DATABASE`

The `DB` or `DATABASE` field must have the value *Archive*, *Web,* or *docs* for the document to return as a result.

`FieldText=MATCH{Premier league}:DB`

The `DB` field must have the value *Premier League* for the document to return as a result.

`FieldText=MATCH{0-226-10389-7}:ISBN`

The `ISBN` field must have the value *0-226-10389-7* for the document to return as a result.

Fields that Contain a Number

You can use the following field specifiers (case sensitive) to return documents with fields that contain numbers. To optimize the processing time of queries for fields that contain numbers, store them as numeric fields in the IDOL Content component during the indexing process.

Related Topics

- NumericType Fields, on page 96

EQUAL

The `EQUAL` field specifier (case sensitive) allows you to find documents in which a specified field contains a number that matches one of the numbers specified by you.

`FieldText=EQUAL{`*yourNumbers*`}:`*yourFields*

where:

yourNumbers	is one or more numbers. A document returns only if one of *yourFields* contains one of these numbers.
yourFields	is one or more fields. A document returns only if it contains one of these fields, and if this field contains one of *yourNumbers*. If you want to specify multiple fields, separate them with colons (there must be no space before or after a colon).

Examples:

`FieldText=EQUAL{1234567890123}:ACCOUNT:KONTO`

The `ACCOUNT` or `KONTO` field must contain the number *1234567890123* for the document to return.

`FieldText=EQUAL{3.9,4.9,7}:ID`

The `ID` field must contain the number *3.9*, *3.90*, *4.9*, *4.90*, *7*, or *7.0* for the document to return.

GREATER

The `GREATER` field specifier (case sensitive) allows you to find documents in which a specified field contains a number that is greater than a number you specify.

`FieldText=GREATER{`*yourNumber*`}:`*yourFields*

where:

yourNumber	is a number. A document returns only if one of *yourFields* contains a number that is greater than this number. By default, the range is exclusive. You can add an equals sign (=) to include the number that you specify.
yourFields	is one or more fields. A document returns only if it contains one of these fields, and if the number in this field is greater than *yourNumber*. To specify multiple fields, separate them with colons (there must be no space before or after a colon).

Examples:

`FieldText=GREATER{66}:ID`

The `ID` field must contain a number greater than *66* for the document to return.

`FieldText=GREATER{5.59}:PRICE:PREIS`

The `PRICE` or `PREIS` field must contain a number greater than *5.59* for the document to return.

`FieldText=GREATER{=6}:QUANTITY`

The `QUANTITY` field must contain a value of *6* or greater for the document to return.

LESS

The `LESS` field specifier (case sensitive) allows you to find documents in which a specified field contains a number that is smaller than a number that you specify.

`FieldText=LESS{`*yourNumber*`}:`*yourFields*

where:

yourNumber	is a number. A document returns only if one of *yourFields* contains a number that is smaller than this number. By default, the range is exclusive. You can add an equals sign (=) to include the number that you specify.
yourFields	is one or more fields. A document returns only if it contains one of these fields, and if the number in this field is smaller than *yourNumber*. If you want to specify multiple fields, separate them with colons (there must be no space before or after a colon).

Examples:

`FieldText=LESS{66}:ID`

The `ID` field must contain a smaller number than *66* for the document to return.

`FieldText=LESS{5.59}:PRICE:PREIS`

The `PRICE` or `PREIS` field must contain a smaller number than *5.59* for the document to return.

`FieldText=LESS{=6}:QUANTITY`

The `QUANTITY` field must contain a value of *6* or lower for the document to return.

NOTEQUAL

The `NOTEQUAL` field specifier (case sensitive) allows you to find documents in which a specified field contains a number that does not match a number that you specify.

`FieldText=NOTEQUAL{`*yourNumber*`}:`*yourFields*

where:

yourNumber	is a number. A document returns only if one of *yourFields* does not contain this number.
yourFields	is one or more fields. A document returns only if it contains one of these fields, and if this field does not contain *yourNumber*. To specify multiple fields, separate them with colons (there must be no space before or after a colon).

Examples:

`FieldText=NOTEQUAL{1234567890123}:ACCOUNT:KONTO`

The `ACCOUNT` or `KONTO` field must not contain the number *1234567890123* for the document to return.

`FieldText=NOTEQUAL{3.9}:ID`

The `ID` field must not contain the number *3.9* for this document to return.

NRANGE

The `NRANGE` field specifier (case sensitive) allows you to find documents in which a specified field contains a number that falls in the inclusive range of two numbers that you specify.

`FieldText=NRANGE{`*yourNumbers*`}:`*yourFields*

where:

yourNumbers	is two numbers, separated by a comma (there must be no space before or after the comma). A document returns only if one of *yourFields* contains a number that falls within the inclusive range of the specified numbers (including decimal numbers). By default, the range is inclusive. You can use the greater than (<) and less than (>) symbols to specify an exclusive range. You can make either or both ends of the

	range exclusive.
yourFields	is one or more fields. A document returns only if it contains one of these fields, and if this field contains a number that falls within the inclusive range of *yourNumbers*. To specify multiple fields, separate them with colons (there must be no space before or after a colon).

Examples:

`FieldText=NRANGE{1,99}:CODE`

The `CODE` field must contain a number between *1* and *99* (inclusive) for the document to return.

`FieldText=NRANGE{1234567890123,2345678901234}:ACCOUNT:KONTO`

The `ACCOUNT` or `KONTO` field must contain a number between *1234567890123* and *2345678901234* (inclusive) for the document to return.

`FieldText=NRANGE{36.5,42.3}:CODE`

The `CODE` field must contain a number between *36.5* and *42.3* (inclusive) for the document to return.

`FieldText=NRANGE{>1,5}:CODE`

The `CODE` field must contain a value that is greater than *1*, up to and including *5* for the document to return.

Fields that Contain a Date

You can use the following field specifiers (case sensitive) to return documents with fields that contain dates.

> **NOTE:** To optimize the processing time of queries for fields that contain dates, store them as numeric date fields in the IDOL Content component during the indexing process.

Related Topics

- NumericDateType Fields, on page 95

GTNOW

The `GTNOW` field specifier (case sensitive) allows you to find documents in which a specified field contains a date that is greater than the `AUTNDATE` (that is, all documents that were indexed with dates after the current time).

`FieldText=GTNOW{}:yourFields`

where:

yourFields	is one or more fields. A document returns only if it contains one of these fields, and if this field contains a date that is greater than the `AUTNDATE` (that is, all documents that were indexed with dates after the current time). To specify multiple fields, separate them with colons (there must be no space before or after a colon).

Examples:

`FieldText=GTNOW{}:TIME`

The `TIME` field must contain a date that is greater than the `AUTNDATE` (that is, all documents that were indexed with dates after the current time) for the document to return.

`FieldText=GTNOW{}:TIME:DATE`

The `TIME` or `DATE` field must contain a date that is greater than the current time (that is, all documents that were indexed with dates after the current time) for the document to return.

LTNOW

The `LTNOW` field specifier (case sensitive) allows you to find documents in which a specified field contains a date that is smaller than the `AUTNDATE` (that is, all documents that were indexed with dates before the current time).

`FieldText=LTNOW{}:yourFields`

where:

yourFields	is one or more fields. A document returns only if it contains one of these fields, and if this field contains a date that is smaller than the current time.
	To specify multiple fields, separate them with colons (there must be no space before or after a colon).

Examples:

`FieldText=LTNOW{}:*/TIME`

The `TIME` field must contain a date that is smaller than the `AUTNDATE` (that is, all documents that were indexed with dates before the current time) for the document to return.

`FieldText=LTNOW{}:TIME:DATE`

The `TIME` or `DATE` field must contain a date that is smaller than the `AUTNDATE` (that is, all documents that were indexed with dates before the current time) for the document to return.

RANGE

The `RANGE` field specifier (case sensitive) allows you to find documents in which a specified field contains a date that falls within the inclusive range of two dates that you specify.

`FieldText=RANGE{yourDates}:yourFields`

where:

yourDates	is two dates separated by a comma (there must be no space before or after the comma). A document returns only if one of *yourFields* contains a date that falls within the inclusive time span of the specified dates. You can use the formats in the table below to specify each date.
	By default, the range is inclusive. You can use the greater than (<) and less than (>) symbols to specify an exclusive range. You can make either or both ends of the range

	exclusive.
yourFields	is one or more fields. A document returns only if it contains one of these fields, and if this field contains a date that falls within the inclusive range of *yourDates*. To specify multiple fields, separate them with colons (there must be no space before or after a colon).

Date formats

Format	Explanation
D+/M+/#YY+	A date. For example, `1/3/05`, `23/12/1999`, `10/07/40`, or `8/5/2012`. If the year is a number less than 40, it is read as a year in the 2000s. If the year is a number between 40 and 99, it is read as a year in the 1900s. For example, `01/02/01` is read as **1 February 2001**, and `01/03/40` is read as **1 March 1940**.
HH:NN:SS D+/M+/#YY+	A time and date. For example, `10:30:45 1/3/05`, `18:55:00 23/12/99`, `01:23:45 10/07/1940`, or `07:15:00 8/5/2012`. If the year is a number less than 40, it is read as a year in the 2000s. If the year is a number between 40 and 99, it is read as a year in the 1900s. For example, `01/02/01` is read as **1 February 2001**, and `01/03/40` is read as **1 March 1940**.
HH:NN:SS D+/M+/#YY+ #ADBC	A time and date with a time period. For example, `10:30:45 1/3/05 AD`, `18:55:00 23/12/99 CE`. For the time period, you can use `AD`, `CE`, `BC`, `BCE` or any predefined list of EPOCH indicators.
N	A positive or negative number of days from the current date. For example, `-1` specifies yesterday's date, `0` specifies today's date, `1` specifies tomorrow's date, `2` specifies two days from now (the current date plus two), and so on.
*N*s	A positive or negative number of seconds from now. For example, `-60s` specifies one minute ago, `-900s` specifies 15 minutes ago, `-3600s` specifies one hour ago and so on. `60s` specifies one minute from now, `900s` specifies 15 minutes from now, `3600s` specifies one hour from now, and so on.
*N*e	Epoch seconds (seconds since 1 January 1970). For example, `1012345000e` specifies 22:56:40 on 29 January 2002. **NOTE:** Content uses the local time zone to match epoch values. For example, if your local time zone is GMT-6, `1012345000e` is 03:56:40 on 29 January 2002. If you use this value as the beginning of a range, Content retrieves documents after 03:56:40 on 29 January 2002.

For `RANGE`, you can specify an open-ended range by using a period (`.`):

- If you type a period for the first point in time, the beginning of the time period is not restricted (the period ranges up to the specified date, including any earlier date).

- If you type a period for the second point in time, the end of the time period is not restricted (the period ranges from the specified date, including any later date).

Examples:

`FieldText=RANGE{01/01/90,1/1/01}:DATE`

The `DATE` field must contain a date between 01/01/1990 and 1/1/2001 for the document to return.

`FieldText=RANGE{01/01/02,01/01/2003}:DATE:DATUM`

The `DATE` or `DATUM` field must contain a date between 01/01/2002 and 01/01/2003 for the document to return.

`FieldText=RANGE{-14,-7}:DATE`

The `DATE` field must contain a date 14 to 7 days before the current date for the document to return.

`FieldText=RANGE{0,1}:DATE`

The `DATE` field must contain today's or tomorrow's date (which is possible, for example, if the document originates from a different time zone or if the field contains an expiration date) for the document to return.

`FieldText=RANGE{01/01/99,.}:DATE:FECHA`

The `DATE` or `FECHA` field can contain any date after 01/01/1999 for the document to return.

`FieldText=RANGE{.,10/10/04}:DATE`

The `DATE` field can contain any date before 10/10/2004 for the document to return.

`FieldText=RANGE{-172800s,-1}:DATE`

The `DATE` field must contain a time between 48 and 24 hours ago.

`FieldText=RANGE{198765e,.}:DATE`

The `DATE` field must contain a date between 198765 seconds after the epoch and the current time.

`FieldText=RANGE{>05/06/2013,05/06/2014}:DATE`

The `DATE` field must contain a date after 05/06/2013, up to and including 05/06/2014.

Fields whose Value Matches Wildcard Strings

WILD

The `WILD` field specifier (case sensitive) allows you to find documents in which a specified field contains a string that matches a specified Wildcard string.

If the query does not contain any Wildcard characters (? or *), the `WILD` field specifier acts in the same way as the `MATCH` field specifier.

`FieldText=WILD{yourStrings}:yourFields`

where:

yourStrings	is one or more strings that contain Wildcards. A document returns only if one of *yourFields* matches one of these strings.
	FieldText queries which include commas and braces within the query have specific percent-encoding requirements. For information about percent-encoding, see FieldText, on page 310.
yourFields	is one or more fields. A document returns only if it contains one of these fields, and if this field contains one of *yourStrings*.
	If you want to specify multiple fields, separate them with colons (there must be no space before or after a colon).

Examples:

`FieldText=WILD{*.html,*.htm}:URL`

The `URL` field value must end with *.html* or *.htm* for this document to return as a result.

`FieldText=WILD{passi*incarnata}:Climbers:Plants`

The `Climbers` or `Plants` field value must contain a phrase that begins with *passi* and ends with *incarnata* (for example, *passionflower incarnata* or *passiflora incarnata*) for this document to return as a result.

`FieldText=WILD{*www.example.com*.txt}:PATH`

The `PATH` field value must contain a path that contains *www.example.com* and ends with *.txt* (for example, *http://www.example.com/files/doc.txt*) for the document to return as a result.

`FieldText=WILD{wom?n }:Clothes`

The `Clothes` field value must contain a word that matches the specified Wildcard string (for example, *woman* or *women*) for this document to return as a result.

Field Specifiers for Advanced Restrictions

Field specifiers

Specifier	Use to find documents in which a specified field...
ARANGE	contains a value that falls within a specific alphabetical range
BITAND, BITANDHEX, BITANDOFFHEX	contains a value that results in a nonzero value when a bitwise AND operation is carried out against it
BITSET	contains a value in a `BitFieldType` field where the specified bit is set
BOOLEANFIELD	contains a Boolean agent
DISTCARTESIAN	contains Cartesian (x/y) coordinates values within a

Field specifiers, continued

Specifier	Use to find documents in which a specified field...
	specified distance from a specified point
DISTSPHERICAL	contains latitude and longitude values within a specified distance from a specified point
EMPTY	does not exist or does not contain a value
EXISTS	exists, irrespective of its value
FUZZY	contains a value that is similar to a specified string
MATCHALL or EQUALALL	contains multiple instances, whose values include at least one match for each of the specified strings or numeric values
MATCHCOVER or EQUALCOVER	contains multiple instances, all of whose values are matched in the specified strings or numeric values
MATCHRECURSE	contains a specified reference in a ReferenceMemoryMappedType field recursively to a maximum number of times
NOTMATCH, NOTSTRING, NOTWILD	contains multiple instances, at least one of whose values does not match the specified string
POLYGON	contains Cartesian (x/y) coordinate values within a specified polygonal shape
GEOCONTAINS	contains a geospatial region that wholly contains a specified point or polygonal shape.
GEOINTERSECTS	contains a point or geospatial region that intersects a specified point or polygonal shape
GEOWITHIN	contains a point or geospatial region that is wholly within the specified polygonal shape.
STRING, STRINGALL, SUBSTRING	contains a specified string
TERM, TERMALL, TERMEXACT, TERMEXACTALL, TERMEXACTPHRASE, TERMPHRASE	whose value match specific terms or phrases

Fields whose Value Falls within a Specific Alphabetical Range

ARANGE

The ARANGE field specifier (case sensitive) allows you to find documents in which a specified field contains a term that falls within the inclusive alphabetical range of two terms that you specify.

```
FieldText=ARANGE{yourTerms}:yourFields
```

where:

yourTerms	is two terms separated by a comma (there must be no space before or after the comma). A document returns only if one of *yourFields* contains a term that falls within the inclusive alphabetical range of the specified terms.
	You can use a period (.) in place of one of the terms to represent an unrestricted value.
	- If you use the period in place of the first term, it includes all values up to the second term. - If you use the period in place of the second term, it includes all values after the first term.
	It uses unicode tables to determine alphabetical order. This means that non-7-bit ASCII characters (α, ä, å, ç, ð, ê, ë, ø, ö, ü, û, ß, ÿ, and so on) come after z in the alphabet.
yourFields	is one or more fields. A document returns only if it contains one of these fields, and if this field contains a term that falls within the inclusive alphabetical range of *yourTerms*.
	To specify multiple fields, separate them with colons (there must be no space before or after a colon).

Examples:

```
FieldText=ARANGE{aardvark,alligator}:ANIMAL
```

The ANIMAL field must contain a value that alphabetically falls between *aardvark* and *alligator*. If the ANIMAL field contains the value *aardvark*, *ant*, *anteater*, *antelope*, or *alligator*, the document returns. If the ANIMAL field contains the value *armadillo*, it does not return.

```
FieldText=ARANGE{bear,buffalo}:ANIMAL:TIER
```

The ANIMAL or TIER field must contain a value that alphabetically falls between *bear* and *buffalo*. If the field contains the value *bear*, *bee*, *Biene*, *bird*, or *buffalo*, the document returns. If the ANIMAL field contains the value *Büffel* or *chipmunk*, it does not return.

```
FieldText=ARANGE{dog,.}:ANIMAL AND ARANGE{.,cat}:PET
```

The ANIMAL field must contain a value that alphabetically falls after *dog*, and the PET field must contain a value that alphabetically falls before *cat*.

Fields with a Nonzero Value for Bitwise AND

You can use the following field specifiers (case sensitive) to return documents with fields whose value result in a nonzero value when a bitwise AND operation is carried out against a specified value.

BITAND

The BITAND field specifier (case sensitive) allows you to find documents with a field whose integer value does not result in 0 when a bitwise AND operation is carried out between this value and an integer

value that you specify.

`FieldText=BITAND{`*`yourInteger`*`}:`*`yourBitFields`*

where:

yourInteger	is an integer. A document returns only if one of *yourBitFields* contains a value that results in a nonzero value when a bitwise AND operation is carried out between this value and the specified integer.
yourBitFields	is one or more fields. A document returns only if it contains one of these fields, and if this field contains an integer that results in a nonzero value when a bitwise AND operation is carried out between it and *yourInteger*. If you want to specify multiple fields, separate them with colons (there must be no space before or after a colon).

Example:

`FieldText=BITAND{128}:BitField`

The binary representation of the integer value 128 is compared with the binary representations of the integer values that `BitField` fields in the IDOL Content component contain. Only documents whose `BitField` values result in a nonzero value when they are compared to the binary representation of 128 return.

For example, if the `BitField` of a document contains the integer value `129`, it returns, but a document whose `BitField` contains the value `127` does not return.

The following tables show the field value comparison.

Integer	Binary	
128	1000 0000	
129	1000 0001	
	1000 0000	this evaluates to `True`

Integer	Binary	
128	1000 0000	
127	0111 1111	
	0000 0000	this evaluates to `False`

BITANDHEX

The `BITANDHEX` field specifier (case sensitive) allows you to find documents with a field whose hexadecimal string value does not result in zero when a bitwise AND operation is carried out between this value and a hexadecimal string that you specify.

`FieldText=BITANDHEX{`*`yourHexString`*`}:`*`yourBitFields`*

where:

yourHexString	is a hexadecimal string. A document returns only if one of *yourBitFields* contains a value that results in a nonzero value when a bitwise AND operation is carried out between this value and the specified hexadecimal string.
yourBitFields	is one or more fields. A document returns only if it contains one of these fields, and if this field contains a hexadecimal string that results in a nonzero value when a bitwise AND operation is carried out between it and *yourHexString*. To specify multiple fields, separate them with colons (there must be no space before or after a colon).

Example:

`FieldText=BITANDHEX{7F}:BitField`

The binary representation of the hexadecimal value 7F is compared with the binary representations of the hexadecimal values that `BitField` fields in IDOL Server contain. Only documents whose `BitField` values result in a nonzero value when they are compared to the binary representation of 7F return.

For example, if the `BitField` of a document contains the hexadecimal value `C0`, it returns, but a document whose `BitField` contains the hexadecimal value 80 does not return.

The following tables show the field value comparison.

Hex	Binary	
7F	0111 1111	
C0	1100 0000	
	0100 0000	this evaluates to **True**

Hex	Binary	
7F	0111 1111	
80	1000 0000	
	0000 0000	this evaluates to **False**

BITANDOFFHEX

The `BITANDOFFHEX` field specifier (case sensitive) allows you to find documents with a field whose hexadecimal string value does not result in zero when a bitwise AND operation is carried out between this value and an offset hexadecimal string you specify.

`FieldText=BITANDOFFHEX{NN,yourHexString}:yourBitFields`

where:

NN	is the number of 16-bit chunks by which the value in *yourHexString* and in *yourBitFields* is shifted before the bitwise AND operation is carried out (this

	allows you to store sparse bit masks more efficiently).
yourHexString	is a hexadecimal string. A document returns only if one of *yourBitFields* contains a value that results in a nonzero value when a bitwise AND operation is carried out between this value and the specified hexadecimal string.
yourBitFields	is one or more fields. A document returns only if it contains one of these fields, and if this field contains a hexadecimal string that results in a nonzero value when a bitwise AND operation is carried out between it and *yourHexString*.
	If you want to specify multiple fields, separate them with colons (there must be no space before or after a colon).

Example:

```
FieldText=BITANDOFFHEX{01,0a001}:BitOffField
```

The binary representation of the hexadecimal value `01,0a001` is compared with the binary representations of the hexadecimal values that `BitOffField` fields in the IDOL Content component contain. Only documents whose `BitOffField` values result in a nonzero value when they are compared to the binary representation of `01,0a001` (after they have been shifted left by one 16-bit chunk) return.

For example, if the `BitOffField` contains the value `1,bc01`, the document returns, but a document whose `BitOffField` contains the value `0,5ffeffff` does not return.

The following tables show the field value comparison.

nn,hexstring	Hex	Binary	
01,0a001	A0010000	1010 0000 0000 0001 0000 0000 0000 0000	
1,bc01	BC010000	1011 1100 0000 0001 0000 0000 0000 0000	
		1010 0000 0000 0001 0000 0000 0000 0000	this evaluates to **True**

nn,hexstring	Hex	Binary	
01,0a001	A0010000	1010 0000 0000 0001 0000 0000 0000 0000	
0,5ffeffff	5FFEFFFF	0101 1111 1111 1110 1111 1111 1111 1111	
		0000 0000 0000 0000 0000 0000 0000 0000	this evaluates to **False**

Fields that Contain BitFieldType Information

BITSET

The BITSET field specifier (case sensitive) allows you to find documents with a BitFieldType field that contains a value where the specified bit is set. This allows you to search for documents that belong to a particular set of documents.

> **NOTE:** You can use the `BITSET` field specifier only for `BitFieldType` fields.

`FieldText=BITSET{`*yourBitSets*`}:`*yourFields*

where:

yourBitSets	are one or more set numbers, separated by commas. These set numbers must be in normal decimal form.
yourFields	are one or more `BitFieldType` fields. A document returns only if it contains one of these fields, and if this field contains a value where the bit corresponding to *yourBitSets* is set.
	To specify multiple fields, you must separate them with colons (there must be no space before or after a colon).

Examples:

`FieldText=BITSET{0,4}:BitField`

This example matches any document where the binary representation of the hexadecimal value contained in `BitField` has the bit set (the binary digit is **1**) for set 0 or set 4.

For example, if a document has a `BitField` that contains the hexadecimal value **A4**, it does not match the query. The binary representation is `10100100`, where the bits for set 0 and 4 are both **0**, so the document is not part of these sets.

If the document has a `BitField` that contains the hexadecimal value **A8**, it does match the query. The binary representation is `10101000`, where the bit for set 4 is a **1**, so the document is part of set 4.

`FieldText=BITSET{2}:BitField AND BITSET{15}:BitField`

The binary representation of the hexadecimal value contained in `BitField` must have the bits switched on for both sets 2 and 15.

For example, if a document has a `BitField` that contains the hexadecimal value **B110**, it does not match the query. The binary representation is `1011000100010000`, where the bit is set to **1** for set 15, but not for set 2.

If the document has a `BitField` that contains the hexadecimal value **B114** it matches the query. The binary representation is `1011000100010100`, where the bit is set to **1** for both set 15 and set 2.

Fields whose Values are Boolean Agents

BOOLEANFIELD

The `BOOLEANFIELD` field specifier (case sensitive) allows you to find documents in which a specified Boolean agent field contains an expression that matches text you specify. A Boolean agent is a Boolean or proximity expression that legacy technologies use to categorize documents.

> **NOTE:** If you are using a `Query` action, Micro Focus recommends that you use the `AgentBooleanField` action parameter rather than the `BOOLEANFIELD` field specifier. However, if you want to match more than one Boolean agent field, you must use the `BOOLEANFIELD` field specifier.

```
FieldText=BOOLEANFIELD{yourText}:yourFields
```

where:

yourText	is text. A document returns only if one of *yourFields* contains a Boolean or proximity expression that matches the specified text.
yourFields	is one or more Boolean agent fields. A document returns only if it contains one of these fields, and if this field contains a Boolean or proximity expression that matches *yourText*. If you want to specify multiple fields, separate them with colons (there must be no space before or after a colon).

For example:

```
BOOLEANFIELD{The cat sat on the mat}:MyFirstBooleanField:MySecondBooleanField
```

Any document that has a `MyFirstBooleanField` or `MySecondBooleanField` field that contains a Boolean or proximity expression that matches the specified text returns. For example, the Boolean and proximity expressions `cat AND mat`, `cat OR mat`, `cat BEFORE mat`, and `cat DNEAR1 sat` could match *The cat sat on the mat*. Therefore, documents that contain any of these Boolean and proximity expressions are returned.

Documents whose `MyFirstBooleanField` or `MySecondBooleanField` fields contain, for example, `cat AND mat AND dog` or `mat BEFORE cat` are not returned.

Fields that are within a Specified Distance from a Specified Point

DISTCARTESIAN

The `DISTCARTESIAN` field specifier allows you to find documents that contain fields that define a point (X and Y coordinates) or region that is within a specified distance from a specified point.

You can specify the document position fields either as a pair of fields (corresponding to X and Y coordinate fields), or a single field that uses POINT or POLYGON definitions (in Well-known text format) to specify position information (for example, a unified `GeospatialType` field).

> **TIP:** If your document location fields contain regions (in Well-known text POLYGON format), the `DISTCARTESIAN` operator calculates distances from the geometric center of the region.

```
FieldText=DISTCARTESIAN{coordX,coordY,dist}:POSITION
```

where:

coordX	is the specified X coordinate.
coordY	is the specified Y coordinate.
dist	is the distance in kilometers from the specified coordinates.
POSITION	The document field or fields that contain the position value. You can use one of the

following options to specify location fields:

- a single field. This field must contain a Well-known text format POINT or POLYGON definition (for example, `POINT(x y)`).

- two fields, in the format *X:Y*, where *X* is the field that contains the x coordinate, and *Y* is the field that contains the y coordinate. You must specify the fields in the order `x:y`.

You can specify multiple options for the location fields, in form *POSITION1:POSITION2:POSITION3*, and so on. This form can include a mix of types (unified location fields and split latitude and longitude fields).

> **NOTE:** If you use multiple position fields or field pairs, IDOL Server cannot match documents where a pair of split geospatial fields occurs interleaved with other geospatial fields. For example, if you use `X1:Y1:POSITION` in your query, and these fields occur in a document in the order `X1`, `POSITION`, `Y1`, IDOL Server does not match that document.
>
> In such cases, you can rewrite the query, in the form `DISTCARTESIAN{...}:X1:Y1 OR DISTCARTESIAN{...}:POSITION`

Example:

```
FieldText=DISTCARTESIAN{10,11,5}:X:Y
```

This example matches all documents whose (X,Y) position is within a distance of 5 units of the point (10,11). The position of a document in this example is contained in the fields `X` and `Y`.

```
FieldText=DISTCARTESIAN{10,11,5}:POSITION
```

This example matches all documents whose (X,Y) position is within a distance of 5 units of the point (10,11). The position of a document in this example is contained in the `POSITION` field.

DISTSPHERICAL

The `DISTSPHERICAL` field specifier allows you to find documents that contain location fields that define a point (latitude and longitude) or region that is within a specified distance from a specified point.

You can specify the document location fields either as a pair of fields (corresponding to latitude and longitude fields), or a single field that uses POINT or POLYGON definitions (in Well-known text format) to specify location information (for example, a unified `GeospatialType` field).

> **TIP:** If your document location fields contain regions (in Well-known text POLYGON format), the `DISTSPHERICAL` operator calculates distances from the geometric center of the region.

```
FieldText=DISTSPHERICAL{lat,long,dist}:LOCATION
```

where:

lat	is the latitude. Specify latitude positions south of the equator as negative.
long	is the longitude. Specify longitude positions west of the Greenwich Meridian as negative.

dist	is the distance in kilometers from the specified latitude and longitude.
LOCATION	The document field or fields that contain the location value. You can use one of the following options to specify location fields: • a single field. This field must contain a Well-known text format POINT definition (that is, `POINT (x y)`). • two fields, in the format *LATFIELD*:*LONGFIELD*, where *LATFIELD* is the field that contains the latitude value, and *LONGFIELD* is the field that contains the longitude value. You must specify the fields in the order `latitude:longitude`. You can specify multiple options for the location fields, in form *LOCATION1*:*LOCATION2*:*LOCATION3*, and so on. This form can include a mix of types (unified location fields and split latitude and longitude fields). **NOTE:** If you use multiple position fields or field pairs, IDOL Server cannot match documents where a pair of split geospatial fields occurs interleaved with other geospatial fields. For example, if you use `LAT1:LONG1:LOCATION` in your query, and these fields occur in a document in the order `LAT1`, `LOCATION`, `LONG1`, IDOL Server does not match that document. In such cases, you can rewrite the query, in the form `DISTSPHERICAL {...}:LAT1:LONG1 OR DISTCARTESIAN{...}:LOCATION`

Example:

```
FieldText=DISTSPHERICAL{37.75,-122.4,20}:LAT:LONG
```

This example matches all documents whose position is within a 20 kilometer radius of San Francisco (37.75N,122.4W). The latitude and longitude position of a document in this example is contained in the fields `LAT` and `LONG`, respectively.

```
FieldText=DISTSPHERICAL{52.2,0.1,20}:LOCATION
```

This example matches all documents whose position is within a 20 kilometer radius of Cambridge (52.2N,0.1W). The latitude and longitude position of a document in this example is contained in the `LOCATION` field.

Fields that Contain Coordinates within a Specified Area

POLYGON

The `POLYGON` field specifier (case sensitive) allows you to find documents that contain a location that wholly fits within a specified polygonal shape.

You can specify the document position fields either as a pair of fields (corresponding to X and Y coordinate fields), or a single field that uses POINT or POLYGON definitions to specify position information (for example, a unified `GeospatialType` field). If you use unified fields and the document contains a POLYGON definition, the document matches only if the polygon fits wholly within the polygon you specify in the field operator.

```
FieldText=POLYGON{coordX,coordY,coordX,coordY,...}:POSITION
```

where:

coordX,coordY	are the coordinates for one of the vertices. Specify an *x,y* pair of coordinates for each vertex of the polygon, working either clockwise or counterclockwise around the polygon. You can specify a concave polygon, but the edges must not cross themselves. You can specify coordinates with decimal numbers.
POSITION	The document field or fields that contain the position value. You can use one of the following options to specify location fields: • a single field. This field must contain unified geospatial position information (that is, Well-known text format POINT or POLYGON definitions). • two fields, in the format *X:Y*, where *X* is the field that contains the x coordinate, and *Y* is the field that contains the y coordinate. You must specify the fields in the order x:y. You can specify more than one pair of fields in the form X1:Y1:X2:Y2 and so on. You can specify multiple options for the location fields, in form :*POSITION1*:*POSITION2*:*POSITION3*, and so on. This form can include a mix of types (unified location fields and split latitude and longitude fields). NOTE: If you use multiple position fields or field pairs, IDOL Server cannot match documents where a pair of split geospatial fields occurs interleaved with other geospatial fields. For example, if you use X1:Y1:POSITION in your query, and these fields occur in a document in the order X1, POSITION, Y1, IDOL Server does not match that document. In such cases, you can rewrite the query, in the form POLYGON{...}:X1:Y1 OR POLYGON{...}:POSITION

For example:

`FieldText=POLYGON{1,1,-1,1,0,-2,1,-1}:XPOS:YPOS`

This example matches all documents whose (X,Y) position is within the quadrilateral with vertices at (1,1), (−1,1), (0,−2), (1,−1).

Fields that Contain a Geospatial Region or Point

GEOINTERSECTS

The GEOINTERSECTS field specifier (case sensitive) allows you to find documents that have a document field that describes a location that wholly or partially fits within a specified point or polygonal shape.

When you specify a polygon, GEOINTERSECTS matches points that occur within the polygon, and polygon regions that at least partially overlap it. When you specify a point, GEOINTERSECTS matches polygons that contain that point, and points that exactly match the point.

`FieldText=GEOINTERSECTS{POINT(coordX coordY)}:POSITION`

or

```
FieldText=GEOINTERSECTS{POLYGON((coordX coordY, coordX coordY,...))}:POSITION
```

coordX coordY	The coordinates for a point or one of the vertices of a polygon. For a polygon, specify an X/Y pair of coordinates for each vertex, working either clockwise or counterclockwise around the polygon. The polygon can be concave, but the edges cannot cross themselves. The polygon can also contain holes. You can specify coordinates with decimal numbers.
POSITION	The document field containing the position information. This field must contain unified geospatial location information (that is, Well-known text format POINT or POLYGON definitions). You can specify multiple options for the location fields, in form *POSITION1*:*POSITION2*:*POSITION3*, and so on.

For example:

```
FieldText=GEOINTERSECTS{POLYGON((1 1, -1 1, 0 -2, 1 -1))}:LOCATION
```

This example matches all documents whose (X,Y) position is within the quadrilateral with vertices at (1,1), (-1,1), (0,-2), (1,-1). A document returns if it has a LOCATION field that contains either a POINT definition that occurs within the specified quadrilateral, or a Well-known text POLYGON definition that at least partially overlaps it.

GEOCONTAINS

The GEOCONTAINS field specifier (case sensitive) allows you to find documents that contain a document field that specifies a location region that contains a specified point or polygonal shape.

```
FieldText=GEOCONTAINS{POINT(coordX coordY)}:POSITION
```

or

```
FieldText=GEOCONTAINS{POLYGON((coordX coordY, coordX coordY,...))}:POSITION
```

coordX coordY	The coordinates for a point or one of the vertices of a polygon. For a polygon, specify an X/Y pair of coordinates for each vertex, working either clockwise or counterclockwise around the polygon. The polygon can be concave, but the edges cannot cross themselves. The polygon can also contain holes. You can specify coordinates with decimal numbers.
POSITION	The document field containing the position information. This field must contain unified geospatial location information (that is, Well-known text format POINT or POLYGON definitions). You can specify multiple options for the location fields, in form *POSITION1*:*POSITION2*:*POSITION3*, and so on.

For example:

```
FieldText=GEOCONTAINS{POINT(0 0)}:LOCATION
```

This example matches all documents that define a region that contains the point (0,0). A document returns if it has a `LOCATION` field that contains either a `POINT` definition that matches the query point (that is, (0,0)), or a `POLYGON` definition that contains it.

GEOWITHIN

The `GEOWITHIN` field specifier (case sensitive) allows you to find documents that have a document field that describes a location region that wholly fits within a specified point or polygonal shape.

When you specify a polygon, `GEOWITHIN` matches points and polygons that occur wholly within the polygon. When you specify a point, `GEOWITHIN` matches only points that exactly match that point.

`FieldText=GEOWITHIN{POINT(coordX coordY)}:POSITION`

or

`FieldText=GEOWITHIN{POLYGON((coordX coordY, coordX coordY,...))}:POSITION`

coordX *coordY*	The coordinates for a point or one of the vertices of a polygon. For a polygon, specify an X/Y pair of coordinates for each vertex, working either clockwise or counterclockwise around the polygon. The polygon can be concave, but the edges cannot cross themselves. The polygon can also contain holes. You can specify coordinates with decimal numbers.
POSITION	The document field containing the position information. This field must contain unified geospatial location information (that is, Well-known text format POINT or POLYGON definitions). You can specify multiple options for the location fields, in form *POSITION1:POSITION2:POSITION3*, and so on.

For example:

`FieldText=GEOWITHIN{POLYGON((1 1, -1 1, 0 -2, 1 -1))}:LOCATION`

This example matches all documents whose (X,Y) position is within the quadrilateral with vertices at (1,1), (-1,1), (0,-2), (1,-1). A document returns if it has a `LOCATION` field that contains either a POINT or POLYOGN definition that occurs wholly within the specified quadrilateral.

Fields that Do Not Exist or Contain No Value

EMPTY

The `EMPTY` field specifier (case sensitive) allows you to find documents in which a specified field does not exist or contains no value.

`FieldText=EMPTY{}:yourFields`

where:

yourFields	is one or more fields. A document returns only if it does not contain any of these fields or if these fields are empty. To specify multiple fields, separate them with colons (there must be no space before or after a colon).

Examples:

`FieldText=EMPTY{}:ID`

A document must not contain an `ID` field, or must hold no value in its `ID` field, to return.

`FieldText=EMPTY{}:ID:Name`

A document must not contain an `ID` or `Name` field, or must hold no value in its `ID` or `Name` field, to return.

Specific Fields, Irrespective of their Value

EXISTS

The `EXISTS` field specifier (case sensitive) allows you to find documents that contain a specified field even if this field contains no value.

`FieldText=EXISTS{}:`*`yourFields`*

where:

yourFields	is one or more fields. A document returns only if it contains one of these fields (even if the field is empty).
	To specify multiple fields, separate them with colons (there must be no space before or after a colon).

Examples:

`FieldText=EXISTS{}:ID`

A document must contain an `ID` field to return.

`FieldText=EXISTS{}:ID:NAME`

A document must contain an `ID` or `NAME` field (or both) to return.

Fields whose Values are Similar to a Specified String

FUZZY

The `FUZZY` field specifier (case sensitive) allows you to find documents in which a specified field contains a term that is similar to a specified term or phrase.

`FieldText=FUZZY{`*`yourTerms`*`}:`*`yourFields`*

where:

yourTerms	is one or more terms (or phrases). A document returns only if one of these terms (or phrases) is similar to a string in one of *yourFields*.
	FieldText queries which include commas and braces within the query have specific percent-encoding requirements. For information about percent-encoding, see FieldText, on page 310.

yourFields	is one or more fields. A document returns only if it contains one of these fields, and if the value in this field is similar to one of *yourTerms*.
	If you want to specify multiple fields, separate them with colons (there must be no space before or after a colon).

Example:

```
FieldText=FUZZY{Bisiness News,Arkive}:DRETITLE
```

The `DRETITLE` field value must be similar to the term *Bisiness News*, or *Arkive* for the document to return. For example, a document whose `DRETITLE` field contains *Business News* returns, but a document whose `DRETITLE` field contains *Document Arkive* does not.

At Least One Field Instance Matches a Specified String or Number

You can use the following field specifiers (case sensitive) to return documents with multiple instances of the same fields and at least one field instance contains a specified string or number.

MATCHALL

The `MATCHALL` field specifier (case sensitive) allows you to find documents in which a specified field occurs in multiple instances, and in which there is at least one match among those instances for each of a set of strings that you specify.

> **NOTE:** You can optimize the field specifier speed by restricting the field to the `MatchType` property type.

```
FieldText=MATCHALL{yourStrings}:yourField
```

where:

yourStrings	is one or more strings. A document returns only if all these strings have exact matches among the instances of *yourField*. Separate the strings with commas (there must be no space before or after a comma).
	FieldText queries which include commas and braces within the query have specific percent-encoding requirements. For information about percent-encoding, see FieldText, on page 310.
yourField	is the name of the field to match against. A document returns only if it contains the field and only if all *yourStrings* are matched at least once in various instances of the field.

Examples:

```
FieldText=MATCHALL{Archive,Web,docs}:DIRECTORY
```

The `DIRECTORY` fields must include at least the values *Archive* and *Web* and *docs* for the document to return as a result.

```
FieldText=MATCHALL{Smith,Garcia,Lee}:SURNAME
```

The values *Smith*, *Garcia*, and *Lee* must all have matches in the SURNAME fields for the document to return as a result.

```
FieldText=MATCHALL{Smith%5C, John, Garcia%5C, Joaquin}:FULLNAME
```

The values *Smith*, *John* and *Garcia*, *Joaquin* must both have matches in the FULLNAME fields for the document to return as a result.

EQUALALL

The EQUALALL field specifier (case sensitive) allows you to find documents in which a specified field occurs in multiple instances, and in which there is at least one value among those instances that is equal to each of a set of numeric values that you specify.

> **NOTE:** You can optimize the field specifier speed by restricting the field to the NumericType property type.

```
FieldText=EQUALALL{yourValues}:yourNumericField
```

where:

yourValues	is one or more numeric values. A document returns only if all these values occur among the instances of *yourNumericField*. Separate the numbers with commas (there must be no space before or after a comma).
yourNumericField	is the name of the field to match against. A document returns only if it contains the field and only if all *yourValues* are matched at least once in various instances of the field.

Example:

```
FieldText=EQUALALL{32,98.6,212}:FAHRENHEIT
```

The FAHRENHEIT fields must include at least the values *32, 98.6*, and *212* for the document to return as a result.

```
FieldText=EQUALALL{1999,2000,2001}:YEAR
```

The values *1999*, *2000*, and *2001* must all appear in YEAR fields for the document to return as a result.

All Field Instances Match a Specified String or Number

You can use the following field specifiers (case sensitive) to return documents with multiple instances of the same fields and all field instances contain a specified string or number.

MATCHCOVER

The MATCHCOVER field specifier (case sensitive) allows you to find documents in which the values in all instances of a specified field have matches in the set of values provided in the specifier. In other words, the specifier must *cover* all instances of the field. A search that uses MATCHCOVER is slower than one that uses MATCH.

> **NOTE:** You can optimize the field specifier speed by restricting the field to the MatchType and

> `CountType` property types. You must specify both property types.

`FieldText=MATCHCOVER{`*`yourStrings`*`}:`*`yourField`*

where:

yourStrings	is one or more strings. A document returns only if the value in each of its instances of *yourField* matches one of the strings in *yourStrings*. Separate the strings with commas (there must be no space before or after a comma). FieldText queries which include commas and braces within the query have specific percent-encoding requirements. For information about percent-encoding, see FieldText, on page 310.
yourField	is the name of the field to match against. A document returns only if: • it contains one or more instances of the field and the value of each instance is found in *yourStrings*. • it does not contain the field at all.

Example:

`FieldText=MATCHCOVER{Confidential,Secret,TopSecret,FBI}:SECURITYLEVEL`

For a document to return as a result, its `SECURITYLEVEL` fields must not contain any values that are not in the specified list. For example, if a document includes a `SECURITYLEVEL` field with the value *MI5*, it does not return. (If a document has no `SECURITYLEVEL` field at all, it returns.)

EQUALCOVER

The `EQUALCOVER` field specifier (case sensitive) allows you to find documents in which the values in all instances of a specified field are found in the set of values provided in the specifier. In other words, the specifier must *cover* all instances of the field.

> **NOTE:** You can optimize the field specifier speed by restricting the field to the `NumericType` and `CountType` property types. You must specify both property types.

`FieldText=EQUALCOVER{`*`yourValues`*`}:`*`yourField`*

where:

yourValues	is one or more numeric values. A document returns only if the value in each of its instances of *yourField* equals one of the values in *yourValues*. Separate the numbers with commas (there must be no space before or after a comma). FieldText queries which include commas and braces within the query have specific percent-encoding requirements. For information about percent-encoding, see FieldText, on page 310.
yourField	is the name of the field to match against. A document returns only if: • it contains one or more instances of the field and the value of each instance equals a value in *yourValues*.

	• it does not contain the field at all.

Example:

```
FieldText=EQUALCOVER{9,10,11,12}:GRADELEVEL
```

For a document to return as a result, its GRADELEVEL fields must have no values that are not in the specified list. For example, if a document includes a GRADELEVEL field with the value *8*, it does not return. (If a document has no GRADELEVEL field, it returns.)

Fields that Contain a Specified ReferenceMemoryMappedType Field

MATCHRECURSE

The MATCHRECURSE field specifier matches documents that contain a specified reference in a ReferenceMemoryMappedType field recursively to a maximum number of times. You must restrict this field specifier to a single ReferenceMemoryMappedType field. It has the following syntax:

```
action=Query&FieldText=MATCHRECURSE{Ref,RecurseNumber}:yourField
```

where:

Ref	is the initial reference.
RecurseNumber	is the maximum number of times to recursively return references by using the value of the ReferenceMemoryMappedType field.
yourField	is the name of the ReferenceMemoryMappedType field.

For example, if you define PARENT as a ReferenceMemoryMapped field, the query

```
action=Query&FieldText=MATCHRECURSE{MyRef,1}:PARENT
```

matches the document with the reference *MyRef* (parent) and documents whose PARENT field contains *MyRef* (children). The query

```
action=Query&FieldText=MATCHRECURSE{MyRef,2}:PARENT
```

matches the document with the reference *MyRef* (parent), documents whose PARENT field contains *MyRef* (children), and documents whose PARENT field contains the references in the returned child documents (grandchildren).

Fields that Do Not Contain a Specified Value

NOTMATCH

The NOTMATCH field specifier (case sensitive) allows you to find documents in which at least one instance of the specified fields contains a value that does not match the specified string.

If there are one or more instances of a particular field in the document, the document returns as long as at least one instance does not contain any of the specified strings, even if another instance of the field

does match. The document does not return if all instances of the specified fields contain an exact match of one of the specified strings.

```
FieldText=NOTMATCH{yourStrings}:yourFields
```

where:

yourStrings	is one or more strings. A document returns only if at least one instance of one of *yourFields* contains a value that is not an exact match for these strings. The matching is case insensitive. FieldText queries which include commas and braces in the query have specific percent-encoding requirements. For information about percent-encoding, see FieldText, on page 310.
yourFields	is one or more fields. A document returns only if it contains one of these fields, and if the value in at least one instance of the field does not exactly match any of *yourStrings*. If you want to specify multiple fields, separate them with colons (there must be no space before or after a colon).

For example:

```
FieldText=NOTMATCH{cat}:ANIMAL
```

At least one instance of the ANIMAL field must have a value other than cat for the document to return as a result.

For example, if a document contains only:

```
#DREFIELD ANIMAL="cat"
```

it does not return as a result.

However, if the document contains:

```
#DREFIELD ANIMAL="cat"
#DREFIELD ANIMAL="dog"
```

it returns as a result, because one of the ANIMAL fields does not contain the value cat.

If you want to find documents in which the specified string is not present in any instance of the specified field, use the MATCH specifier with the Boolean operator NOT. For example, FieldText=NOT+MATCH{cat}:ANIMAL does not return any documents that have an ANIMAL field with the value cat, even if there are other ANIMAL fields with different values.

Related Topics

- MATCH, on page 243

NOTSTRING

The NOTSTRING field specifier (case sensitive) allows you to find documents in which at least one instance of the specified fields contains a value that does not contain any of the specified strings as a substring.

If there are one or more instances of a particular field in the document, the document returns as long as at least one instance does not contain any of the specified strings, even if another instance of the field does contain the string. The document does not return if all instances of the specified fields contain one of the specified strings as a substring.

`FieldText=STRING{yourStrings}:yourFields`

where:

yourStrings	is one or more strings. A document returns only if at least one instance of one of *yourFields* contains a value that is not a substring of any of these strings. FieldText queries which include commas and braces within the query have specific percent-encoding requirements. For information about percent-encoding, see FieldText, on page 310.
yourFields	is one or more fields. A document returns only if it contains one of these fields, and if the value in at least one instance of the field does not contain any of *yourStrings* as a substring. If you want to specify multiple fields, separate them with colons (there must be no space before or after a colon).

For example

`FieldText=NOTSTRING{cat,dog}:ANIMAL:TOPIC`

At least one instance of the `ANIMAL` or `TOPIC` field value must contain a value that does not contain the substring *cat* or *dog* for the document to return.

For example, if a document contains only:

`#DREFIELD ANIMAL="old cat"`

or

`#DREFIELD TOPIC="dogs like playing catch "`
`#DREFIELD ANIMAL="dog"`

it does not return as a result.

However, if the document contains:

`#DREFIELD TOPIC="dogs have trouble catching mice"`
`#DREFIELD ANIMAL="dog"`
`#DREFIELD ANIMAL="mouse"`

it returns as a result, because one of the `ANIMAL` fields does not contain the substring `cat` or `dog`.

If you want to find documents in which the specified string is not present in any instance of the specified field, use the `STRING` specifier with the Boolean operator `NOT`. For example, `FieldText=NOT+STRING{cat,dog}:ANIMAL` does not return any documents that have an `ANIMAL` field with the substring `cat` or `dog`, even if there are other `ANIMAL` fields with different values.

Related Topics

- STRING, on page 272

NOTWILD

The `NOTWILD` field specifier (case sensitive) allows you to find documents in which at least one instance of the specified field contains a value that does not match the specified Wildcard string.

If there are one or more instances of a particular field in the document, the document returns as long as at least one instance does not contain the specified string, even if another instance of the field does match. The document does not return if all instances of the specified fields contain the specified string.

If the query does not contain any Wildcard characters (? or *), the `NOTWILD` field specifier acts in the same way as the `NOTMATCH` field specifier.

```
FieldText=NOTWILD{yourStrings}:yourFields
```

where:

yourStrings	is one or more strings that contain Wildcards. A document returns only if at least one instance of one of *yourFields* does not contain any of these strings. FieldText queries that include commas and braces in the query have specific percent-encoding requirements. For information about percent-encoding, see FieldText, on page 310.
yourFields	is one or more fields. A document returns only if it contains one of these fields, and if the value in at least one instance of the field does not contain any of *yourStrings*. If you want to specify multiple fields, separate them with colons (there must be no space before or after a colon).

For example:

```
FieldText=NOTWILD{passi*incarnata}:Climbers:Plants
```

At least one instance of the `Climbers` or `Plants` field must contain a value which does not contain a phrase that begins with `passi` and ends with `incarnata` (for example, `passionflower incarnata` or `passiflora incarnata`) for this document to return as a result.

For example, if a document contains only:

```
#DREFIELD Climbers="passiflora incarnata"
```

or

```
#DREFIELD Climbers="passiflora incarnata"
#DREFIELD Plants="passionflower incarnata"
```

it does not return as a result.

However, if the document contains:

```
#DREFIELD Climbers="passiflora incarnata"
#DREFIELD Plants="passionflower incarnata"
#DREFIELD Climbers="bindweed"
```

it returns, because one of the `Climbers` fields contains a value that does not match the Wildcard string `passi*incarnata`.

If you want to find documents in which the specified string is not present in any instance of the specified field, use the `WILD` specifier with the Boolean operator `NOT`. For example, `FieldText=NOT+WILD{passi*incarnata}:Climbers` does not return any documents that have a `Climbers` field with a phrase that begins with `passi` and ends with `incarnata`, even if there are other `Climbers` fields with different values.

Related Topics

- [WILD, on page 250](#)
- [MATCH, on page 243](#)

Fields that Contain a Specified String

You can use the following field specifiers (case sensitive) to return documents with fields that contain a specified string.

STRING

The `STRING` field specifier (case sensitive) allows you to specify one or more strings, of which one must be contained as a substring in a specified field.

`FieldText=STRING{yourStrings}:yourFields`

where:

yourStrings	is one or more strings. A document returns only if one of these strings is a substring of the value in one of *yourFields*.
	FieldText queries that include commas and braces in the query have specific percent-encoding requirements. For information about percent-encoding, see [FieldText, on page 310](#).
yourFields	is one or more fields. A document returns only if it contains one of these fields, and if the value in this field contains one of *yourStrings* as a substring.
	If you want to specify multiple fields, separate them with colons (there must be no space before or after a colon).

Examples:

`FieldText=STRING{cat,dog}:ANIMAL`

The `ANIMAL` field value must contain the substring *cat* or *dog* for the document to return. For example, if the `ANIMAL` field has the value *scattering*, the document returns.

`FieldText=STRING{old cat}:ANIMAL:TOPIC`

The `ANIMAL` or `TOPIC` field value must contain the substring *old cat* for the document to return. For example, if the `ANIMAL` field has the value *old cat*, *old caterpillar*, or *bold cats*, the document returns.

`FieldText=STRING{example.com}:COMPANY`

The `COMPANY` field value must contain the substring *example.com* for the document to return. For example, if the `COMPANY` field has the value *example.com* or *http://www.example.com/content/home*, the document returns.

```
FieldText=STRING{a\,b}:MISC
```

The MISC field value must contain the substring *a,b* for the document to return. For example, if the MISC field has the value *a,b* or *a,b,c*, the document returns.

STRINGALL

The STRINGALL field specifier (case sensitive) allows you to specify one or more strings, which must all be contained as a substring in a specified field.

```
FieldText=STRINGALL{yourStrings}:yourFields
```

where:

yourStrings	is one or more strings. A document returns only if all these strings are substrings of the value in one of *yourFields*.
	If you want to specify multiple strings, separate them with commas (there must be no space before or after a comma). FieldText queries which include commas and braces in the query have specific percent-encoding requirements. For information about percent-encoding, see FieldText, on page 310.
yourFields	is one or more fields. A document returns only if it contains one of these fields, and if the value in this field contains *yourStrings* as substrings.
	If you want to specify multiple fields, separate them with colons (there must be no space before or after a colon).

Examples:

```
FieldText=STRINGALL{cat,dog}:ANIMAL
```

The ANIMAL field value must contain the substrings *cat* and *dog* for the document to return. For example, if the ANIMAL field has the value *grooming cats and dogs* or *doggedly scattering seeds*, the document returns.

```
FieldText=STRINGALL{old cat,young dog}:ANIMAL:TOPIC
```

The ANIMAL or TOPIC field value must contain the substrings *old cat* and *young dog* for this document to return. For example, if the ANIMAL field has the value *old cat chases young dog*, or *young doggedly chasing bold cats*, the document returns.

```
FieldText=STRINGALL{a\,b,e\,f}:MISC
```

The MISC field value must contain the substrings *a,b* and *e,f* for this document to return. For example, if the MISC field, for example, has the value *a,b,c,d,e,f* or *0=e,fx 1=da,ba*, the document returns.

SUBSTRING

The SUBSTRING field specifier (case sensitive) allows you to return documents whose field value is a substring of a specified string (or equal to a specified string).

```
FieldText=SUBSTRING{yourStrings}:yourFields
```

where:

yourStrings	is one or more strings. A document returns only if one of *yourFields* contains a substring of one of the specified strings. If you want to specify multiple strings, separate them with commas (there must be no space before or after a comma). FieldText queries that include commas and braces in the query have specific percent-encoding requirements. For information about percent-encoding, see FieldText, on page 310.
yourFields	is one or more fields. A document returns only if it contains one of these fields, and if the value in this field is a substring of *yourStrings*. If you want to specify multiple fields, separate them with colons (there must be no space before or after a colon).

Examples:

```
FieldText=SUBSTRING{Telecommunications,Technology}:SECTOR
```

The SECTOR field must contain a string that is a substring of *Telecommunications* or *Technology*. For example, if the SECTOR field has the value *Telecom* or *Technology*, the document returns. If the SECTOR field has the value *Latest Technology*, the document does not return.

Fields whose Values Match Specific Terms or Phrases

You can use the following field specifiers (case sensitive) to return documents in which specified fields contain specified terms or phrases.

TERM

The TERM field specifier (case sensitive) allows you to find documents with a specified field whose value contains a conceptual match for one or more terms that you specify. A conceptual match exists if a term you specify matches a term in a specified field after it has been stemmed.

> **NOTE:** If the language that you use does not match the DefaultLanguageType specified in the IDOL Content component configuration file, add the LanguageType parameter to your query action (see Specify the Language Type of a Query, on page 121).

```
FieldText=TERM{yourTerms}:yourFields
```

where:

yourTerms	is one or more terms. A document returns only if one of *yourFields* contains a value that includes a term which conceptually matches of one of the specified terms. To specify multiple terms, separate them with commas (there must be no space before or after a comma). FieldText queries that include commas and braces in the query have specific percent-encoding requirements. For information about percent-encoding, see FieldText, on page 310.
yourFields	is one or more fields. A document returns only if it contains one of these fields, and if a term in this field conceptually matches one of *yourTerms*. To specify multiple fields, separate them with colons (there must be no space before or after a colon).

Examples:

```
FieldText=TERM{shopping,centers}:DRETITLE
```

The DRETITLE field must contain a term that conceptually matches *shopping* or *centers* for the document to return. For example, if the DRETITLE field has the value *shop* the document returns, but if it has the value *bookshopping*, it does not return.

```
FieldText=TERM{training,football}:ITEM:PRODUCT
```

The ITEM or PRODUCT field must contain a term that conceptually matches *trainers* or *football* for the document to return. For example, if the ITEM or PRODUCT field has the value *train* or *footballers*, the document returns, while if it has the value *trainer* or *soccer*, it does not return.

TERMALL

The TERMALL field specifier (case sensitive) allows you to find documents with a specified field whose value contains conceptual matches of several terms that you specify. A conceptual match exists if the terms that you specify match terms in a specified field after they have been stemmed.

> **NOTE:** If the language that you are using does not match the DefaultLanguageType specified in the IDOL Content component configuration file, add the LanguageType parameter to your query action (see Specify the Language Type of a Query, on page 121).

```
FieldText=TERMALL{yourTerms}:yourFields
```

where:

yourTerms	is multiple terms. A document returns only if one of *yourFields* contains a value that includes terms which conceptually match the specified terms.
	Separate the terms with commas (there must be no space before or after a comma). FieldText queries that include commas and braces in the query have specific percent-encoding requirements. For information about percent-encoding, see FieldText, on page 310.
yourFields	is one or more fields. A document returns only if it contains one of these fields, and if a term in this field conceptually matches one of *yourTerms*.
	To specify multiple fields, separate them with colons (there must be no space before or after a colon).

Examples:

```
FieldText=TERMALL{shopping,centers}:DRETITLE
```

The DRETITLE field value must contain a term that conceptually matches *shopping* or *centers* for the document to return. For example, if the DRETITLE field has the value *town center shop*, the document returns.

```
FieldText=TERMALL{walk,climb}:DRETITLE:TITLE
```

The DRETITLE or TITLE field value must contain a term that conceptually matches *walking* or *climbing* for the document to return. For example, if the DRETITLE or TITLE field has the value *hill walking and rock climbing*, the document returns.

TERMEXACT

The `TERMEXACT` field specifier (case sensitive) allows you to find documents with a specified field that contains an exact match of any of the terms that you specify.

> **NOTE:** If the language that you are using does not match the `DefaultLanguageType` specified in the IDOL Content component configuration file, add the `LanguageType` parameter to your query action (see Specify the Language Type of a Query, on page 121).

```
FieldText=TERMEXACT{yourTerms}:yourFields
```

where:

yourTerms	is one or more terms. A document returns only if one of *yourFields* contains a value that exactly matches one of the specified terms.
	To specify multiple terms, separate them with commas (there must be no space before or after a comma). FieldText queries that include commas and braces in the query have specific percent-encoding requirements. For information about percent-encoding, see FieldText, on page 310.
yourFields	is one or more fields. A document returns only if it contains one of these fields, and if this field contains an exact match of one of *yourTerms*.
	To specify multiple fields, separate them with colons (there must be no space before or after a colon).

Examples:

```
FieldText=TERMEXACT{help,helped}:DRETITLE
```

The `DRETITLE` field value must contain the term *help* or *helped* for the document to return. For example, if the `DRETITLE` field has the value *helps* or *helping*, the document does not return.

```
FieldText=TERMEXACT{Word,Excel}:FILE:DATEI
```

The `FILE` or `DATEI` field value must contain the term *Word* or *Excel* for the document to return. For example, if the `FILE` or `DATEI` field has the value *WordPerfect*, the document does not return.

TERMEXACTALL

The `TERMEXACTALL` field specifier (case sensitive) allows you to find documents with a specified field that contains an exact match of all terms that you specify.

> **NOTE:** If the language that you are using does not match the `DefaultLanguageType` specified in the IDOL Content component configuration file, add the `LanguageType` parameter to your query action (see Specify the Language Type of a Query, on page 121).

```
FieldText=TERMEXACTALL{yourTerms}:yourFields
```

where:

yourTerms	is multiple terms. A document returns only if one of *yourFields* contains exact matches of the specified terms. Separate the terms with commas (there must be no space before or after a comma). FieldText queries that include commas and braces in the query have specific percent-encoding requirements. For information about percent-encoding, see FieldText, on page 310.
yourFields	is one or more fields. A document returns only if it contains one of these fields, and if this field contains an exact match of all *yourTerms*. If you want to specify multiple fields, separate them with colons (there must be no space before or after a colon).

Examples:

`FieldText=TERMEXACTALL{rabbits,eating,carrots}:DRETITLE`

This query returns only documents whose `DRETITLE` field contains all the specified terms (in their specified form). For example, a document whose `DRETITLE` field has the value *Rabbits like eating carrots*, or *The carrots were there but the rabbits ate all the cabbage*, returns as a result, but a document with a field that contains *Rabbits like to eat a carrot each day* does not return.

`FieldText=TERMEXACTALL{flour,milk,eggs}:DRETITLE:TITLE`

This query returns only documents whose `DRETITLE` or `TITLE` field contains all the specified terms (in their specified form). For example, a document whose `DRETITLE` or `TITLE` field has the value *Most cake recipes include milk, eggs and flour* return as a result, but a document with a field that contains *Use a cup of milk, two cups of flour and one egg* does not return.

TERMEXACTPHRASE

The `TERMEXACTPHRASE` field specifier (case sensitive) allows you to return documents in which a specified field contains an exact match of a phrase specified by you. IDOL Server matches your phrase before it applies stemming (it does not remove stop words). It ignores any punctuation in the specifier or field.

> **NOTE:** If the language that you are using does not match the `DefaultLanguageType` specified in the IDOL Content component configuration file, add the `LanguageType` parameter to your query action (see Specify the Language Type of a Query, on page 121).

`FieldText=TERMEXACTPHRASE{yourPhrase}:yourFields`

where:

yourPhrase	is a phrase. A document returns only if one of *yourFields* contains an exact match of the specified phrase.
yourFields	is one or more fields. A document returns only if it contains one of these fields, and if this field contains an exact match of *yourPhrase*. If you want to specify multiple fields, separate them with colons (there must be no space before or after a colon).

Examples:

```
FieldText=TERMEXACTPHRASE{Batman! and Robins}:FILM
```

A document whose FILM field contains *Showing now, Batman and Robin's film*, returns as a result, but a document whose FILM field contains *Showing now, 'Batman and Robin' the movie* does not return.

```
FieldText=TERMEXACTPHRASE{gift horse }:DRETITLE:TITLE
```

A document whose DRETITLE or TITLE field contains *looking a gift horse in the mouth* returns as a result, but a document whose DRETITLE or TITLE field contains *the gift horse's mouth had rotting teeth* does not return.

TERMPHRASE

The TERMPHRASE field specifier (case sensitive) allows you to return documents in which a specified field contains a conceptual match of a phrase that you specify. Content matches your phrase after it applies stemming (it does not remove stop words). It ignores any punctuation in the specifier or field.

> **NOTE:** If the language that you are using does not match the DefaultLanguageType specified in the IDOL Content component configuration file, add the LanguageType parameter to your query action (see Specify the Language Type of a Query, on page 121).

```
FieldText=TERMPHRASE{yourPhrase}:yourFields
```

where:

yourPhrase	is a phrase. A document returns only if one of *yourFields* contains a conceptual match of the specified phrase.
yourFields	is one or more fields. A document returns only if it contains one of these fields, and if this field contains a conceptual match of *yourPhrase*. If you want to specify multiple fields, separate them with colons (there must be no space before or after a colon).

Examples:

```
FieldText=TERMPHRASE{Batman! and Robins}:FILM
```

A document whose FILM field contains *Showing now: 'Batman and Robin'*, returns as a result.

```
FieldText=TERMPHRASE{gift horse }:DRETITLE:TITLE
```

A document whose DRETITLE or TITLE field contains *the gift horse's mouth had rotting teeth* returns.

Field Specifiers to Bias Result Scores

- **BIAS**. The BIAS field specifier (case sensitive) allows you to bias the score of results according to the numerical proximity of the specified field to a particular value.

  You can also boost the percentage relevance that is given to query results by setting up specific field process or by using multipliers. See Manipulate Result Relevance, on page 343 for details on BIAS and other methods that allow you to manipulate result scores.

- **BIASDATE**. The `BIASDATE` field specifier (case sensitive) allows you to boost the score of result documents by a specified percentage, based on how close the date in a specified field is to a specified date.

- **BIASDISTCARTESIAN**. The `BIASDISTCARTESIAN` field specifier allows you to boost the score of any document according to its distance from a specified point using Cartesian coordinates (X/Y).

- **BIASDISTSPHERICAL**. The `BIASDISTSPHERICAL` field specifier allows you to boost the score of any document according to its distance from a specified point using spherical coordinates (latitude and longitude).

- **BIASVAL**. The `BIASVAL` field specifier (case sensitive) allows you to bias the score of result documents by a specified percentage, based on whether they include a specific value in the specified field.

Field Specifier for Linked Queries

Linking queries allows you to query for documents based on criteria in the document, and in a connected document. For example, you might want to connect documents written by a particular user with values in the user profile.

The `LINK` field specifier allows you to form linked queries. For more information, see Link Queries, on page 300.

Fuzzy Search

If you are not quite sure how to spell some of the words that you want to query for, you can use the `Query` action to submit a fuzzy query to the IDOL Content component. A fuzzy query returns results that contain words which are similar to the query string.

Fuzzy Query Syntax

If you want to submit a fuzzy query, you must specify the `Query` action `Text` parameter in one of the following formats:

- `Text=`*myQueryText*`DREFUZZY(`*fuzzyQueryText*`)`

  For example:

  `http://`*IDOLhost*`:`*port*`/action=Query&Text=best selling author DREFUZZY(Rowlling)`

- `Text=DREFUZZY(`*fuzzyQuerytext*`)`

  For example:

  `http://`*IDOLhost*`:`*port*`/action=Query&Text=DREFUZZY(Caroll Jabberwalky)`

- You can also use Boolean and proximity operators within fuzzy queries.

  `Text=DREFUZZY(`*fuzzyQueryTextOPERATOROtherFuzzyQueryText*`)`

For example:

```
http://IDOLhost:port/action=Query&Text=DREFUZZY(Caroll AND Jabberwalky)
```

Adjust the Tolerance Level of a Fuzzy Search

By default, DREFUZZY internally determines an appropriate tolerance level when it judges whether words in a result document are similar enough to the words that you specify in the query. However, in exceptional circumstances you can postfix DREFUZZY with a numerical value to adjust this tolerance level:

DREFUZZY*N*(*fuzzyQuerytext*)

This value operates like a slider that increases or decreases the tolerance level. The higher the value is, the more words in result documents count as eligible matches to the words specified in the query. The lower the value is, the fewer words in result documents count as eligible matches to the words specified in the query.

Micro Focus does not recommend you specify a value higher than 6, because this can result in fuzzy matching that is so flexible that results are not related to the query.

Examples:

```
action=Query&Text=DREFUZZY1(Caroll Jabberwalky)
```

```
action=Query&Text=DREFUZZY3(Caroll Jabberwalky)
```

The second query in this example is more tolerant in accepting matches for the specified words than the first query, which means that it might return more results.

Parametric Search

The GetTagValues and GetQueryTagValues actions allow you to perform parametric searches.

A parametric search allows you to search for items by their characteristics (values in certain fields). When you provide fixed values in parametric fields, the parametric search returns consistent values in the nonfixed parametric fields. For example, you can search an IDOL wine database for specific wine varieties from a specific region by specifying which fields must match these characteristics. Only wines that match the specified variety and region return.

Configure the IDOL Content component for Parametric Fields

Before you perform parametric searches, configure the IDOL Content component to recognize parametric fields.

> **NOTE:** You must configure parametric field recognition before you index the data that you want to search.

To configure the IDOL Content component to recognize parametric fields

1. Open the IDOL Content component configuration file in a text editor.

2. In the `[Server]` section, set the `ParametricRefinement` parameter to **True**. (If the section does not contain this parameter, add it.)

   > **NOTE:** If you want to define parametric fields or add extra parametric fields, but have already indexed content into the IDOL Content component, also set `RegenerateParametricIndex` to **True**. This parameter allows Content to generate the files that it requires to internally identify parametric fields on startup, so that you need only to restart Content to use parametric fields, rather than having to reindex all your data.
   >
   > You can also use the `DREREGENERATE` index action to regenerate the parametric index while the server is running.

3. List a parametric field process in the `[FieldProcessing]` section.

   For example:

   ```
   [FieldProcessing]
   0=MyFirstProcess
   1=ParametricFields
   ```

4. Create a section for each field process that you listed, in which you create a property for the process (you define the property later by setting one or more applicable configuration parameters). Identify the fields that you want to associate with the process. For example:

   ```
   [MyFirstProcess]
   Property=MyProperty
   PropertyFieldCSVs=*/MyField,*/MyOtherField

   [ParametricFields]
   Property=Parametric
   PropertyFieldCSVs=*/Grape,*/Color,*/Region,*/Price
   ```

   > **NOTE:** The properties that you create must not have the same name as the processes.

5. Create a section for the parametric property in which you set the `ParametricType` parameter to **True**. This property enables Content to recognize the associated `PropertyFieldCSVs` fields as parametric fields. For example:

   ```
   [Parametric]
   ParametricType=True
   ```

6. Save and close the configuration file.

7. Restart the IDOL Content component for your changes to take effect.

You can now index your data into IDOL Server.

Perform a Parametric Search

After you configure the IDOL Content component to index and recognize parametric fields, you can use the following actions to perform a parametric search.

GetTagValues

This action allows you to specify one or more parametric fields and return all values that these fields contain in the IDOL Content component. It includes values in documents that you do not have access to, and values in documents that were deleted (unless you compacted the IDOL Content component data index after the documents were deleted).

For example:

```
action=GetTagValues&FieldName=Grape
```

This action requests the different values of the IDOL Content component `Grape` fields. It returns a list of all grape varieties stored in an IDOL Content component wine database, for example.

You can also restrict the action, so that it returns `Grape` field values only if they are in a document that also contains other specific fields that have specific values. For example:

```
action=GetTagValues&FieldName=Grape&Restriction=MATCH{Barossa Valley}:Region+MATCH
{Red}:Color
```

This action returns `Grape` field values only if they are in a document that also contains a `Region` field that has the value *Barossa Valley* and a `Color` field that has the value *Red*.

GetQueryTagValues

This action combines query text with one or more parametric fields. When Content performs the query, it finds documents that match the specified query text, and returns the values of the specified parametric fields for these documents. Unlike the `GetTagValues` action, the `GetQueryTagValues` action does not return field values in documents that you do not have access to, or in documents that were deleted.

For example:

```
http://localhost:5552/action=GetQueryTagValues&FieldName=GRAPE,COUNTRY&Text= A
smooth red wine that complements game
```

This action requests the different values of the `GRAPE` and `COUNTRY` fields of documents that are conceptually similar to the specified `Text`.

You can also restrict the action by combining it with various action parameters. For example:

```
http://localhost:5552/action=GetQueryTagValues&FieldName=GRAPE,COUNTRY&Text= A
smooth red wine that complements game&MaxValues=10&Sort=Alphabetical
```

This action requests the 10 top values of the `GRAPE` and `COUNTRY` fields of documents that are conceptually similar to the specified `Text`. Content displays the values in alphabetical order when it returns them.

```
http://localhost:5552/action=GetQueryTagValues&FieldName=GRAPE,COUNTRY&Text= A
smooth red wine that complements game&DocumentCount=True
```

This action requests the different values of the GRAPE and COUNTRY fields of documents that are conceptually similar to the specified Text. The DocumentCount parameter instructs Content to return the number of documents that contain each value.

```
http://localhost:5552/action=GetQueryTagValues&FieldName=GRAPE,COUNTRY&Text= A
smooth red wine that complements game&FieldDependence=True
```

This action requests the different values of the GRAPE and COUNTRY fields of documents that are conceptually similar to the specified Text. The FieldDependence parameter instructs Content to find sets of values that occur together. If Content finds documents that contain the first parametric field listed, it checks if they also contain the subsequently listed parametric fields and returns them. You can also use the FieldDependenceMultiLevel parameter to display these results in a hierarchical structure.

For further details on available parameters for the GetTagValues and GetQueryTagValues actions, refer to the *IDOL Server Reference*.

Related Topics

- Display Online Help, on page 30

Proper Names Search

If you want the IDOL Content component to recognize names and treat them as a unit, you must enable proper names searches.

> **NOTE:** To search for exact matches of phrases as well as names, enable AdvancedSearch before you index your content into the IDOL Content component, and set the ProperNames parameter to **7** (see below).

Related Topics

- Phrase Search, on page 226

Enable Proper Names Searches

> **NOTE:** You must enable proper names searches before you index the data that you want to query against.

To enable proper names searches

1. Open the IDOL Content component configuration file in a text editor.

2. Before you store content in the IDOL Content component, terms are always stemmed and stop words are always discarded. If you want to store proper name terms (adjacent terms that begin with a capital letter) in addition to the normal content, you can set the ProperNames

parameter in the `[LanguageTypes]` section to one of the following values.

Value	Meaning
0	Proper name terms are not stored.
1	Adjacent capitalized terms are compounded, then stemmed and indexed as a unit. For example, *Sam James* is indexed as *SAMJAM*.
2	Adjacent terms are compounded (regardless of capitalization), then stemmed and indexed as a unit. For example, *bottlenose dolphins* is indexed as *BOTTLENOSEDOLPHIN*. **NOTE:** This setting considerably increases the number of terms in the IDOL Content component index, which can slow down its performance.

Use the following `ProperNames` options only if you need to query for proper names that contain stop words (for example, *The Who* or *The Queen*).

Value	Meaning
3	Adjacent capitalized stop words are compounded, then stemmed and indexed as a unit. For example, *And His* is indexed as *ANDHI*. Adjacent capitalized terms are compounded, then stemmed and indexed as a unit. For example, *Sam James* is indexed as *SAMJAM*. Capitalized stop words adjacent to capitalized terms are treated as individual terms. For example, *The Queen* is treated as *THE* and *QUEEN*, according to your stop word rules.
4	Capitalized stop words are compounded with adjacent capitalized terms, then stemmed and indexed as a unit. For example, *The Bells* is indexed as *THEBEL*, and *Calling Will* is indexed as *CALLINGWIL*. Adjacent capitalized stop words are compounded, then stemmed and indexed as a unit. Adjacent capitalized terms are compounded, then stemmed and indexed as a unit.
5	Adjacent capitalized stop words are compounded and indexed unstemmed as a unit. For example, *And His* is indexed as *ANDHIS*. Adjacent capitalized terms are compounded and indexed unstemmed as a unit. For example, *Sam James* is indexed as *SAMJAMES* Capitalized stop words adjacent to capitalized terms are treated as individual terms.
6	Capitalized stop words are compounded with adjacent capitalized terms, and indexed unstemmed as a unit. For example, *The Bells* is indexed as *THEBELLS*, and *Calling Will* is indexed as *CALLINGWILL*.

Value	Meaning
	Adjacent capitalized stop words are compounded and indexed unstemmed as a unit.
	Adjacent capitalized terms are compounded and indexed unstemmed as a unit.
7	Capitalized stop words are compounded with adjacent capitalized terms, and indexed unstemmed as a unit.
	Adjacent capitalized stop words are compounded and indexed unstemmed as a unit.
	Adjacent capitalized terms are treated as individual terms. For example, *Sam James* is indexed as *SAM* and *JAME*.
	NOTE: Micro Focus recommends that you use this setting if you set `AdvancedSearch` to **True** in the `[Server]` section of the IDOL Content component configuration file.

You must set this parameter for each of the languages that you want to enable name recognition for (if the language settings do not include the `ProperNames` parameter, you must add it). For example:

```
[LanguageTypes]
DefaultLanguageType=English
LanguageDirectory=C:\HewlettPackardEnterprise\IDOLServer\common\langfiles
0=English
1=Deutsch
2=Francais

[English]
Encodings=UTF8:englishUTF8
ProperNames=1

[Deutsch]
Encodings=UTF8:germanUTF8
ProperNames=1

[Francais]
Encodings=UTF8:frenchUTF8
ProperNames=1
```

3. Save and close the configuration file.

4. Restart the IDOL Content component for your changes to take effect.

5. Index documents into the IDOL Content component. After you finish indexing, Content treats any `Query` action as a proper name query.

Example Proper Name Searches

The following table describes how the `ProperNames` setting affects the terms that the IDOL Content component stores for the sentence *Tom Jones And His greatest hits*.

Original	Tom		Jones		And His	greatest		hits
0	TOM		JONE			GREAT		HIT
1	TOM	TOMJON	JONE			GREAT		HIT
2	TOM	TOMJON	JONE			GREAT	GREATESTHIT	HIT
3	TOM	TOMJON	JONE		ANDHI	GREAT		HIT
4	TOM	TOMJON	JONE	JONESAND	ANDHI	GREAT		HIT
5	TOM	TOMJONES	JONE		ANDHIS	GREAT		HIT
6	TOM	TOMJONES	JONE	JONESAND	ANDHIS	GREAT		HIT
7	TOM		JONE	JONESAND	ANDHIS	GREAT		HIT

If the IDOL Content component contains these documents, the following queries produce different results according to your `ProperNames` settings.

Doc 1:	Doc 2:
Tom Waits and The The in concert with Norah Jones	Tom Jones and the the in concert with Katie Melua

- `action=Query&Text=Tom Jones`

  If you set `ProperNames` to **0** or **7**, both documents return with the same relevance (in both cases, the query to Content has the terms *TOM* and *JONE*, which match both documents).

  If you set `ProperNames` to **1**, **2**, **3**, **4**, **5**, or **6**, Doc 2 returns with a higher relevance than Doc 1 (because it matches not just the terms *TOM* and *JONE*, but also *TOMJON* or *TOMJONES*).

- `action=Query&Text=tom jones`

  If you set `ProperNames` to **0**, **1**, **3**, **4**, **5**, **6**, or **7**, both documents return with the same relevance (in both cases, the query to Content has the terms *TOM* and *JONE*, which match both documents).

  If you set `ProperNames` to **2**, Doc 2 returns with a higher relevance than Doc 1 (because it matches not just the terms *TOM* and *JONE*, but also *TOMJON*).

- `action=Query&Text=The The`

  If you set `ProperNames` to **0**, **1**, or **2**, the query returns no results (because Content discards both instances of the word *The* as stop words).

If you set `ProperNames` to **3**, **4**, **5**, **6**, or **7**, only Doc 1 returns (because in all these cases the query to Content has the term *THETH* or *THETHE*, which matches only Doc 1).

- `action=Query&Text=the the`

If you set `ProperNames` to **0**, **1**, **2**, **3**, **4**, **5**, **6**, or **7**, no results return (because Content discards both instances of the word *the* as stop words).

Soundex Keyword Search

If the spelling of a keyword is not quite accurate but phonetically correct, a Soundex keyword query returns results that contain the keyword and phonetically similar keywords (using a Soundex algorithm).

Enable Soundex Keyword Searches

> **NOTE:** You must enable Soundex keyword searches before you index the data that you want to search.

To enable Soundex keyword searches

1. Open the IDOL Content component configuration file in a text editor.

2. In the `[Server]` section, set the `Soundex` parameter to one of the following values. If the `[Server]` section does not contain the `Soundex` parameter, add it.

1	The IDOL Soundex extension.
2	Standard Soundex.
4	Double Metaphone.
8	The IDOL Double Metaphone extension.

3. Save and close the configuration file.

4. Restart the IDOL Content component for your changes to take effect.

5. Index documents into the IDOL Content component. After you finish indexing, you can perform Soundex keyword searches by using the `Query` action. The phonetic indexing and query uses the algorithm that you selected.

Perform Soundex Keyword Searches

You can use Soundex with single or multiple keywords, or as part of a query text string. For example:

```
http://localhost:5552/action=Query&Text=SOUNDEX(einstine)
```

```
http://localhost:5552/action=Query&Text=albert SOUNDEX(einstine) examined the
phenomenon discovered by max SOUNDEX(plank) in 1905
```

Synonym Search

A synonym query returns results that are conceptually similar to the terms in a `Query` action `Text` parameter or conceptually similar to the synonyms that are available for the `Text` terms.

You can configure and use synonyms in several different ways.

In a query you can use one of the following methods to specify synonyms:

- **Use the `SYNONYM` operator**. The `SYNONYM` operator is a query operator that allows you to specify a list of synonymous terms. Micro Focus recommends that you use this method in most cases. See The SYNONYM Operator, below.

- **Use the `Synonym` parameter in queries**. The `Synonym` parameter enables synonym lists for a query. To use this option, you must configure synonym lists in your IDOL Content component. See Enable Synonym Lists, on the next page.

You can also replace query text directly (without using the `SYNONYM` operator). However, in this case you might find query weighting is affected by the additional query terms.

You can use the following methods to store your synonym lists:

- **Configure synonym lists in the IDOL Content component**. A synonym list is a simple text file that defines synonyms for a list of terms and phrases. When you configure a synonym list, Content uses it at index time to mark synonymous terms in the index. This method enables you to use the `Synonym` parameter.

- **Use Query Manipulation Server**. Query Manipulation Server (QMS) acts as a proxy for your queries, with access to your main query Content component, and the Promotion Agentstore. You store synonym lists as documents in the Promotion Agentstore. When you send a query to QMS, it searches the Promotion Agentstore for synonyms, modifies the query text, and forwards the modified query to the main Content component.

- **Use a Synonym Server**. The Synonym Server is an IDOL Content component that stores synonym lists as documents. You retrieve the synonyms in queries, and use the results to modify the original query text. This method is similar to using QMS, but you must set up your front end to manage the synonym query and text replacement.

Related Topics

- The SYNONYM Operator, below

- Enable Synonym Lists, on the next page

- Set up a Synonym Server, on page 292

The SYNONYM Operator

The `SYNONYM` operator allows you to specify synonyms in the `Query` action `Text` parameter.

In this operator, you specify a list of synonymous words and phrases. The IDOL Content component treats the terms in the operator as individual query terms, separated by Boolean OR operators (that is, results can contain any of the listed terms).

For query weighting, Content scores all the terms in the SYNONYM operator as if they were a single term. This approach means that results that contain only one of the synonymous terms do not necessarily have a lower relevance score than one that contains more of the synonyms.

For example:

```
http://localhost:9010/action=Query&Text=SYNONYM(cat moggy kitten mouser) NEAR sale
```

The effective query text is (cat OR moggy OR kitten OR mouser) NEAR sale. However, if you use this query text without the SYNONYM operator, a document that contains the phrase *cat sale* twice scores lower than a document that contains the phrases *cat sale* and *kitten sale*, because it contains fewer of the query terms. With the SYNONYM operator, these documents score the same.

You can use quotation marks to define a synonym phrase. For example:

```
SYNONYM(dog labrador "golden retriever")
```

> **NOTE:** You cannot use the SYNONYM operator in an exact phrase search, for example, to match an exact phrase where one of the terms has several synonyms. In this case, you can use the NEAR operator to approximate the exact phrase search.

The SYNONYM operator is not the same as configuring a synonym list, because it does not affect how Content stores the documents in the index.

You can use the SYNONYM operator in combination with Query Manipulation Server, or a Synonym server. In both cases, you store the details of your synonyms in the external server, and replace the term in the query text with the SYNONYM operator and the list of synonyms.

For details about how to use synonyms in Query Manipulation Server, refer to the *Query Manipulation Server Administration Guide*. For details about using the Synonym Content Component, see Set up a Synonym Server, on page 292.

Enable Synonym Lists

To enable synonym lists, you must create a synonym file, and configure the IDOL Content component to use it.

When you configure a synonym list, Content uses it at index time to locate the synonyms, and it applies the synonyms at query time, when you use the Synonym parameter.

Synonym lists might be useful if you want to use synonym matching for a small number of terms (100 or less). For longer lists of synonyms, it might affect the performance of indexing and query, and Micro Focus recommends that you use Query Manipulation Server instead.

> **NOTE:** You must set up the synonym file before you index the data that you want to search.

The following sections describe how to set up and configure Content to use the synonym file:

Create a Synonym File

The synonym file contains the synonyms that you want to apply in your queries.

To set up a synonym file

1. Create a text file and save it in the IDOL Content component `IDOL/content` directory.

2. Create a section for each language. The section name must be the name of a valid language type configured in the IDOL Content component (that is, a language and encoding).

> **NOTE:** You do not need to create a section for each encoding in the same language. Content uses the same synonyms for this language, in all encodings. For example, if you add an `[EnglishUTF8]` section, you do not also need to add an `[EnglishASCII]` section.

For example:

```
[EnglishUTF8]
[GermanUTF8]
```

3. In each section, create a line for each list of synonyms. Use the same encoding that you specified in the section name. Content identifies all terms on the same line as synonyms. Separate each synonym term with commas. The individual synonym terms can contain spaces, but must not contain any punctuation.

For example:

```
[EnglishUTF8]
cat,feline,grimalkin,moggy,mouser,puss,tabby
dog,cur,hound,mans best friend,mongrel,mutt,pooch,puppy

[GermanUTF8]
Katze,Mietze,Mietzekatze,Mietzekater,Kater,Mulle,Kätzchen
Hund,Wau Wau,Hündin,Töle,Kläffer,Hündchen,Welpe
```

4. Save the synonym file.

Configure the IDOL Content component to Use a Synonym File

To configure Content to use a synonym file, you must configure a field process to specify the fields that you want to apply the synonym processing to. For these fields, Content processes and stores synonym information when you index documents, and retrieves the synonym information for synonym queries.

You must also configure the synonym job, which defines the location of the synonym file to use.

To configure Content to use a synonym file

1. Open the IDOL Content component configuration file in a text editor.

2. In the `[FieldProcessing]` section of the IDOL Content component configuration file, set up a synonym process. This process allows Content to determine the fields to which it must apply synonym settings. For example:

```
[FieldProcessing]
0=SynonymMatch
```

3. Create a section for the synonym process you listed.

   - Set `Property` to the name of the property configuration section for the process. The property that you create must not have the same name as the process.

   - Set `PropertyFieldCSVs` to the list of fields that you want to associate with the process. These fields must also be configured as `Index` fields.

     To identify the fields, use the format */FieldName* to match root-level fields, */FieldName* to match all fields except root-level, or */Path/FieldName* to match fields that the specified path points to.

   For example:

```
[SynonymMatch]
Property=ApplySynonymMatch
PropertyFieldCSVs=*/DRETITLE,*/DRECONTENT
```

   In this example, Content processes synonyms for terms that occur in the `DRETITLE` and `DRECONTENT` fields. When you run a synonym search, Content searches for the query terms and their synonyms in these fields.

4. Create a section for the property. In this section, set `SynonymType` to the name of the synonym job that you want to apply to these fields.

```
[ApplySynonymMatch]
SynonymType=Synonym_job
```

5. Find the `[Synonym]` section of the IDOL Content component configuration file, or create one if it does not exist. In this section, list the name of the synonym job you want to create. You can set up multiple jobs. However, you normally require only one. For example:

```
[Synonym]
0=Synonym_job
```

6. Create a configuration section with the name of the synonym job.

   In this section, set `File` to the name of the text file that defines your synonym lists (see Create a Synonym File, on the previous page).

   For example:

```
[Synonym_job]
File=animals.txt
```

7. Set any other synonym settings that you want to use for this synonym job. For more information about the available settings, see the *IDOL Server Reference*. For example:

```
[Synonym_job]
File=animals.txt
MaxExpandLevel=1
```

8. Save and close the configuration file.

9. Restart the IDOL Content component for your changes to take effect.

Perform Synonym Searches

After you create a synonym file and configure Content to use it, you can index your content. During the index process, Content processes the synonym rules and stores information about the synonyms in your index.

To run a synonym search, you send a `Query` action with the `Synonym` parameter set to **True**. For example:

```
http://localhost:5552/action=Query&Text=Felix is a great mouser&Synonym=True
```

This query returns documents that match the term *mouser*, as well as documents that match any of the terms configured as synonyms for the term *mouser*.

> **NOTE:** Content identifies and searches for synonyms only in the configured synonym fields. For any index fields that you do not define as synonym fields, Content searches only for the exact query terms.

Synonym queries return synonym links, which indicate internally the set of terms and phrases that a synonym corresponds to. You can use the `TermExpand` action, with the `Expansion` parameter set to **Synonym**, to expand these internal links to see what these term phrases are.

Update Synonym Files

Content processes synonyms at index time. When you want to update or add a synonym in your synonym file, you must reindex your content for the modified synonyms to be available over all content. If you do not reindex, the synonym changes apply only to new documents.

Set up a Synonym Server

A synonym server is an IDOL Content component in which you index documents that define synonyms, rather than ordinary document content.

In most cases, Micro Focus recommends that you use Query Manipulation Server (QMS) to manage your synonyms and perform synonym queries. In QMS, you use the Promotion Agentstore (a specially configured IDOL Content component) to store synonym documents. You query QMS, which manages retrieving the synonyms and modifying the query text. It then forwards the modified query to your data Content component and returns the results.

The synonym server approach is similar, but you must manage the synonym query process in your Front End component.

The following sections describe how to set up and use a synonym server:

- Install the Synonym Server, below

- Create and Index Synonym Documents, below

- Perform Synonym Searches with a Synonym Server, below

For more information about how to use synonyms in QMS, refer to the *Query Manipulation Server Administration Guide*.

Install the Synonym Server

Install and configure an IDOL Content component as normal. For details about how to install the component, refer to the *IDOL Getting Started Guide*.

If you install the synonym server on the same machine as existing IDOL components, you must ensure that the servers use different ports.

Create and Index Synonym Documents

In the synonym Content component, each document in an IDX file describes a list of synonymous terms. You can create synonym files by using an IDOL Connector to crawl a thesaurus Website, or by creating the files manually.

The following table describes the main fields that you use in the synonym documents.

Field	Description
#DREREFERENCE	A unique reference string for the synonym document. Usually this reference is a file name, URL, or unique code number.
#DRECONTENT	A list of associated single words, separated by carriage returns or spaces (you cannot list phrases).
#DREENDDOC	A delimiter that indicates the end of the document.

For example:

```
#DREREFERRENCE Syn1.txt
#DRECONTENT cat feline grimalkin moggy mouser tabby kitten
#DREENDDOC
#DREREFERRENCE Syn2.txt
#DRECONTENT dog cur hound mongrel mutt pooch puppy
#DREENDDOC
```

If you use a Connector to create the synonym file, the connector can also index the documents. If you create the file manually, you can index it by using a DREADD index action. See DREADD: Index IDX and XML Files Directly, on page 51.

Perform Synonym Searches with a Synonym Server

The following procedure describes how to perform a synonym search. Normally, you set up a front end interface to perform these steps.

To perform synonym searches

1. Process the original user query text, and send queries to the Synonym Content component for each of the query terms. For example, if the original query text is *Felix is a great mouser*, you must send queries to the Synonym Server for *Felix*, *great*, and *mouser*.

   For example:

   ```
   http://synonymServerHost:synonymServerPort/action=Query&Text=Felix
   http://synonymServerHost:synonymServerPort/action=Query&Text=great
   http://synonymServerHost:synonymServerPort/action=Query&Text=mouser
   ```

   Each query returns any synonym documents that contains the term.

2. For terms where the synonym query returns a synonym document, extract the content of the synonym document. Modify the query string to include the list of synonyms. Micro Focus recommends that you use the SYNONYM operator in the modified query (see The SYNONYM Operator, on page 288).

   For example, you might change *mouser* to SYNONYM(cat feline grimalkin moggy mouser tabby kitten).

3. Send the modified query string to the main Content component:

   ```
   http://IDOLhost:port/action=Query&Text=Felix is a great SYNONYM(cat feline
   grimalkin moggy mouser tabby kitten)
   ```

   This query returns documents that conceptually match the term *mouser*, as well as documents that conceptually match any of the terms that the synonym server returns as synonyms. The use of the SYNONYM operator means that Content calculates the query result relevance scores as if all the synonyms are a single term.
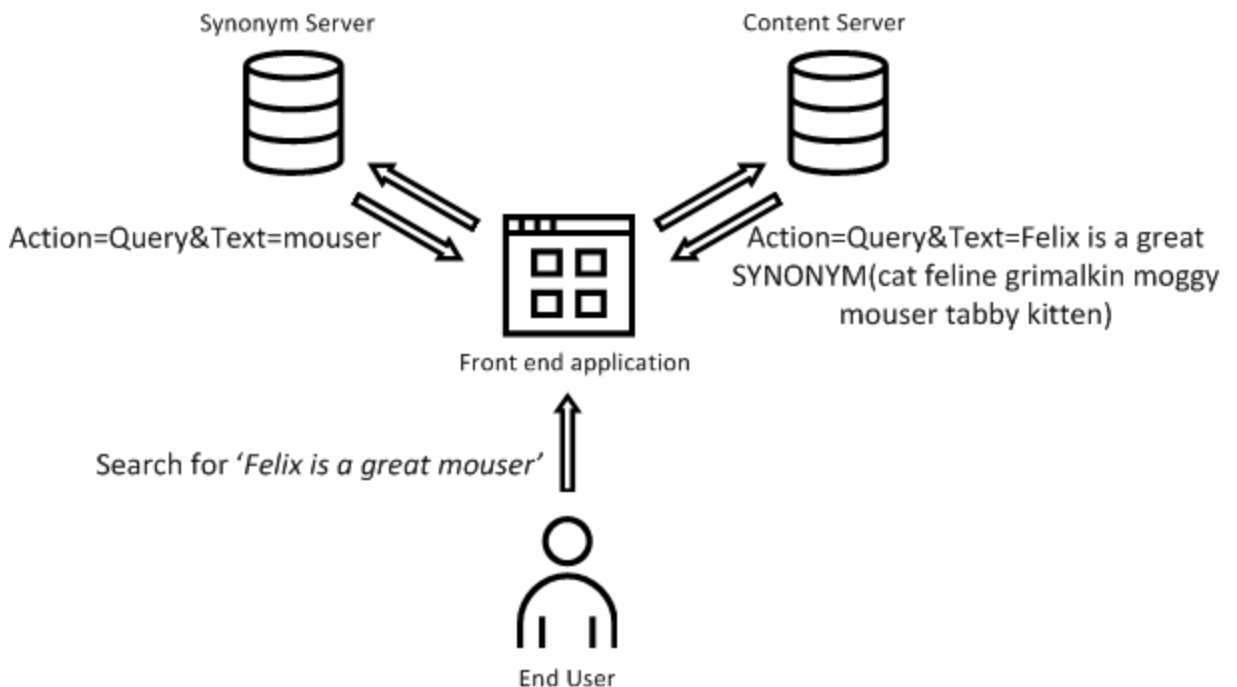
Analytics Functions

Analytics functions allow you to perform an operation on document fields at query time, and create a temporary field that contains the result, which you can use in other query operations such as `FieldText`. The `UserMetaFields` parameter for the `Query` and `GetQueryTagValues` actions specifies an operation that you want to perform, and the name of the field that you want to create.

For example, you can use `UserMetaFields` to create a new field that contains the mean of the values of specified fields in a document. You can then use the average in the `FieldText` parameter to find documents with a particular mean value, or in the `Sort` parameter to sort the results by the mean value.

For operations that you want to use regularly, you might want to use Connector Framework Server (CFS) to process documents before you index, and add a field to the document that you can use in queries. In general, storing the results as permanent fields gives better query performance. However, you can use `UserMetaFields` to perform operations that you do not need regularly, or to create and use fields that you did not consider when you originally indexed the data. You can also use `UserMetaFields` if you want to update fields without recalculating the value of the stored field.

Create Metafields

The `UserMetaFields` parameter has the following syntax:

`UserMetaFields=OPERATION{UserFieldName}:DocumentFieldName`

where:

OPERATION	is the operation that you want to perform on the contents of the specified fields in your documents.
UserFieldName	is the name of the user metafield that you want to create.
DocumentFieldName	is the name of the fields in your documents that you want to run the specified operation on. To specify multiple fields, separate the field names with colons (:) (there must be no space before or after a colon).

> **NOTE:** `UserMetaFields` operations are more efficient when you use them on fields that are configured as `NumericType`, except for the `COUNTA` operation, which is more efficient on fields that are configured as `CountType`, `MatchType`, or `ParametricType`.

To specify multiple operations, separate the operations with a plus sign (+) or a space.

You can optionally use a constant in the `UserMetaFields` definition, in the following form:

`UserMetaFields=OPERATION{UserFieldName, Constant}:DocumentFieldName`

The constants can be positive or negative, and they can include decimals (for example 1.23) and scientific notation (for example 1E2). The meaning of the constants depends on the operation that you use.

> **NOTE:** The IDOL Content component uses the constant only if the `DocumentFieldName` contains values. For example, for the operation `MIN{MINSCORE,10}:SCORE`, Content does not create a `MINSCORE` user metafield if the `SCORE` field is missing for a document.

The following table describes the operations that you can perform on document fields.

Operation	Description	Constant
ADD	The sum of all the values in the specified fields.	The initial value to add the field values to.
SUB	The value of the first field name (or the sum of the values in all instances of the first field name), minus the values of the remaining fields.	An additional value to subtract.
MUL	The product of all the values in the specified fields.	The initial value to multiply the field values by.
DIV	The value of the first field name (or the product of the nonzero values in all instances of the first field name), divided by the nonzero values from the remaining fields.	An additional value to divide by. If the constant value is 0, the constant is ignored.
MIN	The smallest value in the specified fields.	An initial value to include in the minimum calculation.
MAX	The highest value in the specified fields.	An initial value to include in the maximum calculation.
MEAN	The mean of the values in the specified fields.	No effect.
COUNT	The number of numeric values in the specified fields (it does not count any non-numeric values).	No effect.
COUNTA	The number of non-empty occurrences of the specified field, regardless of their value.	No effect.
ABS	The absolute value of the value in the specified field (that is, removing any negative sign).	No effect.
ROUND	The nearest integer to the value in the specified field, or the nearest multiple of the specified constant.	The accuracy to round to. For example, 0.01 rounds to two decimal places.
FLOOR	The nearest integer that is lower than the value in the specified field, or the nearest multiple of the specified constant lower than the value.	The accuracy to round down to. For example, 0.01 rounds to two decimal places.
LOG	The log (to the base of the specified constant) of the value of the first instance in a document of the specified field. The constant is required for this function.	The base of the log.
POW	The value of the first instance in a document of the	The power to use.

Operation	Description	Constant
	specified field, raised to the power of the specified constant. The constant is required for this function.	

> **NOTE:** The ABS, ROUND, FLOOR, LOG, and POW operations act on a single value for each document. Micro Focus recommends that you use these operations on fields that have only a single value, or on the output of other user metafield operations. If you apply these operations to a field with multiple values, it returns the result of the operation on the first field value, which might depend on your configuration.

You can also use the following distance operations. In this case, you must specify two constants and two field names that contain the location information.

Operation	Description	Constant
DISTSPHERICAL	The distance in kilometers between the location specified by the constants, and the location specified by the document fields, in latitude and longitude. You can specify the location fields as two fields in the form :*LAT_FIELD*:*LONG_FIELD*, or a single field that contains Well-known text format POINT definitions.	The latitude and longitude of the location you want to calculate distances from. Specify these values as *Lat,Long*.
DISTCARTESIAN	The distance between the point specified by the constants, and the point specified by the document fields, in cartesian coordinates. You can specify the location fields as two fields in the form :*X_FIELD*:*Y_FIELD*, or a single field that contains Well-known text format POINT definitions.	The X and Y coordinate of the point that you want to calculate distances from. Specify these values as *X,Y*.

> **NOTE:** The DISTSPHERICAL and DISTCARTESIAN operations act on a single pair of values for each document (one value for each coordinate field). Micro Focus recommends that you use these operations on fields that have only a single value, or the output of other user metafield operations. If you apply these operations to a field with multiple values, it returns the result of the operation on the first field value, which might depend on your configuration.
>
> If both of your coordinate fields are configured as NumericType, Micro Focus recommends that you set XMLFullStructure to **True** in your configuration. With this additional configuration, distance-based metafields produce results consistent with distance-based sorting.

You can apply UserMetaFields operations to other user metafields created in the same query. For example, you might use an operation to create one field, and then use that field in a second operation. You must create a metafield before you reference it in another operation (that is, the operation to create the field must appear to the left of the operation that uses it in the UserMetaFields parameter).

When you use your metafield in another operation, the field name has the format autn_user_*UserFieldName*, where *UserFieldName* is the value that you specified in the UserMetaField parameter.

For example, the following query creates the `TaxAmount` user metafield from the document fields, and then uses this new metafield to calculate the total price.

```
UserMetaFields=MUL{taxAmount}:basePrice:taxRate,ADD{totalPrice}:basePrice:autn_
user_taxAmount
```

Use Metafields

The IDOL Content component does not permanently store the values for user metafields, but you can use the results in the `Query` or `GetQueryTagValues` action.

When you use your metafield in another operation, the field name has the format `autn_user_UserFieldName`, where *UserFieldName* is the value you specified in the `UserMetaField` parameter.

FieldText

You can use metafields in `FieldText` operators. For example:

```
action=Query&Text=Television&UserMetaFields=ADD{TOTAL}:PRICE:TAX&FieldText=EQUAL
{500}:autn_user_TOTAL
```

In this example, the `UserMetaFields` operation adds the value of the `PRICE` and `TAX` fields in documents that match the query for `Television`. It creates the `TOTAL` field for each document to store this result, and returns only documents where the new `TOTAL` field has a value of `500`.

Related Topics

- FieldText Search, on page 241

Sort

You can use metafields in the `Sort` parameter to sort your result documents. For example:

```
action=Query&Text=Television&UserMetaFields=MIN{LOWSCORE}:REVIEWSCORE&Sort=autn_
user_LOWSCORE:decreasing
```

In this example, the `UserMetaFields` operation takes the lowest value of the `REVIEWSCORE` fields in a document that matches the query for `Television`. It creates a `LOWSCORE` field for each document to store this result, and returns documents in decreasing order of the `LOWSCORE` field.

Related Topics

- Change Result Sorting (Display Order), on page 315

PrintFields

You can specify metafields in the `PrintFields` parameter to display the metafields in the results.

> **NOTE:** By default, when you set `Print` to **all**, metafields do not return in results. In this case, you can set the `PrintFields` parameter to a list of metafields to display them.

```
action=Query&Text=Television&UserMetaFields=MIN
{LOWSCORE}:REVIEWSCORE&Print=Fields&PrintFields=autn_user_LOWSCORE
```

In this example, the `UserMetaFields` operation takes the lowest value of the `REVIEWSCORE` fields in a document that matches the query for `Television`. It creates a `LOWSCORE` field for each document to store this result, and displays the `LOWSCORE` field in the results.

GetQueryTagValues FieldName

In the `GetQueryTagValues` action, you can use metafields as the `FieldName` that you want to return results for. For example:

```
action=GetQueryTagValues&Text=Television&UserMetaFields=MEAN
{AVERAGE}:PRICE&FieldName=autn_user_AVERAGE
```

In this example, the `UserMetaFields` operation calculates the mean value of the `PRICE` fields in documents that match the query for `Television`. It creates an `AVERAGE` field for each document to store this result, and then returns the values that occur in the `AVERAGE` field as tags for the parametric search.

```
action=GetQueryTagValues&Text=Restaurants&UserMetaFields=DISTSPHERICAL
{HOME,51.3,0.73}:LAT:LONG&FieldName=autn_user_HOME&Ranges=FIXED{0,5,10,20,.}:autn_
user_HOME&DocumentCount=true
```

In this example, the `UserMetaFields` operation finds documents that match the query for `Restaurants`, and uses the document `LAT` and `LONG` fields to calculate the distance from a location in London. It creates a `HOME` field for each document to store this distance, and then returns the number of documents that occur within the specified distances from the original point, grouped by the specified ranges (0-5, 5-10, 10-20, 20+).

Related Topics

- Parametric Search, on page 280

AgentBoolean

You can use metafields as part of an agent, and use the metafield in AgentBoolean queries.

For example, you might create the following agent IDX file, and index it into the Agentstore component:

```
#DREREFERENCE UserMetaFields
#DREFIELD AGENTBOOL="usermeta"
#DREFIELD FTF="GREATER{5}:autn_user_X"
#DREFIELD AGENTPARAMS="UserMetaFields=MEAN{X}:NUMBER"
#DRECONTENT
usermeta
#DREENDDOC
```

This IDX file includes the `UserMetaFields` parameter in the `AGENTPARAMS` field, and a `FieldText` expression that uses the resulting metafield in the `FTF` field.

For this example, when you send a `Query` action to the Agentstore to return agents that match a document, you also set the following parameters in the action:

- `TextParse=True`

- `AgentBooleanField=AGENTBOOL`

- `FieldTextField=FTF`

- `AgentParamsField=AGENTPARAMS`

For a document to match this example agent, it must include the term `usermeta`. When the document matches the `usermeta` term, the agent calculates the mean of the values in the `NUMBER` fields, and creates an `X` field to contain the mean. For the agent to return as a match for the document, the document `X` metafield must have a value greater than 5.

Related Topics

- AgentBoolean Agents and Categories, on page 185

- Perform AgentBoolean Queries, on page 189

Link Queries

You can use link queries to query for documents based on criteria in the document, and in a connected document. For example, you might want to connect documents written by a particular user with values in the user profile.

To use linked queries, you must configure a `ReferenceMemoryMappedType` field that occurs in the documents that you want to query, which contains values that occur in the document references for the connected documents. `ReferenceMemoryMappedType` fields provide a fast lookup between child documents and a parent document. For example, for a user profile, this might be a `UserName` field in the documents that specifies the user that wrote a document, and the value of that field is the document reference for the user profile.

Configure the IDOL Content component for Linked Queries

To use link queries, you must configure a `ReferenceMemoryMappedType` field. You then index the documents. You must either ensure that you index the parent documents first (in which the ReferenceMemoryMapped values occur in the document reference), or use the `DREREGENERATE` index action to fix the parent-child ordering.

To configure the IDOL Content component for linked queries

1. Open the IDX file of the documents that you want to index.

2. Find the field that you want to use to link documents. This field must contain a value that occurs in a normal field in the documents that you want to search, and in a reference field in the parent documents. For example, it might be a `UserName` field in the documents, and the value of this field is the document reference for the user profile.

3. Open the IDOL Content component configuration file in a text editor.

4. In the `[FieldProcessing]` section, go to the end of the list of field processing sections and add the next number. Set this parameter to the name of the field process that you want to create. For example:

```
[FieldProcessing]
...
10=SetNumericDateFields
11=SetReferenceMemoryMappedFields
```

5. Create a section for the field process. Create a property for the process (you define the property later by setting one or more applicable configuration parameters). Identify the fields that you want to associate with the process.

> **NOTE:** The properties that you create must not have the same name as the processes.

For example:

```
[SetReferenceMemoryMappedFields]
Property=ReferenceMemoryMappedFields
PropertyFieldCSVs=*/UserName,*/Author
```

6. Create a section for the new property, in which you set the `ReferenceMemoryMappedType` property to **True**. For example:

```
[ReferenceMemoryMappedFields]
ReferenceMemoryMappedType=True
```

7. Save and close the configuration file.

8. Restart the IDOL Content component for your changes to take effect.

9. Index the parent documents, and then index the rest of the documents. Alternatively, you can index all the documents and then send a `DREREGENERATE` index action to fix the parent-child relationships. For example:

```
http://localhost:9001/DREREGENERATE?Type=ReferenceMemoryMapped
```

Related Topics

-

Send Linked Queries

Linked queries rely on the `LINK` field specifier, and the `LinkFieldText` action parameter. You can use these options in the `Query`, `Suggest`, `SuggestOnText`, and `GetQueryTagValues` actions.

To create a linked query, set the following parameters:

`Text`	The query text that you want to find in the documents.
`FieldText`	A `LINK` field expression, setting the `ReferenceMemoryMappedType` field that connects the results documents to the parent documents that you want to link to. For example: `LINK{}:UserName` You can also add any other `FieldText` expressions that you want to use to restrict the results documents.

LinkFieldText	A `FieldText` expression that you want to apply in the parent documents (the documents in which the `ReferenceMemoryMapped` values occur in a `ReferenceType` field).
	The `LinkFieldText` expression must be a simple `FieldText` expression, that is, it must use either all AND or all OR Boolean expressions. The fields that you use in the expressions must be `MatchType`, `NumericDateType`, `NumericType`, or `ParametricType` (when `ParametricNumericMapping` is enabled).
	You cannot use metadata fields in the `LinkFieldText` expression, including the IDOL default metadata fields, and fields created by using the `UserMetaFields` parameter.

Example

A `Users` database has a set of user profile documents that use the user name as the document reference, and list interests and other user information. For example:

```
<DOCUMENT>
    <DREREFERENCE>User01</DREREFERENCE>
    <INTEREST>Soccer</INTEREST>
    <COUNTRY>USA</COUNTRY>
    <JOINED>01/01/2012</JOINED>
</DOCUMENT>
<DOCUMENT>
    <DREREFERENCE>User02</DREREFERENCE>
    <INTEREST>Food</INTEREST>
    <COUNTRY>UK</COUNTRY>
    <JOINED>02/06/2011</JOINED>
</DOCUMENT>
```

The same index has a `SocialMedia` database, which contains posts by these users. For example:

```
<DOCUMENT>
    <DREREFERENCE>SM0001</DREREFERENCE>
    <USERNAME>User01</USERNAME>
    <DRECONTENT>Just got my tickets for the final in Brazil!</DRECONTENT>
</DOCUMENT>
<DOCUMENT>
    <DREREFERENCE>SM0002</DREREFERENCE>
    <USERNAME>User02</USERNAME>
    <DRECONTENT>Brazil nuts are so moreish...</DRECONTENT>
</DOCUMENT>
```

The following standard query returns both of these documents:

```
action=Query&Text=Brazil
```

You can use a linked query to specify additional restrictions. The following query finds posts about Brazil by users who are interested in soccer:

```
action=Query&Text=Brazil&FieldText=LINK{}:USERNAME&LinkFieldText=MATCH
{Soccer}:INTEREST
```

This returns the document `SM0001`, but not the document `SM0002`.

This query requires the following field configuration:

- `USERNAME` is configured as `ReferenceMemoryMappedType`.

- `INTEREST` is configured as `MatchType`.

Combine Different Search Types

This section describes how you can combine search types.

Synonym and Boolean Searches

If you combine a synonym search with a Boolean search, Content obeys the Boolean rules while it processes the synonym query. For example:

```
action=Query&Text=cat+AND+dog&Synonym=True
```

If the synonym file contains synonyms for the terms *cat* and *dog*, Content returns only documents that contain both the term `cat` or one of the synonyms for *cat*, and the term *dog* or one of the synonyms for *dog*. For example, documents that contain *mouser* and *cur*, *cat* and *man's best friend*, and so on.

Synonym Search and Field Restrictions

You can use field restrictions within a synonym search to return only documents that contain the synonym in a specific field. For example:

```
action=Query&Text=cat:Title&Synonym=True
```

If the synonym file lists synonyms for the term *cat*, Content returns only documents that contain the term *cat* or one of the synonyms for *cat* in their `Title` fields.

Soundex and Proper Names Searches

You can send a proper name search to Content, which includes a `SOUNDEX` keyword search. However, because the `SOUNDEX` keywords are spelled phonetically, Content cannot match proper names for them.

Soundex and Boolean Searches

If you combine a Soundex keyword search with a Boolean search, Content obeys the Boolean rules while it processes the Soundex keyword search. For example:

```
action=Query&Text=Munich+AND+SOUNDEX(einstine)
```

This query returns only documents that contain both the term *Munich* and a term that phonetically matches *einstine* (for example, *Einstein*).

Soundex and Proximity Searches

If you combine a Soundex keyword search with a proximity search, Content obeys the proximity rules while it processes the Soundex keyword search. For example:

```
action=Query&Text=Munich+NEAR2+SOUNDEX(einstine)
```

This query returns only documents in which the term *Munich* is no more than two terms away from a term that phonetically matches *einstine* (for example, *Einstein*).

Soundex Search and Field Restrictions

You can use field restrictions to restrict a Soundex keyword search to specific fields. For example:

```
action=Query&Text=SOUNDEX(einstine:Name)
```

This query returns only documents that contain a term that phonetically matches *einstine* (for example, *Einstein*) in their Name fields.

Exact Phrase and Boolean Searches

If you combine an exact phrase search for multiple phrases (in which the individual phrases are framed by quotation marks) with a Boolean search, Content obeys the Boolean rules while it processes the exact phrase search. To ensure that Boolean operators tie to an entire phrase, enclose the phrase in brackets. For example:

```
action=Query&Text=("Egyptian cats")+AND+("Phoenician sailors")
```

This query returns only documents that match the phrase *Egyptian cats*, as well as the phrase *Phoenician sailors* (after stemming and stopping).

```
action=Query&Text=("Egyptian cats")+NOT+("Phoenician sailors")
```

This query returns only documents that match the phrase *Egyptian cats*, but not the phrase *Phoenician sailors* (after stemming and stopping).

Exact Phrase and Proximity Searches

If you combine an exact phrase search for multiple phrases (in which you frame the individual phrases in quotation marks) with a proximity search, Content obeys the proximity rules while it processes the exact phrase search. To ensure that proximity operators tie to an entire phrase, enclose the phrases in brackets. For example:

```
action=Query&Text=("Egyptian cats")+ NEAR2+("Phoenician sailors")
```

This query returns only documents in which a match of the phrase *Egyptian cats* is no further than two words away from a match of the phrase *Phoenician sailors* (after stemming and stopping).

```
action=Query&Text=("Egyptian cats")+BEFORE+("Phoenician sailors")
```

This query returns only documents that match the phrase *Egyptian cats* and the phrase *Phoenician sailors* (after stemming and stopping). The phrase *Egyptian cats* must occur before the phrase *Phoenician sailors*.

Exact Phrase Search and Field Restrictions

You can use field restrictions to restrict an exact phrase search to specific fields. For example:

```
action=Query&Text="birds of prey" "bird watching":DRETITLE
```

This query returns documents that contain a match for *birds of prey* in any field or a match for *bird watching* in the DRETITLE field (after stemming and stopping).

Boolean and Proximity Searches

Boolean and proximity operators have the following precedence:

```
Highest:      NOT
                NEAR; DNEAR; XNEAR; YNEAR
                AND; BEFORE; AFTER; WHEN; SENTENCE; DSENTENCE; PARAGRAPH
Lowest:       OR; WNEAR; EOR
```

Operators that have the same level of precedence have neither left or right associativity. You can use brackets to bind terms together as appropriate. Proximity operators must have terms on either side, and cannot be adjacent to brackets.

Boolean Search and Field Restrictions

If you combine a field restriction keyword search with a Boolean search, Content obeys the Boolean rules while returning only documents that contain the specified values in the specified fields. For example:

```
action=Query&Text=cat:Animal AND dog:Animal:Fauna
```

This query returns only documents that contain the term *cat* in their Animal fields and the term *dog* in their Animal or Fauna fields.

Proximity Searches and Field Restrictions

If you combine a field restriction keyword search with a proximity search, Content obeys the proximity rules while returning only documents that contain the specified values in the specified fields.

For example:

```
action=Query&Text=cat NEAR2 dog:Animal
```

This query returns only documents that contain the term *cat* and term *dog* in their `Animal` fields. The terms must be no more than two terms away from each other.

Wildcards in Queries

Use Wildcard matching sparingly, because it slows the IDOL Content component performance.

You can use the following Wildcard characters in query `Text` and `FieldText` queries:

?	to match one character.
*	to match zero, one, or more characters.

You can use the `UnstemmedMinDocOccs` parameter in the `[Server]` section of the configuration file to specify the number of documents in which a term must occur to consider it in a Wildcard search.

> **NOTE:** To use Wildcards to search for numbers, set `SplitNumbers` (described in the `[Server]` configuration section of the *IDOL Server Reference*) to **False**.

Wildcards in Query Text

You can use Wildcards in the `Text` parameter of the `Query` action.

You can use the `Text` action parameter to specify field restrictions for result documents, as long as the fields that you restrict results to are Index fields in the IDOL Content component. If you match fields, you can use Wildcards and match multiple fields simultaneously by separating them with colons.

Related Topics

- Set up the Field Index Process, on page 43

Example Wildcard Queries

The following examples demonstrate how to use Wildcards in query text.

Example 1

```
action=Query&Text=Mi?rotech
```

This query returns documents that contain the term *Mikrotech* or *Microtech*.

Example 2

```
action=Query&Text="Co*ins":Name:Author+Arm?dale:Title
```

This query returns documents that contain a `Name` or `Author` field whose value matches the Wildcard string `Co*ins` (for example, *Collins*) and documents that contain a `Title` field whose value matches the Wildcard string `Arm?dale` (for example *Armadale*).

Example 3

Consider the following scenarios:

A. The IDOL Content component has two indexed documents. One document contains the word *rollerskating*, and the other contains the word *rollerskaters* (both of which stem to *ROLLERSK*).

B. The IDOL Content component has only one indexed document. It contains the word *rollerskater* (which also stems to *ROLLERSK*).

- Send the following query:

  `action=Query&Text=rollersk*`

  This query returns all documents that contain any terms that stem to *ROLLERSK*. It returns both documents in situation A and the one document in situation B.

- Send the following query:

  `action=Query&Text=rollerskatin*`

  A Wildcard query returns documents only if the Content index contains one or more matches to the Wildcard value. If the term matches, it is then stemmed, and all occurrences of the stem (in any document) are also considered matches.

  Therefore, this query returns both documents in situation A (because `rollerskatin*` matches *rollerskating*, and because it stems to *ROLLERSK*, which has matches in both documents). It does not return the document in situation B (because `rollerskatin*` does not match *rollerskater*).

- Send the following query when the `AdvancedSearch` configuration parameter is set to **True**:

  `action=Query&Text="rollerskatin*"`

  This query returns documents that contain the Wildcard term. However, because `AdvancedSearch` is enabled *and* the Wildcard term is in quotation marks, this query does not return any further terms that match the stem of the expanded Wildcard term.

  Therefore, with this query, only the *rollerskating* document returns in situation A, and no document returns in situation B.

Wildcards in FieldText Queries

If you use a `Query`, `Suggest`, or `SuggestOnText` action to send a FieldText query to the IDOL Content component (by using the `FieldText` action parameter), you can use Wildcards to match a single string or to match multiple strings.

> **NOTE:** To identify the fields, use the format `/FieldName` to match root-level fields, `*/FieldName` to match all fields except root-level, or `/Path/FieldName` to match fields that the specified path points to.
>
> All string matching is case insensitive.

Matches for One or More Strings

The field specifier `WILD` allows you to Wildcard match:

- one or more strings
- one or more strings that consist of several words
- one or more strings that contain punctuation
- one or more strings that consist of several words and punctuation

Strings can contain punctuation (except braces), which means that if you want to match a string that contains HTML with IDOL content, percent-encode the HTML to avoid confusion with ampersands (&) and so on.

To match a string that contains a comma, percent-encode the comma as `%2C`, otherwise Content reads it as a separator.

You can match multiple fields simultaneously by separating them with colons.

Examples:

`action=Query&FieldText=WILD{wom?n }:Clothes`

The `Clothes` field must contain a word that matches the specified Wildcard string (for example, *woman* or *women*) for the document to return as a result.

`action=Query&FieldText=WILD{of mice and m?n}:Title:Book`

The `Title` or `Book` field must contain a string that matches the specified Wildcard string (for example, *of mice and man* or *of mice and men*) for the document to return as a result.

`action=Query&FieldText=WILD{Glory is fleeting\, but * is forever}:QuotesNapoleon`

The `QuotesNapoleon` field must contain a string that matches the specified Wildcard string (for example, *Glory is fleeting, but obscurity is forever*) for the document to return as a result.

`action=Query&FieldText=WILD{*.html,*.htm}:URL`

The `URL` field value must end with *.html* or *.htm* for the document to return as a result.

`action=Query&FieldText=WILD{passi*incarnata}:Climbers`

The `Climbers` field must contain a phrase that begins with *passi* and ends with *incarnata* (for example, *passionflower incarnata* or *passiflora incarnata*) for the document to return as a result.

`action=Query&FieldText=WILD{ passi*incarnata,passi*alata*}:Climbers`

The `Climbers` field must contain a string that matches one of the specified Wildcard strings (for example, *passionflower incarnata*, *passiflora incarnata*, *passionflower alata*, *passiflora alata*, *passionflower alata shannon*, or *passiflora alata shannon*) for the document to return as a result.

`action=Query&FieldText=WILD{*www.example.com*.txt,*www.example.com*.pdf}:PATH:URL`

The `PATH` or `URL` field must contain a path that contains *www.example.com* and ends with *.txt* or *.pdf* (for example, *http://www.example.com/files/doc.txt* or *http://www.example.com/fields/technicalbrief.pdf*) for this document to return as a result.

Wildcard Searches in Japanese, Chinese, Korean, and Thai

Asian languages do not include spaces or word boundaries. For this reason the IDOL Content component applies *sentence breaking* to Asian text when it processes it, to split the text into individual words or *terms*.

You can carry out Wildcard searches in Japanese, Chinese, Korean, and Thai, provided that you query the IDOL Content component with one or more terms rather than a single string in which words are not delimited by spaces.

The question mark (?) Wildcard might not behave as expected, because it represents a single character, and each Asian *letter* actually consists of multiple characters (usually two). For example, if you want to use a ? single-character Wildcard in a multibyte language query, you must use one ? character for each byte (for example, *???* for a single Japanese character).

Query for Nonalphanumeric Characters

Although the IDOL Content component does not store nonalphanumeric characters, you can query for them using FieldText queries. To speed up the processing time of the query, combine the FieldText queries with an ordinary text query, by using the `Text` parameter and the `FieldText` parameter as follows.

Text

Specify your entire query string including the nonalphanumeric characters. If you include any special characters, you must treat them appropriately, as shown in the following table.

Treat…	Like this:
&	Percent-encode ampersands that are part of the query string.
~ [ ] * ? : ( ) "	If you have not changed the IDOL Content component default behavior, replace the character with a space. If you configured Content not to treat any of these characters as a separator by specifying them in the `DiminishSeparator` configuration parameter, remove the character from the search string. Alternatively, you can instruct Content to interpret the following characters as normal characters in query syntax by setting the `IgnoreSpecials` action parameter to **True** for the `Query` or `GetQueryTagValues` action: *?":() and Boolean or proximity operators, for example AND, NOT, OR, NEAR, and so on. This disables Wildcard searching, phrase queries, field restrictions, and Boolean operations.

FieldText

You must percent-encode all strings in `FieldText`.

You must distinguish punctuation in IDOL query syntax (such as commas and braces) from the same punctuation that might occur in strings. Use the following percent-encoding guidelines:

- percent-encode an ampersand (&) with `%26`

- percent-encode a backslash (\) with `%5C`

- percent-encode a comma (,) with `%2C`

- percent-encode a percentage sign (%) with `%25`

- To distinguish between braces in strings from query syntax braces, percent-encode left ({) and right (}) braces twice, with `%257B` and `%257D`.

- To distinguish between commas in strings from separator commas, percent-encode commas within strings twice (`%252C`).

- Leave commas that separate multiple items, and braces that start and end lists, as regular punctuation. There must be no spaces before or after these commas.

Use the `STRING` field specifier to search for your entire query string, including the nonalphanumeric characters, in the appropriate field in the IDOL Content component.

Examples

To search for *Auto**:

```
action=Query&Text=Auto&FieldText=STRING{Auto*}:DRECONTENT
```

To search for *yahoo!*:

```
action=Query&Text=yahoo!&FieldText=STRING{yahoo!}:DRECONTENT
```

To search for *AT&T*:

```
action=Query&Text=AT%26T&FieldText=STRING{AT%26T}:DRECONTENT
```

To search for "*r-t*"

```
action=Query&Text=r-t&FieldText=STRING{r-t}:DRECONTENT
```

To search for "*eat, drink and be merry*"

```
action=Query&Text=eat, drink and be merry&FieldText=STRING{eat%2c drink and be
merry}:DRECONTENT
```

To search for "*politics [and] their effects*"

```
action=Query&Text=politics and their effects&FieldText=STRING{politics [and] their
effects}:DRECONTENT
```

To search for "**NSYNC*"

```
action=Query&Text=*NSYNC&IgnoreSpecials=True
```

Optimize Retrieval of Tagged Documents

To include one or more attributes that are specific to a document, you can create fields in the document when you store it in the IDOL Content component. When you send a query to Content, you can then use any of the fields that you created to restrict which documents Content returns.

For example, you can create fields that contain authors, categories, folders, product types, or any other attribute. You can then restrict a query to return only documents that were, for example, written by a specific author or belong to one or more specific categories.

When you restrict a query in this way, its processing time might increase. To keep the processing time to a minimum, use the appropriate query syntax for the fields that you create.

Query Syntaxes

Query processing time depends on the syntax that the query uses. Different syntaxes are available, and some require you to create fields in a specific way.

Fastest

Syntax:	action=Query&Text=text&FieldText=EQUAL {*numericalAttribute*}:*fieldName*
Requirements:	- create a numeric field for each attribute - store these fields as numeric fields in the IDOL Content component

Example:	Four attributes are available to indicate which categories a document belongs to:
	
	England, France, Germany, USA
	
	It uses numbers to indicate which attributes apply to a document:
	
	1—England, 2—France, 3—Germany, 4—USA
	
	In documents, you create a numeric field for each category that they belong to. For example, if a document belongs to the categories *France* and *USA*:
	
	`#DREFIELD Cat1="2"`
	`#DREFIELD Cat2="4"`
	
	The following query returns documents that match the specified `Text` and contain the value *4* in one of their `Cat` fields (for example, `Cat1`). The value *4* indicates that the documents belong to the *USA* category.
	
	`action=Query&Text=presidential election&FieldText=EQUAL{4}:Cat*`

Syntax:	`action=Query&Text=text&FieldText=MATCH{attribute}:fieldName`
Requirement s:	• create a field for each attribute

Example:	Four attributes are available to indicate which categories a document belongs to: England, France, Germany, USA
	
	In documents, you create a field for each category that they belong to. For example, if a document belongs to the categories *France* and *USA*:
	
	`#DREFIELD Cat1="France"`
	`#DREFIELD Cat2="USA"`
	
	The following query returns documents that match the specified `Text` and contain the value *France* in one of their `Cat` fields (for example, `Cat1`).
	
	`action=Query&Text=presidential election&FieldText=MATCH`
	`{France}:Cat*`

Syntax:	`action=Query&Text=text&FieldText=BITAND` `{numericalAttribute}:fieldName`
Requirement s:	• assign each attribute a *bit* • create a numeric field for all attributes (if you have more than 32 bits, you need more fields) • store this field as a numeric field in the IDOL Content component

Example:	Four attributes are available to indicate which categories a document belongs to: England, France, Germany, USA
	
	Assign a bit value to each attribute:
	
	1—France, 2—England, 4—Germany, 8—USA
	
	In documents, you create a field for the categories that they belong to. For example, if a document belongs to the categories *France* and *USA*:
	
	`#DREFIELD Cat="9"`
	
	The following query returns documents that match the specified `Text` and contain the value *10* in its `Cat` field. The value *10* indicates that documents must belong to the *England* or the *USA* category.
	
	`action=Query&Text=presidential election&FieldText=BITAND{10}:Cat`

Slowest

Syntax:	`action=Query&Text=text&FieldText=STRING{attribute}:fieldName`

Requirements:	• create a field that contains a comma-separated list of the document attributes
Example:	Four attributes are available to indicate which categories a document belongs to: England, France, Germany, USA In documents, you create a field that contains a comma-separated list of the categories that the documents belong to. For example, if a document belongs to the categories *France* and *USA*: `#DREFIELD Cat="France,USA"` The following query returns documents that match the specified `Text` and contain the value *France* in their `Cat` field. `action=Query&Text=presidential election&FieldText=STRING {France}:Cat`

Chapter 15: Customize Results

You can change the IDOL Content component results display from its default state. You can set the number of results to display, batch results, change the order in which to display results, set additional fields to display, apply templates, or cluster results.

You can also choose to generate document summaries, display hyperlinks, display query summaries, and suggest alternate spellings for query terms.

Change the Results Display

You can control the number of results to display in a results list, and the sort order of the returned documents.

Set the Number of Results to Display

When you perform a query, by default the IDOL Content component returns a maximum of six results. You can modify the number of results that Content returns by adding the `MaxResults` parameter to your query and setting it to the number of results to return. For example:

```
http://IDOLhost:port/action=Query&Text=Harry Potter and the Goblet of
Fire&MaxResults=10
```

In this example, Content returns 10 results for the query, if at least 10 results are available.

Change Result Sorting (Display Order)

By default, the IDOL Content component lists query results in order of relevance. To weight and rank the documents it returns by statistical relevance, Content uses complex algorithms that use a combination of information theory and Bayesian methods. It makes use of information theoretic values

calculated dynamically for all concepts on indexing, which allows it to evaluate relevance both as a percentage, and in the case of agents, as absolute values.

In practice, the relevance acts as a measure of the conceptual overlap between the query text and the text within a document. You can affect the relevance in several ways. You can apply extra weight to certain fields by associating a weighting factor with them at indexing time. For example, you can give extra weight when query terms appear in the document title as opposed to the body of the text.

You can change the order in which Content returns query results by adding the `Sort` parameter to your `Query`, `Suggest`, `SuggestOnText`, `GetTagValues`, or `GetQueryTagValues` action.

Sort Options for Query, Suggest, and SuggestOnText

The following table lists the `Sort` options that are available for the `Query`, `Suggest`, and `SuggestOnText` actions.

`Off`	Displays results unsorted.
`AutnRank`	Displays results in order of the value in their `AutnRankType` field. This method lists the document with the highest `AutnRankType` field value first.
`Cluster`	Displays results in order of cluster (in decreasing cluster ID order), if you also add `Cluster=True` to the action. If you specify `Cluster` as one of multiple sorting options, it automatically takes precedence over the other sorting methods, even if you did not put it in first place.
`Database`	Displays results in order of database number (in increasing order). Define the database numbers in the IDOL Content component configuration file.
`Date`	Displays results in order of their date (the date contained in the `DateType` fields). This method lists the most recent document first. If several documents have the same date, their display order is determined by their `autn:docid` (document ID) number (the highest `autn:docid` is listed first).

`Distcartesian`	Displays results according to their distance from a specified point using Cartesian coordinates (X/Y). The option has this format:
	
	 `sort=Distcartesian{coordX,coordY}:POSITION`
	
	 where:
	
	 • *coordX* is the specified x coordinate.
	
	 • *coordY* is the specified y coordinate.
	
	 • *POSITION* is either:
	
	     ◦ a single document field that contains the position information in Well-known text format POINT definitions. This field must be a unified `GeospatialType` field.
	
	     ◦ two fields, in the format *X*:*Y*, where *X* is the document field that contains the X coordinates, and *Y* is the document field that contains

	the Y coordinates. These fields must be either `GeospatialType` or `NumericType` fields. You must specify the fields in the order X:Y. You must specify two fields in the order `X:Y`.

Distspherical	Displays results according to their distance from a specified point using spherical coordinates (latitude and longitude). The option has this format:
	
	`sort=Distspherical{`*lat,long*`}:`*LOCATION*
	
	where:
	
	• *Lat* is the specified latitude. Specify latitude positions south of the equator as negative.
	
	• *Long* is the specified longitude. Specify longitude positions west of the Greenwich Meridian as negative.
	
	• *LOCATION* is either:
	
	   ○ a single document field that contains the location information in Well-known text format POINT definitions. This field must be a unified `GeospatialType` field.
	
	   ○ two fields, in the format *Latfield*`:`*Longfield*, where *Latfield* is the document field that contains the latitude, and *Longfield* is the document field that contains the longitude. These fields must be either `GeospatialType` or `NumericType` fields. You must specify the fields in the order `latitude:longitude`.

DocIDDecreasing	Displays results in order of their `autn:docid` (document ID) number. This method lists the document with the highest `autn:docid` first.
DocIDIncreasing	Displays results in order of their `autn:docid` (document ID) number. This method lists the document with the lowest `autn:docid` first.
fieldName `:`*sortMethod*	Displays results in the order specified by *sortMethod*, based on the value of the Content field *fieldName*. This option is optimized if *fieldName* is a `MatchType` field. The following sort methods are available: • **alphabetical**. Determines the display order of results by the string contained in *fieldName*. This method lists results in alphabetical order. (Sorting is faster if *fieldName* is `MatchType`.) • **decreasing**. Determines the display order of results by the number or string contained in *fieldName*. This method lists the result with the highest number or the result that is last in alphabetical order first. For `NumericType` fields, this method is equivalent to `numberdecreasing`. For `MatchType` fields, this method is equivalent to `reversealphabetical`. • **increasing**. Determines the display order of results by the number or string contained in *fieldName*. This method lists the result with the lowest number or the result that is first in alphabetical order first. For `NumericType` fields, this method is equivalent to `numberincreasing`. For `MatchType` fields, this method is equivalent to `alphabetical`.

	• **numberdecreasing**. Determines the display order of results by the number contained in *fieldName*. This method lists the result with the highest number first. (Sorting is faster if *fieldName* is NumericType.) • **numberincreasing**. Determines the display order of results by the number contained in *fieldName*. This method lists the result with the lowest number first. (Sorting is faster if *fieldName* is NumericType.) • **reversealphabetical**. Determines the display order of results by the string contained in *fieldName*. This method lists results in reverse alphabetical order. (Sorting is faster if *fieldName* is MatchType.)
Random	Displays results in random order.
Relevance	Displays results in order of their relevance. This method lists the document with the highest relevance first. If documents have the same relevance, this method determines their display order by their autn:docid (document ID) number, and lists the highest autn:docid first.
ReverseDate	Displays results in order of their date (the date contained in the DateType fields). It lists the oldest document first. If several documents have the same date, this method determines their display order by their autn:docid (document ID) number, and lists the highest autn:docid first.

If you want to sort results by several criteria, you can specify them as follows:

sortOption1+sortOption2+...

Example 1:

`http://MyHost:20000/action=Query&Text=presidential elections&Sort=Date`

In this example, Content displays results in order of the document date.

Example 2:

`http://MyHost:20000/action=Query&Text=presidential elections&Sort=DRETITLE:reversealphabetical`

In this example, Content displays results in reverse alphabetical order by their DRETITLE.

Example 3:

`http://MyHost:20000/action=Query&Text=presidential elections&Sort=Relevance+DRETITLE:alphabetical+Date`

In this example, results order by Relevance, then by their DRETITLE, and then by their Date.

Sort for GetTagValues and GetQueryTagValues

The following table lists the Sort options that are available for the GetTagValues and GetQueryTagValues actions.

Off	Displays results unsorted.
Alphabetical	Determines the display order of results by the string contained in the query field. Displays results in alphabetical order.
Date	Determines the display order of results by the date contained in the query field. This method lists the most recent result first.
NumberDecreasing	Determines the display order of results by the number contained in the query field. This method lists the result with the highest number first.
NumberIncreasing	Determines the display order of results by the number contained in the query field. This method lists the result with the lowest number first.
ReverseAlphabetical	Determines the display order of results by the string contained in the query field. This method displays results in reverse alphabetical order.
ReverseDate	Determines the display order of results by the date contained in the query field. This method lists the oldest result first.

For the GetQueryTagValues action, these Sort options are also available:

DocumentCount	If you set the DocumentCount action parameter to **True**, you can use this option to display results in order of their document count (in decreasing order).
ReverseDocumentCount	If you set the DocumentCount action parameter to **True**, you can use this option to display results in the reverse order of their document count (in increasing order).

For example:

```
http://MyHost:20000/action=GetQueryTagValues&FieldName=GRAPE,COUNTRY &Text=A smooth
red wine that complements game&Sort=Alphabetical
```

In this example, Content displays results in alphabetical order.

> **TIP:** The GetQueryTagValues action also has a QuerySort parameter, which determines how it sorts the results of the associated query when it retrieves values.
>
> For example, if you set MaxResults in GetQueryTagValues, the action retrieves parametric values from the first *N* results, according to the result sorting method. You can use QuerySort to specify how to sort these results. The options for QuerySort are the same as for the Query action Sort parameter (see Sort Options for Query, Suggest, and SuggestOnText, on page 316).

Batch (Page) Results

If you are running a Query, Suggest, or SuggestOnText action, you can use the MaxResults parameter with the Start parameter to display only MaxResults number of results, from the Start position onwards.

The `Start` value must be smaller than the `MaxResults` value because if the position it specifies does not fall within the generated range of results, no results return for the query, even though results might be available.

Example 1:

```
http://IDOLhost:port/action=Query&Text=Harry Potter and the Goblet of
Fire&MaxResults=6&Start=6
```

In this example, if a total of 10 results is available for the query, only the sixth result returns for the query.

Example 2:

```
http://IDOLhost:port/action=Query&Text=Harry Potter and the Goblet of
Fire&MaxResults=100&Start=6
```

In this example, if a total of 10 results is available for the query, results 6 through 10 (inclusive) return for the query.

Example 3:

```
http://IDOLhost:port/action=Query&Text=Harry Potter and the Goblet of
Fire&MaxResults=5&Start=6
```

In this example, no results return for the query (even if results are available).

Change the Field Display

To minimize query processing time, by default the IDOL Content component returns only a subset of fields for each query result. You can display additional fields (both metafields and document fields).

Returned Fields

By default, the IDOL Content component returns the following fields for each result:

`<autn:reference>`	The reference of the result document.
`<autn:id>`	The ID of the result document.
`<autn:section>`	The section of the document.
`<autn:weight>`	The relevance weight of the result document.
`<autn:links>`	The terms in the query text that match in the result document.
`<autn:database>`	The database that contains the result document.
`<autn:title>`	The title of the result document.

You can modify the set of fields that Content returns, as described in the following section.

Display Additional Metafields

You can display all metafields that are available for results by adding `XMLMeta=True` to your query. For example:

`http://MyHost:20000/action=Query&Text=Choreographic techniques&XMLMeta=True`

This action instructs Content to return all available metafields for documents that match the text specified by the `Text` parameter.

Related Topics

- Metadata Fields, on page 105

Display Document Fields

You can configure the IDOL Content component to display specific document fields whenever it returns query results. Alternatively, you can specify the document fields that Content returns when you perform an individual query.

Configure IDOL Server to Always Display Specific Fields

To display specific document fields every time Content returns query results, create a field process in the IDOL Content component configuration file that identifies the document fields that you want to display. After you create this field process, the fields that you identified always display when Content returns query results (provided that you set the query action `Print` parameter to **Fields**, which is its default value).

To configure the IDOL Content component to always display specific fields

1. Open the IDOL Content component configuration file in a text editor.

2. In the `[FieldProcessing]` section, list a new field process whose name indicates that the process identifies which fields are displayed.

   For example:

   ```
   [FieldProcessing]
   0=MyFirstProcess
   1=PrintFields
   ```

3. Create a section for the new field process that you listed, in which you create a property for the process (you define the property later by setting one or more applicable configuration parameters). Use the `PropertyFieldCSVs` parameter to identify the fields that you want to display.

   > **NOTE:** The property that you create must not have the same name as the process.

   For example:

```
[PrintFields]
Property=Print
PropertyFieldCSVs=*/SUMMARY,*/DRECONTENT
```

4. Create a section for the property in which you set the `PrintType` parameter to **True**. This displays the associated `PropertyFieldCSVs` fields for query results. For example:

```
[Print]
PrintType=True
```

5. Save and close the configuration file.

6. Restart the IDOL Content component for your changes to take effect.

Every time you perform a `Query`, `Suggest`, `SuggestOnText`, or `GetContent` query, IDOL Server now returns `SUMMARY` and `DRECONTENT` field of the result documents, in addition to the metafields that it displays by default.

Display Specific Fields for Individual Queries

To return specific document fields for individual queries, use the `Print` or `PrintFields` action parameters when you perform a `Query`, `Suggest`, `SuggestOnText`, or `GetContent` query action.

- `Print`. Use one of the following options to determine the types of fields to display:

`All`	All XML fields. **NOTE:** This option does not display temporary fields that you create in a `Query` action by using the `UserMetaFields` parameter. To display those fields, you must list them in the `PrintFields` parameter.
`AllSections`	All fields, and all sections in section breaking fields.
`Date`	Fields with the `DateType` property.
`Fields`	Fields with the `PrintType` property.
`Index`	Fields with the `Index` property.
`IndexText`	The contents of fields with the `Index` property.
`NegativeFields`	All fields, except those listed in the `PrintFields` action parameter.
`None`	The first reference field only.
`NoResults`	No results or fields. The query returns only the number of hits.
`Parametric`	Fields with the `ParametricType` property.
`Reference`	Fields with the `ReferenceType` property.
`Source`	Fields with the `SourceType` property.

For example:

```
http://12.3.4.56:4000/action=Query&Text=Hogwarts school of witchcraft and
wizardry&Print=Index
```

This query returns results that are conceptually similar to the specified query text. Each result returns with fields that were set up as `Index` fields in the IDOL Content component configuration file.

- `PrintFields`. A list of fields that you want to display in the results.

  If you also set `Print`, `Print` generally takes precedence. However:

  - when you set `Print` to **Fields**, only the specified `PrintFields` are displayed.

  - when you set `Print` to **NegativeFields**, the specified `PrintFields` are not displayed.

  For example:

  ```
  http://12.3.4.56:4000/action=Query&Text=Hogwarts school of witchcraft and
  wizardry&PrintFields=Author,Title
  ```

  This query returns results that are conceptually similar to the specifid query text. Each result returns with its `Author` and `Title` fields.

Related Topics

- [Display Online Help, on page 30](#)

Use XSLT Templates to Change Output Format

You can use XLST templates to format the results that your IDOL components return. You can write your own XSLT templates or use the templates that the IDOL Server installation includes in the `acitemplates` directory:

- `QueryForm.tmpl`. You can apply this template to the `Query` action to produce a simple HTML query interface from which you can query the IDOL Content component.

  The default version of this template has the following appearance.

- `GetStatusForm.tmpl`. You can apply this template to the `GetStatus` action to generate HTML formatted status information.

  The default version of this template has the following appearance.

Enable the XSLT Templates

Use the following procedure to enable the XSLT templates in an IDOL component.

To enable the XSL templates

1. Open the IDOL component configuration file in a text editor.

2. Find the `[Server]` section.

3. Set the `XSLTemplates` setting to **True**. (If the `[Server]` section does not contain this setting, add it.)

4. Save and close the configuration file.

5. Restart the component for your changes to take effect.

Apply XSLT Templates to Actions

To apply a template to action output, add the following parameters to the action:

`Template`	Specify the name of the template that you want to use to format action output.
`OutputEncoding`	Set this parameter to **UTF8**.

Example 1:

`action=GetStatus&Template=QueryForm&OutputEncoding=UTF8`

In this example, the `QueryForm.tmpl` XSLT template is applied to a `GetStatus` action.

Example 2:

`action=GetStatus&Template=GetStatusForm&OutputEncoding=UTF8`

In this example, the `GetStatusForm.tmpl` XSLT template is applied to a `GetStatus` action.

> **NOTE:** If the action returns an error response, the IDOL component does not apply the XSLT template to the response.

Generate Summaries

You can specify several types of document summaries to return with query results, or you can directly generate summaries for arbitrary documents or blocks of text.

Types of Summaries

The IDOL Content component can automatically generate one of these summary types for the results it produces. It generates all summaries in real time.

- **Concept**. A conceptual summary of each result document. A concept summary contains sentences that are typical of the result content (these sentences can be from different parts of the result document).

- **Context**. A conceptual summary of each result document, biased by any terms in the query `Text` and `FieldText` parameters. A context summary contains sentences that are particularly relevant to the terms in the query. These sentences can be from different parts of the result document.

  > **NOTE:** To ensure that the summary contains the query terms, use the `Characters` action parameter, rather than `Sentences`. You can also use the `MaxSourceCharacters` configuration parameter to use more than the first 10,000 characters of a document to generate summaries.

- **Quick**. A brief summary of each result document. A quick summary contains the first few sentences of the result document.

- **ParagraphConcept**. A conceptual summary of each result document. The summary contains the paragraphs that are most typical of the content of the result document. These paragraphs can be from different parts of the result document.

- **ParagraphContext**. A conceptual paragraph summary of each result document, biased by any terms in the query `Text` and `FieldText` parameters. This summary contains paragraphs that are particularly relevant to the terms in the query.

Return Summaries with Query Results

Use the following procedure to return summaries with query results.

To generate summaries for query results

1. Send a `Query`, `Suggest`, or `SuggestOnText` action to the IDOL Content component that includes the `Summary` parameter. Set the `Summary` parameter to the type of summary that you want to return for results (**Concept**, **Context**, **Quick**, **ParagraphConcept**, or **ParagraphContext**). For example:

   `action=Query&Text=Undulant fever&Summary=Concept`

   Each result of this query returns with a conceptual summary.

2. Optionally, make the following settings in the IDOL Content component configuration file, depending on the type of summary that you want to generate:

   - **For Concept, Context, or Quick summaries**. Use the `SourceType` property to specify the fields to generate the summary from.

   - **For Concept or Context summaries**. In the `[Summary]` section, set the `MinWordsPerSentence` parameter to the minimum number of words that a sentence must contain to consider it as a sentence for the summary.

   - **For Context summaries**. In the `[Server]` section, set the `ContextSummaryQueryTermWeight` parameter to the weight that it must use for the terms in user queries. The context summary gives this weight to sentences that contain terms in common with the query text. The other terms are given their APCM weight.

3. Save and close the configuration file.

4. Restart the IDOL Content component for your changes to take effect.

> **NOTE:** If you remove fields from the Source fields list, you must first perform a `DREINITAL` index

> action and then reindex the content into the IDOL Content component. You cannot remove source
> fields after you index the data.

Related Topics

- [About Fields, on page 81](#)

Summarize Text or Documents

To generate summaries for text or documents, send a `Summarize` action to the IDOL Content
component that includes the `Summary` parameter. Set the `Summary` parameter to the type of summary to
generate (**Concept**, **Quick**, **ParagraphConcept**, or **Concepts**).

To generate a summary for text, identify the text in the `Text` parameter. To generate a summary for one
or more documents, identify them in the `ID` or `Reference` parameters.

For example:

```
action=Summarize&ID=30&Summary=Concept
```

This action generates a concept summary from the content of the document with the ID `30`.

Cluster Results

If you perform a `Query`, `Suggest`, or `SuggestOnText` query, you can set `Cluster` to **True** in the action to
cluster the results that the query produces.

The IDOL Content componentreturns an `<autn:cluster>` field for each result that contains the ID of
the cluster into which the result has been grouped. ID 1 is given to the cluster that forms around the
most relevant result document. After this cluster forms, the cluster with the ID 2 forms around the most
relevant of the remaining results, and so on.

The `ClusterThreshold` parameter in the IDOL Content component configuration file sets the
percentage relevance that results must have to each other to group them in one cluster.

For example:

```
action=Query&Text=cats&Cluster=True
```

The IDOL Content component clusters the results of the query above, and returns the ID of the cluster
that each result belongs to in an `<autn:cluster>` field:

```
<autn:hit>
    <autn:reference>http://example.wikipedia.org/wiki/Big Cat
Rescue</autn:reference>
    <autn:id>595644</autn:id>
    <autn:section>0</autn:section>
    <autn:weight>88.04</autn:weight>
    <autn:cluster>1</autn:cluster>
    <autn:clustertitle>Big Cat Rescue, Easy Street</autn:clustertitle>
    <autn:links>CAT</autn:links>
```

```
    <autn:database>Wikipedia</autn:database>
    <autn:title>Big Cat Rescue</autn:title>
</autn:hit>
<autn:hit>
    <autn:reference>http://example.wikipedia.org/wiki/The Cat in the
Hat</autn:reference>
    <autn:id>229422</autn:id>
    <autn:section>1</autn:section>
    <autn:weight>86.93</autn:weight>
    <autn:cluster>2</autn:cluster>
    <autn:clustertitle>Allan, ballad, Hat, Krinklebine</autn:clustertitle>
    <autn:links>CAT</autn:links>
    <autn:database>Wikipedia</autn:database>
    <autn:title>The Cat in the Hat</autn:title>
</autn:hit>
<autn:hit>
    <autn:reference>http://example.wikipedia.org/wiki/The Cat in the
Hat</autn:reference>
    <autn:id>229421</autn:id>
    <autn:section>0</autn:section>
    <autn:weight>86.93</autn:weight>
    <autn:cluster>3</autn:cluster>
    <autn:clustertitle>sight vocabulary, Cat, Hat, Seuss</autn:clustertitle>
    <autn:links>CAT</autn:links>
    <autn:database>Wikipedia</autn:database>
    <autn:title>The Cat in the Hat</autn:title>
</autn:hit>
<autn:hit>
    <autn:reference>http://example.wikipedia.org/wiki/Cat</autn:reference>
    <autn:id>56711</autn:id>
    <autn:section>18</autn:section>
    <autn:weight>86.93</autn:weight>
    <autn:cluster>4</autn:cluster>
    <autn:clustertitle>breeds Polydactyl cat, Halloween, List,
pets</autn:clustertitle>
    <autn:links>CAT</autn:links>
    <autn:database>Wikipedia</autn:database>
    <autn:title>Cat</autn:title>
</autn:hit>
```

Generate Hyperlinks

When the IDOL Content component returns results, it can automatically generate hyperlinks in real time that connect to contextually similar content.

About Hyperlinks

You can use hyperlinks in results to recommend related articles, documents, affinity products or services, or media content that relates to textual content.

Because IDOL Server inserts links automatically at the time that it retrieves the document, they can include references to documents and articles written long before. Similarly, hyperlinks from archived material can link to the latest news or material on that subject.

Here are some example applications of hyperlinks:

- **New Media**. When a user is viewing an article on a news media Internet site, IDOL Server can dynamically link to contextually similar content and recommend related articles in real time.

- **Corporate**. Within a corporate environment, while an employee is reading or writing a document, IDOL Server can suggest contextually similar documents from various sources through dynamically created hyperlinks to documents, multimedia content, and related emails.

- **Legal**. Hyperlinking facilitates the ability to suggest contextually relevant legal content pertinent to the legal issues being researched, reducing the time taken to navigate to the right information and identify previous precedents.

- **CRM**. When a customer service representative attends a customer enquiry, IDOL Server presents answers to the frequently asked questions and related emails in the form of dynamic hyperlinks. This process raises the level of customer service and shortens cycle times.

Implement Hyperlinks

If you connect IDOL Server to an IDOL interface application, the interface generates hyperlinks automatically, for example, when it returns query results or when a user refines a query.

If you connect IDOL Server to a third-party interface application, you can generate hyperlinks automatically by running a `Suggest` action when you return query results. Refer to the ACI API documentation for details on the functions that you require for this.

Provide Spelling Correction

When you perform a query, you can set `Spellcheck` to **True** in your query string to instruct the IDOL Content component to check the spelling of the query text. It suggests corrections for the terms that it detects to be misspelled.

How Spelling Correction Works

To enable spell checking, set the parameters `SpellCheckMaxCheckTerms`, `SpellCheckIncorrectMaxDocOccs`, and `UnstemmedMinDocOccs` in the `[Server]` section of the configuration file before you index content. When you perform a query that includes `Spellcheck=True`, the IDOL Content component uses these settings in the spell checking process, as shown below:

1. Content determines if the query is eligible for spell checking.

   Content checks how many terms the query text contains (it ignores stop words, proper-name terms and hyphenated terms). If the number does not exceed the specified `SpellCheckMaxCheckTerms`, the query is eligible for spell checking.

2. Content determines which terms are misspelled.

   Content checks how many times each query term occurs in its data index. If a term occurs fewer times than the specified `SpellCheckIncorrectMaxDocOccs`, Content assumes that the term is misspelled.

3. Content finds correct spellings and suggests them.

   Content uses a proprietary term-distancing algorithm to find terms in its data index that are closest to the misspelled terms. It then checks how many times these terms occur. If a term occurs at least the specified number of `UnstemmedMinDocOccs` times, it uses it as a spell check suggestion.

   Content returns the corrected terms as a comma-separated list in an `<autn:spelling>` field. It also returns a corrected version of the query text in an `<autn:spellingquery>` field.

4. When you shut down the IDOL Content component, it creates a spelling correction file.

   The spelling correction file stores the corrections that you make. You can add further corrections to the file or amend existing corrections.

Related Topics

- Spelling Correction File, below

Spelling Correction File

If the IDOL Content component has performed spell checking, it generates a `prx.db` spelling correction file in its `IDOL > content > main` directory when it is shut down. The spelling correction file stores the corrections that you make.

For example:

`http://IDOLhost:port/action=Query&Text=san+fransisco&Spellcheck=True`

If you perform this action, Content returns the obvious correction and a corrected version of the query text.

After Content stops, it creates the `prx.db` spelling correction file that contains the corrections it has made in a human-readable form. For example:

```
<?xml version="1.0" encoding="UTF-8" ?><PROXIMALS>
<PROXIMAL ORIG="FRANSISCO" CORRECT="FRANCISCO" />
</PROXIMALS>
```

You can add further corrections to the file or amend any existing ones. You must specify the original and corrected spellings in upper case.

Your changes take effect when you start Content again.

You can exempt words from being spell checked by entering them and omitting the corrected attribute, for example:

```
<PROXIMAL ORIG="TELEFONE" />
```

> **NOTE:** Any changes that you make to the `prx.db` file must be valid XML, otherwise they are ignored.

> **NOTE:** You can use the `SpellCheckCacheMaxSize` setting in the `[Server]` section of the configuration file to increase the number of spelling corrections that the cache and the `prx.db` file can store.

Related Topics

- Edit the Spelling Correction Cache, on page 438

Automatic Query Guidance

When it performs a query, the IDOL Content component can find the most salient terms and phrases in the query results. It can automatically cluster these terms and phrases and use the clustered phrases to provide a hierarchical set of queries that guide users to the result area that they are looking for.

About Automatic Query Guidance

Here is an example of the use of Automatic Query Guidance:

In this example, the IDOL Content component generates three clusters from documents that match the query **apollo**, and lists the top terms or phrases in each cluster as alternative queries. You can click any of these queries to perform them. In this example, you can display results that do not fall into any of the clusters by clicking **Other**.

To set up Automatic Query Guidance

1. Configure the IDOL Content component to enable Automatic Query Guidance.

2. Send an action with the `QuerySummary` parameter.

Enable Automatic Query Guidance

Use the following procedure to enable automatic query guidance.

To configure the IDOL Content component to generate query summaries

1. Open the IDOL Content component configuration file in a text editor.

2. In the `[Server]` section, configure the following parameters:

`QuerySummaryAdvanced`	Set this parameter to `True` to enable the advanced algorithm that Content requires to provide query guidance. This algorithm clusters the best terms and phrases in the query results, and uses the phrases of the top clusters as query summaries. The default value is `False`.
`QuerySummaryLength`	Set this parameter to the number of cluster phrases to generate. To provide good query guidance phrases, enter a number that is large enough to allow Content to generate high-quality clusters. However, setting `QuerySummaryLength` too high might slow the query performance (a sensible maximum value might be 25–100). The default value is `10`.
`QuerySummaryTerms`	Set this parameter to the number of terms that the algorithm for `QuerySummaryLength` can use to generate the query summaries (a sensible value might be 50–500). The default value is 0. In this case, Content uses the specified `TermsPerDoc` number, which in turn defaults to 50.

> **TIP:** You can also set these parameters in the `Query` action so that you can test different values without reconfiguring the IDOL Content component, or tune `QuerySummary` functionality for different uses.

About the QuerySummary Parameter

To perform an action with the `QuerySummary` parameter, add the following parameters to your `Query`, `Suggest`, or `SuggestOnText` action:

QuerySummary=True	Content identifies the three most relevant terms and phrases of the clustered phrases, and returns them in an `<autn:querysummary>` field. It also returns a list of the specified `QuerySummaryLength` number of the best terms and phrases.
Combine=Simple	If multiple sections in a result document match the query, Content displays only the section with the highest conceptual similarity (rather than returning different sections of the same result document). This increases the relevance of the final phrases.
Print=NoResults	The purpose of the query is to provide query guidance, but not return results at the same time. Printing the results slows the IDOL Content component performance.
MaxResults=*N*	Set *N* to the number of results from which to generate the clustered terms and phrases. Content can produce high-quality clusters only if it has enough result documents available. However, setting `MaxResults` too high can slow the query performance (a sensible maximum value might be 50–500).

For example:

```
action=Query&Text=Apollo&QuerySummary=True&Combine=Simple&Print=NoResults&MaxResult
s=100
```

In this example, if you have set `QuerySummaryLength` to **10** in the configuration file, Content clusters the results that the query produces and lists the top 10 relevant terms or phrases that the results contain:

```
<autn:element pdocs="52" poccs="108" cluster="0" docs="56">Apollo
program</autn:element>
<autn:element pdocs="29" poccs="83" cluster="0" docs="34">Neil
Armstrong</autn:element>
<autn:element pdocs="22" poccs="43" cluster="0" docs="27">command
module</autn:element>
<autn:element pdocs="18" poccs="61" cluster="0" docs="54">Lunar
Module</autn:element>
<autn:element pdocs="13" poccs="13" cluster="1" docs="13">Greek
mythology</autn:element>
<autn:element pdocs="15" poccs="22" cluster="2" docs="35">Royal Navy</autn:element>
<autn:element pdocs="22" poccs="28" cluster="2" docs="26">HMS Apollo</autn:element>
<autn:element pdocs="14" poccs="21" cluster="0" docs="14">moon
landings</autn:element>
<autn:element pdocs="8" poccs="8" cluster="2" docs="9">Leander-class
frigate</autn:element>
<autn:element pdocs="12" poccs="14" cluster="1" docs="15">Greek god
Apollo</autn:element>
```

The information for the top phrases and terms in the clusters that Content has generated consists of the following elements:

pdocs	The number of documents in which the entire phrase appears.
poccs	The total number of times that the phrase appears in all documents.
cluster	The number of the cluster that the phrase falls into. A negative number indicates that the associated documents do not fall into any of the generated clusters.
docs	The number of documents in which all the terms appear.

For example:

```
<autn:element pdocs="52" poccs="108" cluster="0" docs="56">Apollo
program</autn:element>
```

In this example, the phrase "Apollo program" occurs in 52 documents out of the 100 results. The phrase appears 108 times in total in these documents. The phrase falls into cluster 0.

The terms "Apollo" and "program" appear in 56 documents.

Content also returns the best phrases or terms of the top three clusters in a comma-separated list in the `<autn:querysummary>` field:

```
<autn:querysummary>Apollo program, Greek mythology, Royal Navy</autn:querysummary>
```

You can use each of these top cluster phrases as an alternative query, as well as the less relevant terms and phrases in each cluster.

Generate Query Summaries (Dynamic Thesaurus)

When it performs a query, the IDOL Content component can automatically generate query summaries that contain the most salient terms and phrases of the query result documents. It can use these terms and phrases to suggest alternative queries to users, allowing them to further refine queries and quickly produce a variety of relevant result sets.

About Query Summaries

Here is an example of the use of Query Summaries:
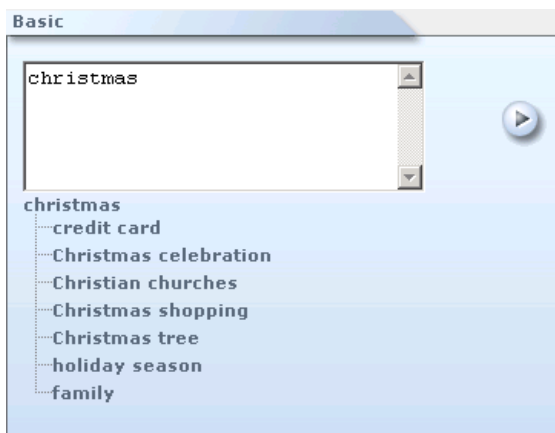
In this example, the IDOL Content component has identified the seven phrases that are most relevant to the query **christmas** that result documents contain, and listed them as query summaries. Users can click any of these phrases to perform a new query.

To automatically generate query summaries

1. Configure the IDOL Content component to generate query summaries.

2. Perform an action with the `QuerySummary` parameter.

Configure the IDOL Content component to Generate Query Summaries

Use the following procedure to enable query summary generation.

To configure the IDOL Content componentto generate query summaries

1. Open the IDOL Content component configuration file in a text editor.

2. In the `[Server]` section, set the following parameters:

QuerySummaryLength	Set this parameter to the number of query summaries to return when you perform an action that includes the `QuerySummary` parameter. For example, if you set `QuerySummaryLength` to **5**, Content identifies the five top phrases or terms that the result documents contain, and returns them in a comma-separated list. If Content cannot find your requested number of relevant phrases or terms, it returns fewer. The default value is **10**
QuerySummaryAdvanced	Set this parameter to **True** to use an advanced algorithm to generate the specified `QuerySummaryLength` number of query summaries. This algorithm clusters the results and uses the phrases of the top clusters as query summaries. This process

	produces better query summaries, but is more system intensive.
	Set this parameter to **False** if you want query summaries without first clustering the results.
	The default value is **False**.
QuerySummaryTerms	If you set QuerySummaryAdvanced to **True**, QuerySummaryTerms allows you to specify the number of terms that the algorithm uses to generate the query summaries (a sensible value might be 50–500). If you set QuerySummaryTerms to **0**, Content uses the specified value of TermsPerDoc, which in turn defaults to 50.
	The default value is **0**.

> **TIP:** You can also set these parameters in the Query action so that you can test different values without reconfiguring the IDOL Content component, or tune QuerySummary functionality for different uses.

Perform an Action with the QuerySummary Parameter

To perform an action with the QuerySummary parameter, you can add the following parameters to your Query, Suggest, or SuggestOnText action:

QuerySummary=True	Content identifies the most relevant terms and phrases that the query result documents contain, and returns them in an `<autn:querysummary>` field. You can use each term or phrase in this field as an alternative query suggestion.
Combine=Simple	If multiple sections in a result document match the query, Content displays only the section with the highest conceptual similarity (rather than returning different sections of the same result document). This option increases the relevance of the final phrases.
Print=NoResults	The purpose of the query is to generate query summaries but not return results at the same time. Printing the results slows IDOL Content component performance.
MaxResults=N	Enter the number of results from which you want to generate the query summaries. Content can produce good query summaries only if it has enough result documents available from which it can identify sufficiently different phrases. However, setting MaxResults too high might slow the query performance (a sensible value might be 50–500).

For example:

```
action=Query&Text=Date&QuerySummary=True&Combine=Simple&Print=NoResults&MaxResults=
100
```

In this example, if you have set `QuerySummaryLength` to **5** in the configuration file, Content finds the top five phrases or terms in results that match the specified `Text` value, and returns them in a comma-separated list in the `<autn:querysummary>` field:

```
<autn:querysummary>Date conversion, rendezvous, stuffed dates, appointment, go
steady</autn:querysummary>
```

It can use each of these terms and phrases as an alternative query.

Generate Dynamic Clusters

Every time it runs a query, the IDOL Content component can automatically cluster the results that are available for this query, and then in turn cluster the first set of clusters further to produce subclusters. This process allows you to generate a hierarchy of dynamic clusters that allows users to navigate quickly to their area of interest.

For example:
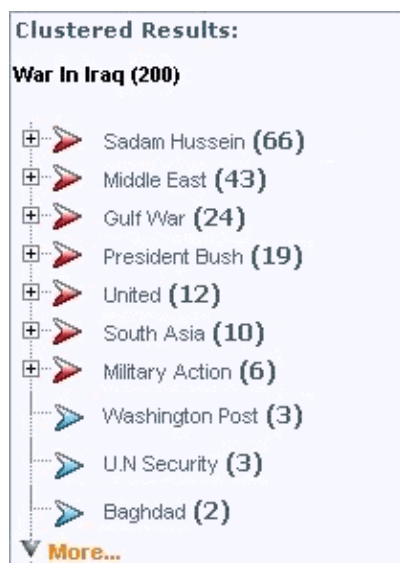
In this example, Content has generated 10 dynamic clusters from documents that match the query **War In Iraq**. A plus sign (+) and a red arrow next to a cluster in this interface indicates that you can expand the cluster to display subclusters. A blue arrow indicates clusters that cannot cluster further.

Content lists each cluster with the best phrase it contains. The number in brackets indicates how many documents the cluster contains. The user can click the best phrase for a cluster to run a query with this phrase and automatically cluster the results that this query produces.

Dynamic clustering can be faster than clustering in the background, through `ClusterCluster` and `ClusterServe2DMap`. However, using `ClusterCluster` and `ClusterServe2DMap` is more powerful and accurate. Also, dynamic clustering is more suited to finding trends in smaller data sets. `ClusterServe2DMap` is better for clustering an entire database.

Related Topics

- Cluster Information, on page 197

To generate dynamic clusters

1. Configure the IDOL Content component to enable dynamic clustering.

2. Perform an action with the `QuerySummary` parameter.

Configure the IDOL Content component to Enable Dynamic Clusters

Use the following procedure to enable dynamic clustering in the IDOL Content component.

To configure the IDOL Content componentto enable dynamic clustering

1. Open the IDOL Content component configuration file in a text editor.

2. In the `[Server]` section, set the following parameters:

QuerySummaryAdvanced	Set this parameter to **True** to enable the advanced algorithm that provides dynamic clustering. This algorithm clusters the best terms and phrases from the query results, and lists them. The default value is **False**.
QuerySummaryLength	Set this parameter to the number of cluster phrases to generate. Setting `QuerySummaryLength` too high can slow the query performance (a sensible value might be 25–250). The default value is **10**.
QuerySummaryIDs	Set this parameter to **True** to return the document IDs of the parent documents that phrases come from. The IDs return in the list that Content generates when you perform an action with the `QuerySummary` parameter. You can use the IDs to display the number of documents that each dynamic cluster contains. The default value is **False**.
QuerySummaryTerms	Specify the number of terms that the algorithm for `QuerySummaryLength` can use to generate the query summaries (a sensible value might be 50–500). If you set this parameter to **0**, Content uses the specified `TermsPerDoc` number, which in turn defaults to `50`. The default value is **0**.

> **TIP:** You can also set these parameters in the `Query` action so that you can test different values without reconfiguring Content, or tune `QuerySummary` functionality for different uses.

Perform an Action with the QuerySummary Parameter

To perform an action with the `QuerySummary` parameter, add the following parameters to your `Query`, `Suggest`, or `SuggestOnText` action:

`QuerySummary=True`	Content identifies the three most relevant terms and phrases of the clustered phrases, and returns them in an `<autn:querysummary>` field. It also returns a list of the specified `QuerySummaryLength` number of the best terms and phrases.
`Combine=Simple`	If multiple sections in a result document match the query, Content displays only the section with the highest conceptual similarity (rather than returning different sections of the same result document). This process increases the relevance of the final phrases.
`Print=NoResults`	The purpose of the query is to provide dynamic clustering but not return results at the same time. Printing the results slows the IDOL Content component performance.
`MaxResults=N`	Set this parameter to the number of results from which you want to generate the terms and phrases. Setting `MaxResults` too high can slow the query performance (a sensible value might be 50–500).

For example:

```
action=Query&Text=War In
Iraq&QuerySummary=True&Combine=Simple&Print=NoResults&MaxResults=100
```

Display Cluster Information

In the previous example, if you have set `QuerySummaryLength` to **10** in the configuration file, the IDOL Content component lists the top 10 relevant terms or phrases that the results contain:

```
<autn:element pdocs="63" poccs="132" cluster="0" docs="66"
ids="398867,388941,...">Saddam Hussein</autn:element>
<autn:element pdocs="43" poccs="78" cluster="0" docs="43"
ids="398867,343399,...">Middle East</autn:element>
<autn:element pdocs="22" poccs="41" cluster="0" docs="24"
ids="388941,338798,...">Gulf War</autn:element>
<autn:element pdocs="17" poccs="32" cluster="0" docs="19"
ids="409508,398867,...">President Bush</autn:element>
<autn:element pdocs="12" poccs="17" cluster="1" docs="12"
ids="326892,326496,...">United</autn:element>
<autn:element pdocs="10" poccs="15" cluster="1" docs="10"
ids="388941,326892,...">South Asia</autn:element>
<autn:element pdocs="4" poccs="7" cluster="1" docs="6"
ids="428429,326497,...">Military Action</autn:element>
<autn:element pdocs="3" poccs="4" cluster="1" docs="3"
```

```
ids="326496,227758,...">Washington Post</autn:element>
<autn:element pdocs="2" poccs="4" cluster="1" docs="3" ids="324404,227756,...">U.N.
Security</autn:element>
<autn:element pdocs="2" poccs="3" cluster="1" docs="2"
ids="398567,343199,..."Baghdad</autn:element>
```

The information for the top phrases and terms that Content has generated consists of the following elements:

pdocs	The number of documents in which the entire phrase appears.
poccs	The total number of times that the phrase appears in documents.
cluster	The number of the Automatic Query Guidance cluster that the phrase falls into. A negative number indicates that the associated documents do not fall into any of the generated clusters.
docs	The number of documents in which all the terms appear.
ids	The IDs of the documents in which the phrases appear. You can use the IDs in a front end to display how many documents a dynamic cluster contains.

Related Topics

- Automatic Query Guidance , on page 331
- Display the Number of Documents a Dynamic Cluster Contains, below

For example:

```
<autn:element pdocs="63" poccs="132" cluster="0" docs="66"
ids="398867,388941,...">Saddam Hussein</autn:element>
```

In this example, the phrase "Saddam Hussein" occurs in 63 documents out of the 100 results. The phrase appears 132 times in total. The phrase falls into Automatic Query Guidance cluster 0. A total of 66 documents contain the terms "Saddam" and "Hussein".

Content also returns the best phrases or terms of the top three Automatic Query Guidance clusters in a comma-separated list in the `<autn:querysummary>` field. However, dynamic clustering does not use these terms.

```
<autn:querysummary>Saddam Hussein, Middle East, Gulf War</autn:querysummary>
```

Display the Number of Documents a Dynamic Cluster Contains

You can use the document IDs that dynamic clustering returns to display how many documents a dynamic cluster contains in a front-end application.

For example, assume that a query for `Apollo` returns 10 result documents with the document IDs listed in the following table.

Dynamic cluster name	IDs of documents in this dynamic cluster							
Neil Armstrong		2			5	7		9	10
command module					5	7		9	
moon landing		2			5		8	9	
Greek mythology	1		3	4					
Zeus	1		3						

When Content displays the total number of documents that a dynamic cluster contains, it counts documents for a dynamic cluster only if they were not already counted for a previous dynamic cluster. The clusters have the following totals:

- Neil Armstrong (5)

- Greek mythology (3)

- moon landing (1)

- other (1)

Content determines the totals in the following way:

- The `Neil Armstrong` cluster contains five documents.

- The `command module` cluster contains only documents that were counted for the previous cluster, so it is not listed.

- The `moon landing` cluster contains one document because the documents with the IDs `2`, `5`, and `9` were counted for previous clusters.

- The `Greek myth` cluster contains three documents.

- The `Zeus` cluster contains only documents that were counted for the previous cluster, so it is not listed.

- One document that the query returned did not fit into any of the clusters. Content lists this document in the `other` cluster.

Create a Cluster Map

You can use the `ClusterMapFromResults` action to create a map of the clusters that you generate from your query.

You provide XML-formatted cluster information to `ClusterMapFromResults` as the value of the `ImportXML` parameter, and the action returns a 2-D cluster map as binary image data. The file is by default in JPEG format, but (as in the case of the `ClusterServe2DMap` action) you can specify another format by setting the `PictureFormat` configuration parameter in the `[Cluster]` section of the IDOL Content component configuration file.

`ClusterMapFromResults` also creates and stores a new set of cluster results that includes the map coordinates of the clusters. If your application uses these cluster results to support mouse-over effects when displaying the cluster map, you can retrieve the results by using the `ClusterResults` action. In

this case, use the `TargetJobName` from the `ClusterMapFromResults` action as the `SourceJobName` in the `ClusterResults` action.

When you use the `ImportXML` parameter to send data to `ClusterMapFromResults`, you must use the following format:

```
<?xml version='1.0' encoding='UTF-8' ?>
<autn:clusters xmlns:autn='http://schemas.autonomy.com/aci/'>
    <autn:numclusters>5</autn:numclusters>
    <autn:cluster>
        <autn:title>MyClusterTitle</autn:title>
        <autn:score>5.63</autn:score>
        <autn:term>CAT~[85]</autn:term>
        <autn:term>DOG~[76]</autn:term>
        <autn:term>RABBIT~[65]</autn:term>
        <autn:term>HAMST~[64]</autn:term>
        <!-- more terms -->
        <autn:docs>
            <autn:numdocs>3</autn:numdocs>
            <autn:doc>
                <autn:title>MyDocTitle</autn:title>
                <autn:ref>http://foo.com</autn:ref>
                <autn:score>5</autn:score>
            <autn:doc>
        <!-- more docs-->
        </autn:docs>
    </autn:cluster>
    <!-- more clusters-->
</autn:clusters>
```

This XML format is similar to the format of the XML results of a `Query` action. If you pass this data to the `ClusterMapFromResults` action in the `ImportXML` parameter, you must first percent-encode it.

> **NOTE:** As an alternative to providing XML data to `ClusterMapFromResults`, you can provide it with a job name from a previous `ClusterCluster` action.

Related Topics

- Generate a Cluster Map after You Cluster, on page 201

Chapter 16: Manipulate Result Relevance

This section describes the methods that you can use to boost the relevance of a result in a results list.

Boost Relevance

You can boost the percentage relevance that the IDOL Content component gives to query results by using the following methods:

- **Use a field process**. You can set up a field process in the IDOL Content component configuration file to boost the percentage relevance of query results according to the number of times terms in specified fields match the query terms.

- **Use BIAS**. You can use the `BIAS` field specifier at query time to boost the percentage relevance of query results according to the numerical proximity of a specified field to a particular value.

- **Use multipliers**. You can multiply the weight of individual query terms to boost the relevance of results that match these terms accordingly.

- **Use the AutnRankType field**. You can create a field in documents that indicates their importance and instruct Content to take the value of this field into account when it calculates the relevance that the document is given if it returns as a result.

Use a Field Process to Boost Relevance

You can set up a field process that identifies specific fields in documents and manipulates the weight of terms in these fields if they match the query terms.

For example, use the following procedure to boost the weight of results that contain query terms in their `DRETITLE` and `SUMMARIES` fields.

1. For each field whose content you want to use to determine whether to boost result weights, list a process in the `[FieldProcessing]` section of the IDOL Content component configuration file that indexes the field and manipulates its term weights. You can use one field process to boost terms

in several fields by the same factor.

For example:

```
[FieldProcessing]
0=IndexAndWeightHigher1
1=IndexAndWeightHigher2
```

2. Create a section for each of the processes that you listed, in which you create a property for the process (you define the property later by setting one or more applicable configuration parameters). Identify the fields that you want to associate with the processes.

   > **NOTE:** The properties that you create must not have the same name as the processes.

   For example:

   ```
   [IndexAndWeightHigher1]
   Property=IndexHigherWeight1
   PropertyFieldCSVs=*/DRETITLE


   [IndexAndWeightHigher2]
   Property=IndexHigherWeight2
   PropertyFieldCSVs=*/SUMMARY
   ```

3. Create a section for each of the properties and specify configuration settings for each. The `Index` parameter ensures that Content indexes the fields that you associate with the field process. The `Weight` parameter determines the factor by which to boost terms in the associated `PropertyFieldCSVs` fields if they match query terms.

   For example:

   ```
   [IndexHigherWeight1]
   Index=True
   Weight=4


   [IndexHigherWeight2]
   Index=True
   Weight=2
   ```

4. Save and close the configuration file.

5. Restart the IDOL Content component for your changes to take effect.

In future queries, IDOL Server boosts the relevance of the result to the query according to how many times the terms in the result SUMMARY and DRETITLE fields match the query terms.

For example, the following query boosts results whose SUMMARY and DRETITLE field matches the query terms *cat* and *dog*:

```
http://IDOLhost:port/action=Query&Text=cat and dog
```

This means that these results return in the following order:

- **Result 1**:

  Title = Cats & Dogs

Summary = **Cats** and **dogs** duke it out in this live action feature about a professor on the brink of discovering a cure for **dog** allergies. The **dogs** assign an agent to protect the professor and his family from a feline invasion.

Content = Unbeknownst to humans, **dogs** have fought for thousands of years to keep mankind from falling under the rule of **cats**. Using combinations of live animals, animatronic puppets, and digital wizardry, this film has just enough imagination to match its effects, climaxing with a feline global-domination scheme involving mice sprayed with chemicals that will make all humans allergic to their canine friends.

- **Result 2**:

  Title = Garfield

  Summary = Garfield comes to life in an all new live action major motion picture.

  Content = Garfield is a fat **cat**. A **cat** that eats lots of Lasagne. A **cat** that is lazy and sleeps as much as possible. Nevertheless, Garfield is a clever **cat**, always able to outwit his owner, Jon and the neighbor's **dog**, Odie. Garfield is a cool and sarcastic **cat** but he is also a **cat** with a heart as is shown when he comes to the rescue of Odie the **dog**, in the movie that is coming out this year. The hapless pup disappears and is kidnapped by a nasty **dog** trainer, and Garfield feels responsible. Pulling himself away from the TV, Garfield springs into action. Maybe it's friendship for **cat** and **dog** after all.

- **Result 3**:

  Title = Tom and Jerry: The movie

  Summary = The celebrated **cat** and mouse team meets a young runaway who desperately needs their help to find her missing father. Along the way they run into her evil Aunt who tosses them into a pet prison. Bonding together, Tom & Jerry outwit the Aunt and mastermind a great escape to set off on the wildest adventures of their **cat** and mouse careers.

  Content = The popular animated duo team up again to appear this time on the big screen. Homeless, the 'toons end up helping out a young girl who stays with a nasty auntie while she is separated from her father. Will the young Robyn be reunited with her loving father? Will the odd pair make it on the streets? Will they find a home? Those are some of the burning questions that may plague the minds of young viewers of this fun adventure.

Without the boosting to the weight of the SUMMARY and DRETITLE field, Result 2 is the top result, with Result 1 following in second place.

Result 3 does not rank higher than Result 2 in either case. Although its weight is slightly boosted because its SUMMARY field contains one of the query terms, this boost is not sufficient to outrank Result 2.

Use the BIAS Field Specifier to Boost Relevance

The BIAS field specifier allows you to bias the score of results at query time according to the numerical proximity of the specified field to a particular value. It ignores initial dollar ($), pound sign (£), or hyphen

(–) characters in the field names.

Specify BIAS in the format:

BIAS{*optimum,range,percentage*}

where:

optimum	is the value that the field must contain to increase or decrease the result weight by the maximum *percentage*.
range	is a positive number that determines the range of the *optimum*. If the specified field contains a value that is in the range of (*optimum* – *range*) to (*optimum* + *range*), the result weight increases or decreases according to the specified *percentage*.
percentage	is a percentage in the range –100 to 100. If the value of the specified field is in the specified range, the score of the result increases or decreases according to how close the value is to the specified *optimum*.

For example:

- `http://`*IDOLhost*`:`*port*`/action=Query&FieldText=BIAS{100,50,10}:*/PRICE`

  A document whose PRICE field value is within the range 50 either side of 100 has its weight increased on a linear scale from 10 percent if the price is 100, to 0 percent if the price is 50 or 150:
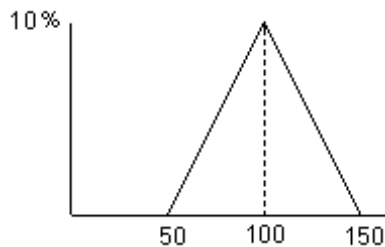
- `http://`*IDOLhost*`:`*port*`/action=Query&FieldText=BIAS{100,50,-10}:*/PRICE`

  A document whose PRICE field value is within the range 50 either side of 100 has its weight decreased on a linear scale from –10 percent if the price is 100, to 0 percent if the price is 50 or 150:
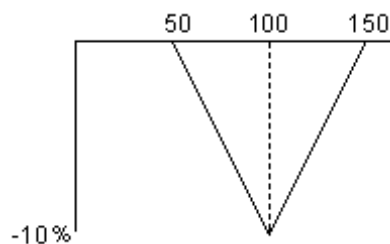
- You can also use the BIAS field specifier to bias the score of results according to the numerical proximity in their autn_date metafield to a particular value. For example:

  `FieldText=BIAS{1103918400,259200,25}:autn_date`

A document whose `autn_date` field value is within the range 259200 either side of 1103918400 has its weight increased on a linear scale from 25 percent if the price is 1103918400, to 0 percent, if the date is 1103659200 or 1104177600.
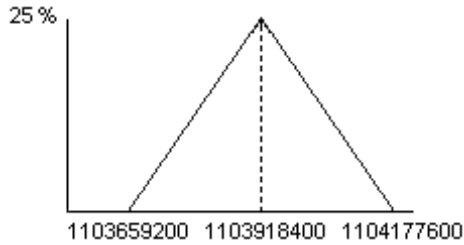
Related Topics

- [Metadata Fields, on page 105](#)

BIASDATE

The `BIASDATE` field operator works in a similar way to the `BIAS` field operator. It allows you to bias the score of results at query time according to the proximity of the specified field to a particular date.

Specify `BIASDATE` in the format:

`BIASDATE{`*optimumDate,range,percentage*`}:`*yourFields*

where:

optimumDate	is the date that the field must contain to increase or decrease the result weight by the maximum *percentage*. See Date formats , on page 249 for a list of the available date formats.
range	is a positive value that determines the range in seconds of the specified optimum. If the specified field contains a value that is in the range ($optimum - range$) to ($optimum + range$), the result weight is increased or decreased according to the specified *percentage*.
percentage	is a percentage in the range –100 to 100. If the value of the specified field is within the specified range, the score of the result increases or decreases according to how close the value is to the specified optimum. When you set `Absweight` to **True**, this is the absolute value by which to boost the weight, and it is then not limited by +/– 100.
yourFields	are one or more fields. A document has its score boosted if it contains one of these fields, and if this field contains a date that falls within the specified range of the optimum date. Separate multiple fields with colons (`:`). There must be no space before or after a colon.

For example:

- `FieldText=BIASDATE{1/12/2008,864000,10}:DATE`

  A document whose `DATE` field value is within a range of 10 days (864000 seconds) either side of 1/12/2008 has its weight increased on a linear scale from 10 percent if the date is 1/12/2008, to 0 percent if the date is 20/11/2008 or 11/12/2008.

- `FieldText=BIASDATE{1/12/2008,86400,-10}:DATE`

  A document whose `DATE` field value is within a range of 10 days (864000 seconds) either side of 1/12/2008 has its weight decreased on a linear scale from −10 percent if the date is 1/12/2008, to 0 percent if the date is 20/11/2008 or 11/12/2008.

BIASDISTCARTESIAN

The `BIASDISTCARTESIAN` field operator works in a similar way to the `BIAS` field operator. It allows you to bias the score of results at query time according to the proximity of the specified field to a particular location, specified using Cartesian coordinates.

> **TIP:** If your document location fields contain regions (in Well-known text POLYGON format), the `BIASDISTCARTESIAN` operator calculates distances from the geometric center of the region.

Specify `BIASDISTCARTESIAN` in the format:

`FieldText=BIASDISTCARTESIAN{coordX,coordY,range,percentage}:POSITION`

where:

coordX	is the X coordinate.
coordY	is the Y coordinate.
range	is the distance in kilometers from the coordinates. If the fields contain the coordinates of a location that is within this distance of the coordinates, the results weight increases or decreases according to the specified percentage.
percentage	is a percentage in the range −100 to 100. If the value of the field is within the specified range, the score of the result increases or decreases according to how close the value is to the optimum.
POSITION	The document field or fields that contain the location value. You can use one of the following options to specify location fields: - a single field. This field must contain a Well-known text format POINT or POLYGON definition (for example, `POINT (x y)`). This field must be a unified `GeospatialType` field. - two fields, in the format *X:Y*, where *X* is the field that contains the x coordinate, and *Y* is the field that contains the y coordinate. You must specify the fields in the order `x:y`. The fields must be `NumericType` or `GeospatialType`.

For example:

`FieldText=BIASDISTCARTESIAN{10,11,5,7}:X:Y`

`FieldText=BIASDISTCARTESIAN{10,11,5,7}:POSITION`

In these examples, all documents whose (X/Y) position is within a distance of 5 units of the point (10,11) are given a relevance boost. The maximum boost of 7 percent is given to documents at the given point. The boost decreases linearly down to 0 boost at 5 units.

In the first example, the location data is stored in two document fields, X and Y. In the second example, the location data is stored in a single POSITION field.

BIASDISTSPHERICAL

The BIASDISTSPHERICAL field operator works in a similar way to the BIAS field operator. It allows you to bias the score of results at query time according to the proximity of the specified field to a particular location, specified using a latitude and longitude value.

> **TIP:** If your document location fields contain regions (in Well-known text POLYGON format), the BIASDISTSPHERICAL operator calculates distances from the geometric center of the region.

Specify BIASDISTSPHERICAL in the format:

`FieldText=BIASDISTSPHERICAL{`*`Lat,Long,range,percentage`*`}:`*`LOCATION`*

where:

|	
Lat	is the latitude. Specify latitude positions south of the equator as negative.
Long	is the longitude. Specify longitude positions west of the Greenwich Meridian as negative.
range	is the distance in kilometers from the specified coordinates. If the fields contain the coordinates of a location that is within this distance of the specified coordinates, the results weight increases or decreases according to the specified percentage.
percentage	is a percentage in the range –100 to 100. If the value of the field is within the specified range, the score of the result increases or decreases according to how close the value is to the specified optimum.
LOCATION	The document field or fields that contain the location value. You can use one of the following options to specify location fields: ● a single field. This field must contain a Well-known text format POINT or POLYGON definition (for example, POINT (x y)). This field must be a unified GeospatialType field. ● two fields, in the format *LATFIELD*:*LONGFIELD*, where *LATFIELD* is the field that contains the latitude value, and *LONGFIELD* is the field that contains the longitude value. You must specify the fields in the order latitude:longitude. The fields must be NumericType or GeospatialType.

For example:

`FieldText=BIASDISTSPHERICAL{52.2,0.1,100,7}:LAT:LONG`

`FieldText=BIASDISTSPHERICAL{52.2,0.1,100,7}:LOCATION`

In these examples, all documents within 100 kilometers of the point (`lat,long`)=(`52.2,0.1`) are given a relevance boost. The maximum boost of 7 percent is given to documents at the given point. The boost decreases linearly down to 0 boost at 100 kilometers.

In the first example, the location data is stored in two document fields, `LAT` and `LONG`. In the second example, the location data is stored in a single `LOCATION` field.

Use Multipliers to Boost Relevance

You can add multipliers to individual query terms to boost the relevance of results that match these terms accordingly.

To apply a multiplier to a query term, use this format:

queryTerm[**N*]

where:

queryTerm	is the query term whose weight to multiply.
N	is the factor to multiply the specified query term weight by. This weight can be any positive number.

For example:

`http://`*IDOLhost*`:`*port*`/action=Query&Text=bread[*2.5]+brown+loaf`

This action multiplies the weight of the query term `bread` by 2.5. The weight of the query terms `brown` and `loaf` does not change.

When results return for the query, the relevance of documents that contain the term `bread` is boosted relative to those that do not.

`http://`*IDOLhost*`:`*port*`/action=Query&Text=SOUNDEX(bred)+bred[*4]`

In this example, a supermarket wants to ensure that an online search for `bread` returns appropriate results. The supermarket has found that customers tend to misspell *bread* as *bred*. If a customer queries for `bread`, appropriate results return as usual. If a customer queries for `bred`, the term is submitted twice; once as a Soundex keyword search, and once with a multiplier. This ensures that if results exist that match *bred* (for example, a new CD by a band called *bred*), they return with a higher relevance than results that match *bred* phonetically.

Similarly, you can use multipliers to reduce the influence of individual query terms. For example:

`http://`*IDOLhost*`:`*port*`/action=Query&Text=cat[*0.5]+dog`

In this example, the weight of the query term `cat` is halved by multiplying it by 0.5. The weight of the query term `dog` does not change.

When you return results for the query, the relevance of documents that contain the term `cat` is reduced relative to those that do not.

Related Topics

-

Use the AutnRankType Field to Boost Relevance

Each document that is stored in the IDOL Content component is given an `AutnRank` value which indicates the importance of the document. By default Content ignores this value and gives all documents an `AutnRank` value of `0`.

You can boost the relevance of documents according to how important they are to you by creating a field in the documents that indicates their importance, and instructing Content to read the `AutnRank` value from this field. If two documents match a query equally well, the one that has the higher value in this field returns with a higher relevance rating.

To boost results according to their AutnRank value

1. In the `[Server]` section of the IDOL Content component configuration file, set `AutnRank` to **True**.

   This instructs Content to take the `AutnRank` value of a document into account when it calculates the relevance that the document is given if it returns as a result.

2. In the `[FieldProcessing]` section, set up a ranking process. This process allows Content to identify which field in a document contains its `AutnRank` value.

   For example:

   ```
   [FieldProcessing]
   0=SetAutnRankField
   ```

3. Create a section for the ranking process that you listed, in which you create a property for the process. Identify the field that you want to associate with the process.

   To identify the fields, use the format */FieldName* to match root-level fields, **/FieldName* to match all fields except root-level, or */Path/FieldName* to match fields that the specified path points to.

   > **NOTE:** The property that you create must not have the same name as the process.

   For example:

   ```
   [SetAutnRankField]
   Property=ReadAutnRank
   PropertyFieldCSVs=*/AUTNRANK
   ```

   In this example Content reads the `AutnRank` value of the document from its `AUTNRANK` field.

4. Create a section for the property in which you set the `AutnRankType` parameter to **True**.

```
[ReadAutnRank]
AutnRankType=True
```

5. Save and close the configuration file.

6. Restart the IDOL Content component for your changes to take effect.

Chapter 17: Manipulate the Results Set

If you are running a `Query`, `Suggest`, or `SuggestOnText` action, you can use several parameters to manipulate the generation of results. You can also store the state of your results from any of those actions, for later retrieval and processing.

Combine Parameter

The `Combine` action parameter combines results that:

- derive from the same document.

- contain the same content.

- have the same value in a specific `ReferenceType` field.

By default, the IDOL Content component displays the result with the highest relevance. However, you can use the `Sort` action parameter to set alternative sorting methods.

You can set `Combine` to one of the following options:

- `Simple`

- `FieldCheck`

- *`ReferenceTypeFields`*

Simple

In general, Micro Focus recommends that you use the `Simple` option for `Combine`.

When the IDOL Content component indexes very long texts, by default it breaks them into sections, and indexes them as individual documents. Each document has its own ID, but they all have the same document reference. This process stabilizes indexing and ensures that the most relevant section of a text returns when you query Content (for example, rather than an entire book). However, if several sections match the query, each section returns as a result. A query can contain multiple results with the same document reference, which belong to the same text, for example, different pages of the same book. If you display each result with the `Print` action parameter set to **AllSections**, you see the same text every time.

You can prevent Content from returning different sections of the same source text by adding `Combine=Simple` to the query. Content displays only the section that has the highest conceptual similarity to the query (unless you add `Print=AllSections` to the query, in which case it displays the

entire source text). If multiple sections have the same conceptual relevance, Content returns the one with the lowest section number.

For example:

`http://IDOLhost:port/action=Query&Text=The Moonstone&Combine=Simple`

In this example, if several results derive from the same source text, Content displays only the result that has the highest relevance to the query text.

FieldCheck

The `FieldCheck` option combines results based on the hash value of their `FieldCheckType` field. The `FieldCheckType` field holds a value that is frequently used to restrict results (for example, a field that stores category names). When Content indexes a `FieldCheckType` field, it stores it in a fast lookup table in memory, so that it can return quickly.

Related Topics

- FieldCheckType Fields, on page 97

For example:

`http://IDOLhost:port/action=Query&Text=The best thing to do in your spare time&Combine=FieldCheck`

In this example, Content is configured to store the `Category` field as a `FieldCheckType` field.

If Content contains 50 documents that match the query text, of which eight contain a `Category` field with the value `Sport`, five contain a `Category` field with the value `Gardening`, and one contains a `Category` field with the value `Cooking`, the above query returns only three results:

- the most relevant of the documents whose `Category` contains the value `Sport`.
- the most relevant of the documents whose `Category` contains the value `Gardening`.
- the document whose `Category` contains the value `Cooking`.

> **NOTE:** If you set `URLAnalysis` to **True** in the `[Server]` section of the IDOL Content component configuration file, you cannot identify a field as a `FieldCheckType` field, because Content automatically uses the domain of the URL it finds in the document `ReferenceType` fields as the `FieldCheck` value.

ReferenceTypeFields

For this option, *ReferenceTypeFields* is a plus-, space-, or comma-separated list of `ReferenceType` fields. If a query produces several results that contain the same value in one or more of the specified `ReferenceType` fields, Content returns only the most relevant result. If several results have the same relevance, the result with the highest `DocID` returns (unless you enable a `Sort` option that overrides this).

For example:

`http://IDOLhost:port/action=Query&Text=The Moonstone&Combine=DRETITLE`

In this example, if several results contain the same value in the `DRETITLE` field, it displays only the result that has the highest relevance to the query text.

When you combine using a `ReferenceType` field, Content automatically uses all the fields that you list in the configuration file in the same `PropertyFieldCSVs` parameter as this field. To ensure that Content combines using only the specified field, you must set up an individual field process to identify this field as a `ReferenceType` field.

If you want to use multiple `ReferenceType` fields to combine, you might want to create a process that identifies all these fields as `ReferenceType`.

Related Topics

- Configure a Field Process, on page 84

For example:

```
[SetupReferenceFields]
Property=ReferenceFields
PropertyFieldCSVs=*/DREREFERENCE,*/url

[CombineField1]
Property=ReferenceFields
PropertyFieldCSVs=*/DRETITLE

[CombineField2]
Property=ReferenceFields
PropertyFieldCSVs=*/CombineField
```

In this example, if you set `Combine` to **DREREFERENCE**, Content combines using the `DREREFERENCE` and `url` fields. If you set `Combine` to **DRETITLE**, it uses only the `DRETITLE` field to combine.

Multiple Options

You can combine the `Simple` and `FieldCheck` options, in which case you must specify `Simple` first. For example:

```
Combine=Simple+FieldCheck
```

If you set `Combine` to ***ReferenceTypeFields***, you cannot combine the fields with another `Combine` option.

Exceptions

In some cases, you might not want to combine documents. If you set `Combine` to **FieldCheck** or ***ReferenceTypeFields***, by default Content combines all documents that do not contain the field, and returns only one set.

To return these results without combining them, set the `CombineIgnoreMissingValue` configuration parameter to **True**. For example:

```
[Server]
CombineIgnoreMissingValue=True
```

FieldCheck Parameter

You can use the `FieldCheck` action parameter to return only documents whose `FieldCheckType` field matches the value you specify, for example a category name. You must identify a `FieldCheckType` field before you store content in the IDOL Content component.

If you set `UrlAnalysis` to **True** in the IDOL Content component configuration file, enter a domain name for `FieldCheck` to restrict results to documents that were aggregated from this domain. For example:

```
http://IDOLhost:port/action=Query&Text=A fast sports car&FieldCheck=Red
```

In this example, Content is configured to store the `Color` field as a `FieldCheckType` field. The query returns only those results whose content matches the specified `Text` and whose `FieldCheckType` field has the value `Red`.

Predict Parameter

You can use the `Predict` parameter to instruct the IDOL Content component to use statistical sampling to estimate the total number of results that are available. You must also set `TotalResults` to **True** for your query action. Prediction can increase the query speed.

If you set `Predict` to **False**, Content counts all results and in addition prints the number of results for each database. For example:

```
http://IDOLhost:IDOLport/action=Query&Text=A fast sports
car&TotalResults=True&Predict=True
```

In this example, if you have set `TotalResultsPredictionThreshold` to **100**, Content uses statistical sampling to estimate the total number of results that are available for the query. If fewer than 100 results are available for the query, Content returns the exact number of total results.

If you do not want Content to display the number of results for each database, you can set the `TotalResultsPrintDatabase` configuration parameter to **False**.

Store and Retrieve the Result State

Whenever you use the `Query`, `Suggest`, or `SuggestOnText` actions, you can save the result state (the `DocID` values of all returned results) in a *state token*. In subsequent actions, you can use this token to represent the documents from that result set.

A state token is an array of `DocID` values. Its name is of the form *ABCDEFGH–N*, where *ABCDEFGH* is a random alphanumeric string (generated at query time), and *N* is the number of document IDs that the token holds. You can also use document references, rather than document IDs, to create state tokens. In this case, use the `StoredStateField` parameter to specify the name of the field that contains the references.

When you have a state token, supply the token name to specify the entire set of documents (for example, `CWQ4FJ9LZSE5-6`). You can also use a (zero-based) array notation to specify a subset of the documents (for example, `CWQ4FJ9LZSE5-6[3]` to select the fourth document in the array).

Tokens in the array are in the same order as in the returned results.

Content stores tokens persistently, in the directory *installDir*`/agentstore/storedstate`.

Store the Result State

To create a state token from your query results, use the `StoreState` parameter when you issue the query:

`http://localhost:20000/action=Query&Text=apple&StoreState=True`

To use document references, rather than `DocIDs`, add the `StoredStateField` parameter:

`http://localhost:20000/action=Query&Text=apple&StoreState=True&StoredStateField=Ref`
`erence`

The returned results include a tag set that holds the token name, for example
`<autn:state>CWQ4FJ9LZSE5-6</autn:state>`.

If the query is likely to return a very large data set, you can optimize performance by not printing the query results (assuming that you want only the state token):

`http://localhost:20000/action=Query&Text=apple&StoreState=True&MaxResults=5000&Prin`
`t=noresults`

Query with the State Token

When you query the IDOL Content component, you can use the `StateID`, `StateMatchID`, and `StateDontMatchID` parameters to pass a state token. These parameters are similar to the `ID`, `MatchID`, and `DontMatchID` parameters that specify individual document IDs.

Action	State parameters supported
GetQueryTagValues	StateMatchID, StateDontMatchID
GetContent	StateID
List	StateMatchID, StateDontMatchID
Query	StateMatchID, StateDontMatchID
Suggest	StateID, StateMatchID, StateDontMatchID
SuggestOnText	StateMatchID, StateDontMatchID
Summarize	StateID
TermGetBest	StateID

If you use `StateMatchID` and `StateDontMatchID` in the same action, Content removes any documents in the `StateDontMatchID` list from the `StateMatchID` list.

Examples:

- **Retrieve all documents**. Use the `StateID` parameter to retrieve all documents in the token:

  `action=GetContent&StateID=CWQ4FJ9LZSE5-6`

- **Query only the specified documents**. Use the `StateMatchID` parameter to, for example, query only the first four documents in the stored result set:

  `action=Query&Text=pear&StateMatchID=CWQ4FJ9LZSE5-6[0-3]`

- **Query all but the specified documents**. Use the `StateDontMatchID` parameter to, for example, query all content except for the documents specified in the token:

  `action=Query&Text=pear&StateDontMatchID=CWQ4FJ9LZSE5-6`

> **NOTE:** A stored-state-aware DAH forwards any query that includes a state token to the IDOL Content component that originally created the token.

Use a State Token with Index Actions

You can use the `StateID` and `StateMatchID` parameters to pass a state token in the `DREDELETEDOC`, `DREEXPORTIDX`, `DREEXPORTXML`, `DREEXPORTREMOTE`, and `DRECHANGEMETA` actions.

Index action	State parameters supported
`DREDELETEDOC`	`StateID`
`DREEXPORTIDX`	`StateMatchID`
`DREEXPORTXML`	`StateMatchID`
`DREEXPORTREMOTE`	`StateMatchID`
`DRECHANGEMETA`	`StateID`

Examples:

- **Change document database**. Use the `StateID` parameter to assign all 100 documents identified by state token `CTBJ1V4QO4I5-100` (as well as the indexed documents with document IDs 1 and 2) to the `archive` database:

  `DRECHANGEMETA?Type=database&NewValue=archive&Docs=1+2&StateID=CTBJ1V4QO4I5-100`

- **Delete documents**. Use the `StateID` parameter to delete the first five documents identified by state token `CTBJ1V4QO4I5-100`:

  `DREDELETEDOC?StateID=CTBJ1V4QO4I5-100[0-4]`

- **Export documents**. Use the `StateMatchID` parameter to export the second, fourth, and sixth documents identified by state token `CTBJ1V4QO4I5-100`:

  `DREEXPORTXML?StateMatchID=CTBJ1V4QO4I5-100[1+3+5]`

Expire State Tokens

You can manually expire stored state tokens by using the `TokenManagement` action. For example:

`action=TokenManagement&TokenAction=Expire&StateID=B8UGI95FKJG-23`

This action expires the state token `B8UGI95FKJG-23` from the IDOL Content component.

For more information about the `TokenManagement` action, refer to the *IDOL Server Reference*.

Chapter 18: View Documents

The IDOL View component can display documents in a Web browser. This section describes the View service, which converts documents to HTML for viewing. View is primarily used to display result documents.

About the IDOL View component

The IDOL View component uses IDOL KeyView filters to convert documents into HTML format for display in a Web browser. It can convert locally stored documents, as well as documents from an intranet or Internet source. It can also retrieve the document in its original format.

View can convert documents to HTML from many different formats, including:

- word processing documents
- spreadsheets
- presentations
- graphics files
- CAD Drawings
- email messages

When you convert documents, you can select terms for highlighting. During the conversion process, View sends a `Highlight` action to the IDOL Content component. When the document is displayed, it includes highlighting for the specified terms.

Configure the IDOL View component

You can configure the IDOL View component by using the IDOL View component configuration file. Most settings that control document viewing are found in the `[Viewing]` section. You might also need to configure settings in the `[Paths]` section.

The `[Viewing]` section contains general settings that enable View to access documents to convert them to HTML. There are also settings that determine how View manages its cache.

For full details of the available settings for View, refer to the *IDOL Server Reference*.

Related Topics

- Display Online Help, on page 30

- Set up SSL for the IDOL View component, on page 396

Enable View to Access Documents

By default, View cannot access documents in the local file system. Additionally, by default, it cannot access UNC paths to Windows shared folders that contain dollar symbols ($). You must enable View to access documents in these directories by using the IDOL View component configuration file.

To enable View to access documents

1. Open the IDOL View component configuration file in a text editor.

2. Find the `[Viewing]` section, or create one if it does not exist.

3. Set the `ViewLocalDirectoriesCSVs` parameter to a comma-separated list of directories that contain documents that you want to allow the IDOL View component to access. You must list the full paths. For example:

   ```
   [Viewing]
   ViewLocalDirectoriesCSVs=C:\Shared,C:\PDFs
   ```

   View can access these directories and any subdirectories, and it can convert documents within these directories to HTML for display.

4. To allow the IDOL View component to access Windows shared folders that contain dollar symbols ($), you must set `ViewAllowedSpecialUNCPathsCSVs` to a comma-separated list of these paths. For example:

   ```
   ViewAllowedSpecialUNCPathsCSVs=C:\Shared$Files\
   ```

5. To allow the IDOL View component to access only the listed directories, set `RestrictToAllowedDirectories` to **True**. This parameter prevents View from accessing any directory or URL that is not listed. By default, only local disk access is restricted and View can access any (non-special) UNC path or Web URL.

6. Save and close the configuration file.

7. Restart the IDOL View component for your changes to take effect.

Configure View to Block Particular URLs or Hosts

By default, the IDOL View component attempts to access and process any URL or Web address that you send in the `Reference` action parameter for the `View` action. You can configure View to block certain URLs, host names or IP addresses. In this case, View does not process viewing requests for those URLs.

To configure View to block a URL or host

1. Open the IDOL View component configuration file in a text editor.

2. Find the `[Viewing]` section.

3. Decide which strategy to use to block URLs and hosts.

- To specify lists of URLs or hosts to block, and allow all others, set one or both of the following parameters.

`ViewBlockedURLsCSVs`	A comma-separated list of URLs that you want to block. You can use Wildcard values in the URLs. The IDOL View component matches the URL string, and does not retrieve files for any matching URLs. For example:
	
	`ViewBlockedURLsCSVs=*restart*,*example.com,1`
	`5.*.*.*`
	
	This example blocks any URL that contains the string `restart`, and also blocks any Web address that ends with `example.com`, or an IP address that starts with `15`. It does not block the IP address that `example.com` maps to, or the name of a host that uses an IP address that starts with 15.

`ViewBlockedHostCSVs`	A comma-separated list of IP addresses or host names that you want to block. View performs a host name lookup and blocks both the IP address and the host name. For example: `ViewBlockedHostCSVs=12.34.56.78,www.example.com` In this example, View blocks the IP address `12.34.56.78` and any host name that points to it. It also blocks `www.example.com` and the corresponding IP address.

- To specify lists of URLs or hosts to allow, and block all others, set one or both of the following parameters.

`ViewAllowedURLsList`	A comma-separated list of URLs that can be accessed to view documents. You can use wildcards. For example: `ViewAllowedURLsList=example.com/doc/*,10.1.*.*` This example would not allow access to a machine with an IP address `10.1.2.3` if a hostname was used in the URL instead of the IP address.
`ViewAllowedHostsList`	A comma-separated list of hosts, specified by host name or IP address, that can be accessed to view documents. For example: `ViewAllowedHostsList=example.com:8080,12.34.56.78`

> **TIP:** For more complex scenarios, you can specify lists of URLs and hosts to allow, and lists of URLs and hosts to block. If a URL matches both a blocked list and an allowed list, it is blocked.

4. Save and close the configuration file.

5. Restart the IDOL View component for your changes to take effect.

Configure View to Use a Proxy Server

To allow the View service to access HTTP documents using a proxy server, you must configure the details for the proxy server in the IDOL View component configuration file.

To configure a proxy server for the View service

1. Open the IDOL View component configuration file in a text editor.

2. Find the `[Viewing]` section.

3. Set `ProxyHost` to the IP address or host name of your proxy server. For example:

   `ProxyHost=proxy.example.com`

4. Set `ProxyPort` to the port that View must use to communicate with the proxy server. For example:

   `ProxyPort=8080`

5. If the proxy server requires NTLM authentication, set `NTLMProxy` to **True**. For example:

   `NTLMProxy=True`

6. If the proxy server requires a user name and password, set `ProxyLogin` and `ProxyPassword`. For example:

   `ProxyLogin=idolview`
   `ProxyPassword=9szJwMkD28A`

7. If you set `NTLMProxy` to **True**, you must set `ProxyLogin` to a fully qualified NTLM login. For example:

   `ProxyLogin=mydomain\\myusername`

8. Set `NoProxyHostsCSVs` to a comma-separated list of host names for which the View service does not use the Proxy server. You can use Wildcard values. For example:

   `NoProxyHostsCSVs=localhost,*.companyname.com`

9. Save and close the configuration file.

10. Restart the IDOL View component for your changes to take effect.

Configure View to Use a Distributed Connector or IDOL NiFi Ingest

When you set up a system to view documents that you have indexed into the IDOL Content component, you need a method to retrieve the original document to convert for viewing. One method for doing this is to use Distributed Connector or IDOL NiFi Ingest.

Distributed Connector is a central connector that manages operations across the connectors that retrieve documents from your repositories. Similarly, IDOL NiFi Ingest is a central system that manages your connectors, using the Apache NiFi Framework.

In both cases, the Connectors each have access to the document source repository for indexing, so you can use this access for viewing as well. This method simplifies authentication and retrieval for your repositories, and does not require a secondary store of documents for the IDOL View component.

This method simplifies authentication and retrieval from your data repositories. However, it requires more network transfer than other viewing methods.

> **NOTE:** For viewing through Distributed Connector or NiFi to work, the connectors must recognize the document reference that you use. Micro Focus recommends that you use the `AUTN_IDENTIFIER` field as the reference, which all connectors use.

To configure View to use Distributed Connector or IDOL NiFi Ingest

1. Open the IDOL View component configuration file in a text editor.

2. Find the `[Viewing]` section.

3. Set the following parameters in the `[Viewing]` section:

`DistributedConnectorHost`	The host name where the Distributed Connector or NiFi component runs.
`DistributedConnectorPort`	The ACI port for the Distributed Connector, or the port that you have configured the Connector ACI group in NiFi to listen on.

   When you send a `View` action, View checks whether the requested reference is a Web URL or a local file. If it is, View Server retrieves it as normal. If it does not recognize the reference, View sends it to the configured Distributed Connector or NiFi Ingest for retrieval.

4. Save and close the configuration file.

5. Restart the IDOL View component for your changes to take effect.

You can use this method only for connectors that support the `View` action.

For Distributed Connector, you must also register the connector in the Distributed Connector, and use a connector group configuration to identify your connectors. For more information about the connector configuration, refer to *IDOL Expert*.

Configure Universal Viewing

Universal viewing is an additional View configuration that allows the IDOL View component to work out how to retrieve your original documents. It allows you to retrieve documents from various sources to view, without requiring you to use complex logic in your front end components to work out how to send the View request.

> **NOTE:** Universal viewing requires more internal communication between your IDOL components, which might affect performance.

When you enable universal viewing, you also configure the location of your document store (an IDOL Content component or DAH). You also configure a set of reference fields to specify how to retrieve the original documents (by using Distributed Connector or IDOL NiFi Ingest, a Web URL, or from the file system).

To enable universal viewing

1. Open the IDOL View component configuration file in a text editor.

2. Add a `[UniversalViewing]` configuration section.

3. In the `[UniversalViewing]` section, set `Enabled` to **True** to enable universal viewing.

4. Set `DocumentStoreHost` and `DocumentStorePort` to the host and port of your primary data index component (either the Content component, or DAH). View Server uses this component to retrieve the index representation of your documents to find the appropriate reference fields.

5. To use IDOL NiFi Ingest for viewing, set `ConnectorGroupField` to the name of a document field that contains the connector group for a document.

6. Set one or more of the following parameters to specify the reference fields in your documents that View Server must use to retrieve the documents from the associated sources:

`DistributedConnectorReferenceField`	The document reference field that contains a reference that View Server can use to retrieve the document by using the Distributed Connector or IDOL NiFi Ingest. To use this option, you must configure the details for the Distributed Connector or NiFi in the `[Viewing]` section. See Configure View to Use a Distributed Connector or IDOL NiFi Ingest, on page 364.
`FileSystemReferenceField`	The document reference field that contains a reference value that View Server can use to retrieve the document from a local file system or network share.
`WebURLField`	The document reference field that contains a reference value that View Server can use to retrieve the document from the Web.

7. Save and close the configuration file.

8. Restart the IDOL View component for your changes to take effect.

For example:

```
[UniversalViewing]
Enabled=True
ConnectorGroupField=CONNECTOR_GROUP
DocumentStoreHost=localhost
DocumentStorePort=9900
```

```
DistributedConnectorReferenceField=AUTN_IDENTIFIER
WebURLField=DREREFERENCE
FileSystemReferenceField=ORIGINAL_LOCATION
```

When you send a View request, the IDOL View component uses the requested reference to retrieve the index representation of the document from the document store. It then attempts to find the configured reference fields in the document:

- If the document has the configured `DistributedConnectorReferenceField`, View attempts to retrieve the document by using the Distributed Connector or IDOL NiFi Ingest. For this option, you must also configure a Distributed Connector or NiFi in the `[Viewing]` section.

- If the document has the configured `WebURLField` (and does not have the `DistributedConnectorReferenceField`), View Server attempts to retrieve the document from the Web.

- If the document has the configured `FileSystemReferenceField` (and does not have the `DistributedConnectorReferenceField` or `WebURLField`), View attempts to retrieve the document directly from a file system or network share. Your `[Viewing]` configuration must allow View to retrieve the file.

If the document does not contain any of the configured reference fields, View attempts to retrieve the original document by the reference in the request, as usual. If it cannot retrieve the original document by reference, View returns the document title and content from the document store.

Configure View to Highlight Terms

You might need to add additional parameters to your configuration file to use View to highlight link terms in returned documents.

To configure View to highlight link terms

1. Open the IDOL View component configuration file in a text editor.

2. Find the `[Viewing]` section.

3. Set the following parameters in the `[Viewing]` section:

IDOLHost	The host name where the IDOL Content component runs.
IDOLPort	The value of the ACI port for this IDOL Content component.
IDOLTimeout	The amount of time (in seconds) that View waits for a response from the IDOL Content component.

   These parameters determine where View sends `Highlight` actions. You must configure these parameters if you use View in a stand-alone configuration. If you use a unified IDOL Server, View uses the IDOL Content component in the IDOL Server, but you can configure these parameters if you want to use a different IDOL Content component.

   For information on using stand-alone components, refer to the *IDOL Getting Started Guide*.

4. Set `HighlightChunkSize` to the maximum size (in bytes) of the chunks that View splits text

into before highlighting. Enter `–1` for an unlimited chunk size.

5. Set `DefaultLanguageType` to the language type that View uses for the link terms when highlighting. This parameter ensures that the IDOL Content component stems the terms correctly when highlighting. You can override this parameter by using the `LanguageType` parameter in the `View` action. See .

6. Set `DefaultStartTag` and `DefaultEndTag` to the default opening and closing HTML tags to use to highlight link terms.

7. Save and close the configuration file.

8. Restart the IDOL View component for your changes to take effect.

Configure View to Embed Images

By default, the View service converts any images to links when it converts a document to HTML. When your browser displays the HTML document, it retrieves the image by accessing the link in a subsequent request. Alternatively, you can configure the View service to embed each image as base64-encoded data in the HTML output.

Embedding the images in this way means that View must process only a single action for the entire document, rather than processing the subsequent requests for images. This process means that you can use a load balancer to process `View` requests, without having to ensure that the image requests go to the correct View component.

To configure View to embed images

1. Open the IDOL View component configuration file in a text editor.

2. In the `[Viewing]` section, set `EmbedImages` to **True**. For example:

    `EmbedImages=True`

3. Save and close the configuration file.

4. Restart the IDOL View component for your changes to take effect.

You can override the `EmbedImages` configuration parameter for an individual `View` action by using the `EmbedImages` action parameter. This action parameter uses the configuration setting as its default value.

Configure View to Use Original URLs

By default, the View service converts any URLs that occur in the HTML and CSS background attributes in input documents into `View` requests. Alternatively, you can configure the View service to use the original links in the output document. In this case, View adds the `base` tag (with the `href` attribute set) to the beginning of the output, so that relative URLs resolve back to the original Web server.

To configure View to use the original internal links

1. Open the IDOL View component configuration file in a text editor.

2. In the [Viewing] section, set OriginalBaseURL to **True**. For example:

   OriginalBaseURL=True

3. Save and close the configuration file.

4. Restart the IDOL View component for your changes to take effect.

You can override the OriginalBaseURL configuration parameter for an individual View action by using the OriginalBaseURL action parameter. This action parameter uses the configuration setting as its default value.

Configure View to Retrieve RMS Protected Files

You can configure View to access and decrypt files that are protected by Microsoft Rights Management System (RMS).

To enable RMS decryption

1. Open the IDOL View component configuration file in a text editor.

2. In the [Viewing] section, set DocumentDecryptionSettings to the name of a configuration section where you define the decryption settings. For example:

   DocumentDecryptionSettings=Azure

3. Create a new configuration section with this name.

4. Set RMSClientId and RMSClientSecret to the client ID and secret that View must use to identify itself to Microsoft Azure.

5. Set RMSUserName and RMSUserPassword to the user name and password to use to log in to Microsoft Azure to retrieve documents.

6. Set any other parameters for your system, such as proxy or SSL settings. For more information, refer to the *IDOL View component Reference*.

7. Save and close the configuration file.

8. Restart the IDOL View component for your changes to take effect.

For example:

```
[Viewing]
DocumentDecryptionSettings=Azure

[Azure]
RMSClientId=abc123de-45fg-678h-a90b-cd12ef34gh56
RMSClientSecret=abc!@-1234567890-dEF!@-abcdef123456
RMSUserName=myusername@myusername.onmicrosoft.com
RMSUserPassword=passwORd!
```

Configure the View Cache

When the View service converts a document to HTML, it caches the results. If it receives a request for the same document again, it retrieves the version from the cache, rather than converting it again.

You can use an internal View cache, or you can cache documents on a memcached server instance. For details about memcached servers, refer to http://memcached.org.

You can also share a cache between multiple View services.

Configure the Internal View Cache

By default, the View service uses an internal cache, stored on disk. You can configure how this cache behaves.

To configure the internal View cache

1. Open the IDOL View component configuration file in a text editor.

2. In the `[Viewing]` section, set `CacheType` to **Internal**. For example:

   `CacheType=Internal`

3. In the `[Viewing]` section, set `CacheExpirySeconds` to the length of time (in seconds) that you want to keep the cache. Use a large value to ensure that the View service does not have to regularly convert the same documents. For example:

   `CacheExpirySeconds=604800`

4. In the `[Paths]` section, set `ViewCacheDirectory` to the directory in which you want the View service to store the cache. The View service must have write access to this directory. For example:

   `ViewCacheDirectory=C:\IDOL\IDOLServer\IDOL\view\Cache`

5. Save and close the configuration file.

6. Restart the IDOL View component for your changes to take effect.

Use a Memcached Server View Cache

The View service can store cache documents in a memcached server instance.

> **NOTE:** By default, the memcached program has a maximum entry size limit of 1 megabyte, so it cannot process files bigger than 1 megabyte.

To configure View to use a memcached cache

1. Open the IDOL View component configuration file in a text editor.

2. Find the `[Viewing]` section.

3. Set `CacheType` to **Memcached**. For example:

   `CacheType=Memcached`

4. Set `CacheServers` to a comma-separated list of *host*:*port* pairs, where:

host	is the host name or IP address of the memcached server.
port	is the port that View must use to contact the memcached server.

For example:

```
CacheServers=localhost:11211,cacheserver:21532
```

5. Save and close the configuration file.

6. Restart the IDOL View component for your changes to take effect.

Use a Shared View Cache

You can share a View cache between multiple IDOL View component. To use a shared cache, you must have a memcached server instance that each IDOL View component can access. This memcached server stores information about where to find the converted cache documents, so that each IDOL View component can retrieve them.

When you use a shared cache, the `ResetCache` action and automatic cache expiration are unavailable. The `CacheExpirySeconds` parameter specifies only how often to delete old, duplicate job files, but it does not automatically delete the newest instance of a job.

To use a shared View cache

1. Open the IDOL View component configuration file in a text editor.

2. Find the `[Viewing]` section.

3. Set `CacheType` to **Internal**. For example:

```
CacheType=Internal
```

4. Set `SharedCache` to **True**. For example:

```
SharedCache=True
```

5. Set `CacheServers` to a comma-separated list of *host*:*port* pairs, where:

host	is the host name or IP address of the memcached server.
port	is the port that View must use to contact the memcached server.

6. In the `[Paths]` section, set `ViewCacheDirectory` to the directory to use to store the cache. For example:

```
ViewCacheDirectory=C:\View\Cache
```

7. Save and close the configuration file.

8. Restart the IDOL View component for your changes to take effect.

9. Repeat Step 1 to Step 7 for each IDOL View component that must share the cache.

Configure View to Redact Documents

You can optionally configure View to redact certain items in the HTML output that it produces. View uses IDOL Eduction to location entities (snippets of information) and redacts them in the output file, rather than returning the original value.

For example, if you want to redact personally identifiable information (PII) such as names, you can configure View to find names in the document and hide them before it returns the HTML file.

> **IMPORTANT:** To use document redaction, your license must allow you to use Eduction and the particular grammar categories that you want to use (such as PII).

When you use a View cache, View saves the redacted version in the cache so that any further requests for the same document are also redacted.

To configure View to redact entities

1. Open the IDOL View component configuration file in a text editor.

2. In the `[Viewing]` section, set `DocumentRedactionSettings` to the name of a configuration section where you configure the Eduction settings. For example:

   ```
   DocumentRedactionSettings=Redaction
   ```

3. Create a new configuration section with the name that you set.

4. Set the appropriate Eduction configuration parameters to use the grammars and entities that find values that you want to redact, as well as any additional settings for Eduction. For more information, refer to the *Eduction User and Programming Guide*. For example:

   ```
   [Redaction]
   ResourceFiles=C:\PII\names.ecr,C:\PII\address.ecr
   ```

5. Save and close the configuration file.

6. Restart the IDOL View component for your changes to take effect.

Distribute View Servers

When you have multiple IDOL View components set up to process `View` requests, you can use the Distributed Action Handler (DAH) to distribute requests between the servers.

To distribute IDOL View components with a DAH, you must configure the DAH to distribute the `View` actions by reference. The DAH distribute by reference mode ensures that it always sends repeated requests for the same document to the same View child server. This process ensures that each document is cached in only one IDOL View component, and the cache is used in subsequent requests for the same document.

DAH distribution by reference is available only for the `View`, `GetLink`, and `ViewGetDocInfo` actions. When you upload documents to view by using the `View` action with the `ViewUpload` parameter, you can add the `Reference` parameter to ensure consistent routing.

> **NOTE:** If you do not configure DAH in distribute by reference, images in converted documents might fail to load.

For more information about this method, refer to the *Distributed Action Handler Administration Guide*.

View Documents

To convert a document into HTML, use the `View` action. For example:

`http://`*ViewHost*`:`*ViewPort*`/action=View&Reference=`*DocumentReference*

where:

ViewHost	is the name or IP address of the host on which the IDOL View component runs.
ViewPort	is the IDOL View component ACI port.

DocumentReference	is the full reference of the document that you want to view. The reference can be:
	
	• A document in the local file system, for example:
	
	  `Reference=C:\Documents\report.doc`
	
	• A document on an intranet site, for example:
	
	  `Reference=//intranet/report.doc`
	
	• A Web page or document accessible on the Internet, for example:
	
	  `Reference=http://news.bbc.co.uk`
	
	> **NOTE:** You can also upload a file to view. See View an Uploaded Document, on page 375.

For a full list of parameters available for the `View` action, refer to the *IDOL Server Reference*.

View the Document Directly in the Web Browser

By default, the document returns in the standard ACI XML format. The document is written as a string of base64-encoded data. If you pass the results to a client application for further processing, you might want the document in this format.

To view the document directly in your Web browser, set the `NoACI` parameter to **True** in the `View` action. This returns the document in a format that the Web browser can read, so that you can view the HTML directly.

For example:

`http://localhost:9080/action=View&Reference=C:\PDFs\report.pdf&NoACI=True`

Use IDOL Admin to View Documents

As an alternative to using the View action, you can use the IDOL Admin interface to convert documents into HTML to display in a Web browser.

To use IDOL Admin to view a document in your Web browser

1. In the **Control** menu, click **View**.

2. Type the document reference into the text box. The reference can be:
   - a document in the local file system, for example:
     ```
     C:\Documents\report.doc
     ```
   - a document on an intranet site, for example:
     ```
     //intranet/report.doc
     ```
   - a Web page or document accessible on the Internet, for example:
     ```
     http://news.bbc.co.uk
     ```

3. If your document references have a common format, you can configure a standard prefix in the IDOL View component configuration file to add to each document reference. If you select the **Use reference prefix** check box, IDOL Admin adds this prefix to the View action automatically.

   For more information on using the ReferencePrefix and UseReferencePrefix parameters, refer to the *IDOL Server Reference*.

4. Click **View**.

   The document opens in your Web browser.

View the Latest Version of a Document

The IDOL View component stores converted documents in the cache. It stores them for a length of time determined by one of the following configuration parameters:

- CacheExpirySeconds for non-secured documents

- SecureCacheExpirySeconds for secured documents

If a user requests a document that exists in the cache, View retrieves it from the cache to improve performance.

You can set the IgnoreCache parameter in a View action to convert the document again, rather than retrieving the cached version. For documents that change frequently, this parameter ensures that View returns the most recent form of the document. For example:

```
http://localhost:9080/action=View&Reference=C:\PDFs\report.pdf&IgnoreCache=True
```

> **TIP:** You can delete all the files from the View cache directory by using the **Reset View Cache** feature in the Controls section of the **Overview** tab in the Status page of the IDOL Admin interface. For more information, refer to the *IDOL Admin User Guide*.

View an Uploaded Document

You can view a document that the IDOL View component cannot access directly on a Web site or file system, by uploading the file in the `View` action. To upload a document, you must send the `View` action as a POST request, and upload the document as `multipart/form-data` in the `ViewUpload` parameter.

For example:

```
<form name="example" method="post" enctype="multipart/form-data"
action="http://localhost:14000/a=view&output=html&noaci=true" target="view_output">
View This File:(input type="file" name="viewupload" />
     <br><br>
     Highlight: <input type="text" name="links" />
     <br><br>
     <input type="submit" value="Submit" />
</form>
```

This example shows a POST request for the `View` action, including the usage of `ViewUpload`.

Highlight Terms

When you convert documents to HTML you can highlight link terms in the returned documents.

To highlight terms in a View action

1. Set the `Links` parameter in a `View` action to the name of the terms that you want to highlight.

2. Set the `StartTag` and `EndTag` parameters to the HTML tags to use to highlight the term.

For example:

```
action=View&Reference=C:\Documents\report.doc&NoACI=True&Links=price&StartTag=<font
color="red">&EndTag=</font>
```

When the IDOL View component receives this action, it sends the document text to the IDOL Content component as part of a `Highlight` action. For example:

```
action=Highlight&Text=<document text>&Links=price&StartTag=<font
color="red">&EndTag=</font>
```

The response from Content contains the specified HTML tags around the link terms, which View incorporates into the HTML conversion of the document.

In this example, View converts `report.doc` to HTML for viewing directly by the Web browser. In the returned document, all instances of the term `price` are highlighted using the HTML tags `<font color="red">` and `</font>`. For example:

```
In some cases the <font color="red">price</font> has risen considerably.
```

When you view this document in a Web browser, these terms display in red text.

Highlight Boolean Expressions

To use Boolean expressions in your link terms, set the `Boolean` parameter to **True** in the `View` action. This process is similar to setting the `Boolean` parameter to **True** for a `Highlight` action. The IDOL Content component then treats terms in the `Links` parameter as query text, rather than a list of terms. For example:

```
action=View&Reference=C:\Documents\report.doc&NoACI=True&Boolean=True&Links=price
AND risen&StartTag=<font color="red">&EndTag=</font>
```

This action highlights the terms `price` and `risen`, if both terms appear in the document.

If you use a proximity expression in `Links`, Content highlights only instances of the terms that meet the proximity requirements. You can also set the `Highlight` parameter to **Proximity** to highlight the whole proximity expression as a span, rather than highlighting each term individually.

Highlight Expressions in Different Languages

You can specify the language type of the text in the `Links` parameter by adding the `LanguageType` parameter to the `View` action. This setting ensures that the IDOL Content component stems words correctly before highlighting.

Use this parameter if the language type of the text to highlight is not the same as the language type you set in the `DefaultLanguageType` configuration parameter. For example:

```
action=View&Reference=C:\Documents\report.doc&NoACI=True&Links=price&StartTag=<font
color="red">&EndTag=</font>&LanguageType=EnglishUTF8
```

Highlight Multiple Link Terms

You can use the `MultiHighlight` action parameter to highlight different link terms with different HTML tags.

To highlight multiple terms with different HTML tags

1. In the `View` action, set `MultiHighlight` to **True**.

2. Set `Links` to a semicolon-separated list of terms or groups of terms.

3. Set `StartTag` and `EndTag` to a semicolon-separated list of opening and closing HTML tags respectively. The first pair of tags apply to the first link term, the second pair of tags apply to the second link term and so on.

For example:

```
action=View&MultiHighlight=True&Links=dog+OR+cat;rabbit;"apples and
pears"&StartTag=<b>;<i>;<font color="red">&EndTag=</b>;</i>;</font>&Boolean=True
```

This action highlights the terms **dog** or **cat** in bold, the term **rabbit** in italic, and the phrase **apples and pears** in red.

Specify Document Processing

When you send a `View` action, use the `OutputType` parameter to specify the way that the IDOL View component processes the document. Set `OutputType` to one of the following values:

HTML	Converts the document to HTML. It also modifies relative or inline URL links with a prefix specified by the `URLPrefix` parameter.
ReplaceURLs	Modifies all relative or inline URL links with a prefix specified by the `URLPrefix` action parameter. If you do not specify a `URLPrefix`, View uses the value of the `DefaultURLPrefix` configuration parameter. This output type does not convert the document to HTML.
Raw	Returns the original document without any conversion or modification. In this case, highlighting is not available, unless the original document is in HTML format.
Redirect	Directs the client to the original Web site to display the original document. In this case, highlighting is not available.

By default, `OutputType` is set to:

- `ReplaceURLs` when the document reference is a URL with the prefix `http:`, `https:`, `msx:`, or `notes`.

- `HTML` for all other documents.

View Document Information

You can use the `ViewGetDocInfo` action to return information about the `View` action for a document. For example, this action returns the document type and the number of pages that result from the KeyView HTML conversion.

The `ViewGetDocInfo` action accepts all `View` action parameters. It returns information for the particular `View` action that you send. For example, if you specify the `Links` parameter, `ViewGetDocInfo` returns information about which pages contain highlighted terms in the converted document.

View Templates

The View service can apply *templates* when it converts documents to HTML, which alter the appearance of the returned document. These templates are text files with `.ini` file extensions, which contain section names in square brackets, and key=*value* pairs. For example:

```
[KVHTMLOptionsEx]
OutputCharSet=KVCS_UTF8
bUseDocumentColors=True
```

There are several basic templates installed with the IDOL View component in the IDOL Server installers. By default, these are found in:

`installDir\common\keyview\templates`

If you move the templates folder, you must edit the value of the `ViewingTemplatesPath` configuration parameter in the `[Paths]` section of your IDOL View component configuration file.

The following table lists the provided templates.

View service templates

Template	Description
k2lowband.ini	This template is useful when you need to provide information to a mobile workforce that might not always have access to fast connections. • Creates a single HTML file. • Creates a table of contents at the top of the HTML document. • Suppresses embedded graphics from the source document.
k2lowbandunix.ini	Similar to k2lowband.ini except that it saves vector graphics in a vector format to view using a Java applet.
k2frames.ini	• Divides word processing documents, spreadsheets, and presentations into multiple files according to the heading levels in the document. • Creates two frames. The table of contents (based on source document heading levels and page breaks) appears in the left frame, and the HTML files appear in the right frame. • Inserts **Previous** and **Next** links at the end of each block.
k2framesunix.ini	Similar to k2frames.ini except that it saves vector graphics in a vector format to view using a Java applet.
k2onefile.ini	• Creates a single HTML file. • Creates a table of contents at the top of the HTML document. • Lists all metadata (Title, Subject, Author, Comments, and so on). • Converts graphics to JPEG with 640 x 640 resolution.
k2pdfonefile.ini	Similar to k2onefile.ini except that it converts graphics to JPEG and preserves the original resolution.
k2onefileunix.ini	Similar to k2onefile.ini except that it saves vector graphics in a vector format to view using a Java applet.

Apply a Template to a Document

To view a document with one of these templates, add the `ViewTemplate` parameter to the `View` action. For example:

```
action=View&Reference=C:\PDFs\report.pdf&NoACI=True&ViewTemplate=k2frames.ini
```

Apply a Default Template to All Documents

To define a default template to use for all documents in all `View` actions, set the `DefaultTemplate` parameter in the `[Viewing]` section of your IDOL View component configuration file.

```
[Viewing]
DefaultTemplate=k2onefile.ini
```

Modify the HTML Output for Documents

You can edit the supplied templates to change the appearance of the HTML output. Some basic configuration parameters are discussed in this section. All available sections and parameters are fully described in the *KeyView HTML Export SDK C and COM Programming Guide*. These are advanced parameters that can improve the fidelity and accuracy of the HTML output. They are for users who are already familiar with using template files.

To modify the HTML output

1. Open the template file that you want to update, or create a new template file with any other options that you want to use.

2. Add a `[KVHTMLConfig]` configuration section.

3. Add the parameters that you require (the parameters are described in the following sections). For example:

   ```
   [KVHTMLConfig]
   KVCFG_INCLREVISIONMARK=True
   KVCFG_LOGICALPDF=LPDF_RTL
   KVCFG_SETTEXTROTATE=True
   KVCFG_BLANKPICTURE=True
   ```

4. Save and close the template file.

5. To use the template, send a `View` action with `ViewTemplate` set to the name of the template file with these modifications. For example:

   ```
   action=View&Reference=C:\PDFs\report.pdf&NoACI=True&ViewTemplate=MyTemplate.
   ini
   ```

Modify the HTML Output for PDF Files

You can modify the HTML output from the conversion of PDF files by setting the following parameters in the [KVHTMLConfig] section of the template file.

Template configuration options for PDF files

Parameter	Description
KVCFG_SETHIFIPDF	Converts each page of a PDF document to a JPEG file, which provides a high-fidelity conversion of the document. By default, PDF files are converted to HTML.
KVCFG_ SUPPRESSTOCPRINTIMAGE	Prevents the use of bookmarks in a PDF file to generate a table of contents in the HTML output (the default behavior).
KVCFG_DELSOFTHYPHEN	Removes soft hyphens in the source document and joins the hyphenated words in the HTML output. By default, soft hyphens are maintained.
KVCFG_LOGICALPDF	Determines the order in which to extract paragraphs from a PDF. By default, the IDOL View component extracts PDF paragraphs in the order in which they are stored in the file (unstructured reading order), not the order in which they appear on the visual page (logical reading order). You can specify the following paragraph directions: • **LPDF_LTR**. Logical reading order and left-to-right paragraph direction. Use this option when most of your documents are in a language that uses left-to-right reading order, such as English or German. • **LPDF_RTL**. Logical reading order and right-to-left paragraph direction. Use this option when most of your documents are in a language that uses right-to-left reading order, such as Hebrew or Arabic. • **LPDF_AUTO**. Logical reading order. The PDF reader determines the paragraph direction for each PDF page, and then sets the direction accordingly. This option is used when no paragraph direction is specified. • **LPDF_RAW**. Unstructured paragraph flow. This value is the default behavior. If you enable logical reading order, and you want to return to an unstructured paragraph flow, set this flag.
KVCFG_SETTEXTROTATE	Displays rotated text in a file at zero degrees at the bottom of the page that it appears on. The IDOL View component enlarges the page to accommodate the text. By default, rotated text in a file is displayed in its original position, at the original font size, and at zero degrees rotation. Because the text is the

Template configuration options for PDF files, continued

Parameter	Description
	original size, but might be displayed in a smaller space, the text might overlap adjacent text in the HTML output. Use the `KVCFG_ SETTEXTROTATE` option to avoid this problem. HTML markup does not support text rotation.
`KVCFG_ SETPDFINVISTEXTTOGGLE`	Displays a JavaScript button in the upper right corner of the exported page, which you can click to switch between invisible and normal text. When you turn on invisible text, the invisible text is displayed and the normal content is hidden; when you turn off invisible text, the invisible text is hidden.
`KVCFG_ SETPDFINVISTEXTOPACITY`	Displays invisible text with an image in the PDF. Invisible text often occurs in PDF documents when the PDF software processes rasterized images through optical character recognition and then inserts the text in the PDF. You might want to display both the invisible text and the rasterized image. Set the invisible text *opacity* to an integer between 0 and 100, where 0 hides the invisible text and 100 displays it fully.

Related Topics

Hide Graphics

To hide graphics when you display a document, but still maintain the text flow of the original document, set the `KVCFG_BLANKPICTURE` parameter to **True** in the `[KVHTMLConfig]` section of the template file. KeyView does not convert graphics in a document, but it generates an image tag with an empty `src` attribute, creating an empty placeholder for the graphic. For example:

```
<img src="" height="136" width="101">
```

This parameter applies only to word processing formats, and is disabled by default.

To hide graphics completely, and exclude the image tags from the converted HTML document, set the `bNoPictures` parameter to **True** in the `[KVHTMLConfig]` section of the template file. This parameter is disabled by default.

Related Topics

Show Revised Content and Revision Information

You can show content that was deleted from a document with revision tracking enabled by setting the KVCFG_INCLREVISIONMARK parameter to **True** in the [KVHTMLConfig] section of the template file. When you enable this parameter, revision information (revision title, reviewer name, and revision date and time) for deletions and insertions of content display in a tooltip.

To reset the parameter and exclude deleted content and revision information from the HTML output, set the parameter to **False**. The default is False.

Format Revised Content

You can display revised content with a specified title and revision information. By default, the title is either the text string *inserted:* or *deleted:*, and it includes the reviewer name, and date and time.

You can also define a unique HTML style (such as, color: red; background: orange) to apply to the modifications for each reviewer. This allows you to easily differentiate between edits from multiple reviewers. For example, changes made by JSmith are highlighted in red, changes made by RBrown are highlighted in blue, and so on.

For example, you can configure View server to generate the following markup for inserted text:

```
<ins style="color: red" title="Inserted: JohnD, 2006-04-24Tl4:47:00"
cite="mailto:JohnD" datetime="2006-04-24T14:47:00">This text was added</ins> in a
previous version.
```

This text is displayed in the browser as:

This text was added in a previous version.

When you hover the cursor over the underlined text in the browser, the text "**Inserted: JohnD, 2006-04-24Tl4:47:00**" is displayed as a tooltip.

Use the following parameters to define the display of revised content.

Parameter	Description
KVCFG_ INSTITLEPREFIX	Specifies a string to include at the beginning of the tooltip for insertions. By default, the string *inserted:* is included.
KVCFG_ INSTITLEFLAG	Specifies whether to display the reviewer name and the revision date and time in the tooltip for insertions. The following values are available: • **RMT_OFF**. Does not display the information with the insertion. • **RMT_AUTHOR**. Displays the name of the reviewer who made the insertion. • **RMT_DATETIME**. Displays the date and time of the insertion. • **RMT_AUTHORDATETIME**. Displays the reviewer, date, and time of the insertion. This is the default.
KVCFG_	Specifies a string to include at the beginning of the tooltip for deletions. By

Parameter	Description
DELTITLEPREFIX	default, the string *deleted:* is included.
KVCFG_ DELTITLEFLAG	Specifies whether to display the reviewer name and the revision date and time in the tooltip for deletions. The following values are available: • **RMT_OFF**. Does not display the information with the deletion. • **RMT_AUTHOR**. Displays the name of the reviewer who made the deletion. • **RMT_DATETIME**. Displays the date and time of the deletion. • **RMT_AUTHORDATETIME**. Displays the reviewer, date, and time of the deletion. This is the default.
KVCFG_ AUTHORSTYLE*N*	Specifies a list of HTML style attributes to apply to the changes for each reviewer.

For example:

```
[KVHTMLConfig]
KVCFG_INCLREVISIONMARK=True
KVCFG_INSTITLEFLAG=RMT_AUTHOR
KVCFG_INSTITLEPREFIX=New:
KVCFG_DELTITLEFLAG=RMT_AUTHOR
KVCFG_DELTITLEPREFIX=deleted:
KVCFG_AUTHORSTYLE0=color: red; background: white
KVCFG_AUTHORSTYLE1=color: yellow; background: blue
KVCFG_AUTHORSTYLE2=color: green; background: black
```

This configuration:

- enables revision marks.

- displays only the name of the reviewer in the tooltip.

- adds the string *New:* to the beginning of the tooltip.

- defines styles to use for each reviewer.

Related Topics

Show Hidden Content

Microsoft Word, Excel, or PowerPoint documents contain hidden information, some of which is shown by default when exported and some of which is hidden by default. There are several options that allow you to determine exactly which types of hidden data to export.

Hidden Content in Microsoft Documents

You can display four types of hidden data from Microsoft Word, Excel, and PowerPoint documents, each of which has a corresponding parameter, which you can change to determine whether to show the hidden data. The table below lists each data type, its default behavior, and its corresponding configuration parameter.

Hidden data settings

Hidden data type	Default behavior	Configuration API flag
Microsoft Word		
Comments	Shown[1]	`KVCFG_WP_NOCOMMENTS`
Hidden text	Hidden	`KVCFG_WP_SHOWHIDDENTEXT`
Date field codes	Hidden	`KVCFG_WP_SHOWDATEFIELDCODE`
File name field codes	Hidden	`KVCFG_WP_SHOWFILENAMEFIELDCODE`
Microsoft Excel		
Hidden information	Hidden	`KVCFG_SS_SHOWHIDDENINFOR`
Comments	Hidden	`KVCFG_SS_SHOWCOMMENTS`
Formulas	Hidden	`KVCFG_SS_SHOWFORMULA`
Microsoft PowerPoint		
Hidden slides	Shown	`KVCFG_PG_HIDEHIDDENSLIDE`
Comments	Shown[2]	`KVCFG_PG_HIDECOMMENT`
Comments slide	Hidden	`KVCFG_PG_SHOWCOMMENTSSLIDE`[3]
Slide notes	Hidden	`KVCFG_PG_SHOWSLIDENOTES`

Related Topics

- Modify the HTML Output for Documents, on page 379

[1]Shown by default in Microsoft Word 97 to 2003 documents.

[2]Shown by default in Microsoft PowerPoint 97 to 2000 documents.

[3]This setting affects PowerPoint 2003 and 2007 only.

Part V: Administration and Maintenance

This section explains how to administer your IDOL Server installation and how to perform routine maintenance.

- Set up Security
- Add Users to IDOL Server
- Mail
- Administer IDOL Server
- Back up IDOL Server
- Troubleshoot IDOL Server

Chapter 19: Set up Security

This section describes the process of applying security settings to documents, and the process of setting up an SSL connection between IDOL components.

Set up Security on Documents

You can apply custom security settings to the documents that you index into the IDOL Content component. To do this, you identify fields in these documents that determine the security settings that are appropriate for each document.

Alternatively, you can specify the security property of a document every time that you index it by sending an additional parameter with the index action.

For more details on the settings that the `[Security]` section can contain and on how you can configure them, refer to the *IDOL Server Reference*.

Related Topics

- Display Online Help, on page 30
- Edit IDOL Configuration Files, on page 32
- Encrypt Passwords, on page 507

To set up automatic security application for documents

1. Open the IDOL Content component configuration file in a text editor.

2. In the `[Security]` section, set the `SecurityInfoKeys` parameter to specify the security encryption keys to use to encrypt and decrypt the security information used by the IDOL Content component. You can set the `SecurityInfoKeys` parameter either to the name of an AES key file (recommended) or to a comma-separated list of four signed 32-bit integers. For information about how to generate a key file with the `autpassword` command-line tool, see Encrypt Passwords, on page 507.

   For example:

   ```
   [Security]
   SecurityInfoKeys=MyAESKeyFile.ky
   ```

3. In the `[Security]` section, list the security types that you want to use.

   ```
   0=NT
   1=Netware
   ```

```
2=Notes
3=Exchange
```

4. Create a section for each of the security types you defined (the section must have the same
    name as the security type). For each section, provide settings that determine how Content
    handles that security type. For example:

```
[NT]
SecurityCode=1
Library=nt_security.dll
Type=AUTONOMY_SECURITY_V4_NT_MAPPED
ReferenceField=*/AUTONOMYMETADATA

[Netware]
SecurityCode=2
Library=netware_security.dll
Type=AUTONOMY_SECURITY_NETWARE_MAPPED
ReferenceField=*/AUTONOMYMETADATA

[Notes]
SecurityCode=3
Library=notes_security.dll
Type=AUTONOMY_SECURITY_V4_NOTES_MAPPED
ReferenceField=*/AUTONOMYMETADATA

[Exchange]
SecurityCode=4
Library=exchange_security.dll
Type=AUTONOMY_SECURITY_EXCHANGE_MAPPED
ReferenceField=*/AUTONOMYMETADATA
```

5. In the [FieldProcessing] section, set up processes that allow Content to recognize the security
    type of documents (unless you send an additional parameter to specify the security property of a
    document every time you index a document).

    If you use a version 4 security type (for example, AUTONOMY_SECURITY_V4_NOTES_MAPPED), you
    must include a process that defines how to handle metadata. For example:

```
[FieldProcessing]
0=DetectNT
1=DetectNetware
2=DetectNotes
3=DetectExchange
4=DefineMetaData
```

6. Create a section for each of the processes that you listed, in which you create a property for the
    process (security properties always point to a defined security type). Identify the field that you
    want to associate with the processes.

    > **NOTE:** The property that you create must not have the same name as the process.

To identify the fields, use the format */FieldName* to match root-level fields, */FieldName* to match all fields except root-level, or */Path/FieldName* to match fields that the specified path points to.

You can use the `PropertyMatch` parameter to identify a specific value that fields must have to be processed.

For example:

```
[DetectNT]
Property=SetNTProperty
PropertyFieldCSVs=*/DRESECURITYTYPE
PropertyMatch=*nt

[DetectNetware]
Property=SetNetwareProperty
PropertyFieldCSVs=*/DRESECURITYTYPE
PropertyMatch=*netware

[DetectNotes]
Property=SetNotesProperty
PropertyFieldCSVs=*/DRESECURITYTYPE
PropertyMatch=*notes

[DetectExchange]
Property=SetExchangeProperty
PropertyFieldCSVs=*/DRESECURITYTYPE
PropertyMatch=*exchange

[DefineMetaData]
Property=HideMetaData
PropertyFieldCSVs=*/AUTONOMYMETADATA
```

7. Create a section for each of the properties and specify appropriate configuration settings for each property. These configuration parameters define the processes to apply to all the fields (or all documents that contain the fields) that you previously associated with the processes.

   If you use a version 4 security type (for example, `AUTONOMY_SECURITY_V4_NOTES_MAPPED`), you must set `ACLType` to **True** in the section that sets up how Content handles metadata, to implement optimized security.

```
[SetNTProperty]
SecurityType=NT

[SetNetwareProperty]
SecurityType=Netware

[SetNotesProperty]
SecurityType=Notes

[SetExchangeProperty]
SecurityType=Exchange
```

```
[HideMetaData]
HiddenType=True
ACLType=True
```

8. Save and close the configuration file.

9. Restart the IDOL Content component for your changes to take effect.

> **NOTE:** For details of how to ensure security in an IDOL infrastructure, refer to the *IDOL Document Security Administration Guide*.

> **TIP:** You can view information about the configured security types in IDOL Server on the **Security Types** tab on the Status page in the IDOL Admin interface. For more information, refer to the *IDOL Admin User Guide*.

Configure Client Authorization

You can configure IDOL Server to authorize different operations for different connections.

Authorization roles define a set of operations for a set of users. You define the operations by using the `StandardRoles` configuration parameter, or by explicitly defining a list of allowed actions in the `Actions`, `IndexActions` and `ServiceActions` parameters. You define the authorized users by using a client IP address, SSL identities, and GSS principals, depending on your security and system configuration.

For more information about the available parameters, see the *IDOL Server Reference*.

> **IMPORTANT:** To ensure that IDOL Server allows only the options that you configure in `[AuthorizationRoles]`, make sure that you delete any deprecated `RoleClients` parameters from your configuration (where `Role` corresponds to a standard role name, for example `AdminClients`).

To configure authorization roles

1. Open your configuration file in a text editor.

2. Find the `[AuthorizationRoles]` section, or create one if it does not exist.

3. In the `[AuthorizationRoles]` section, list the user authorization roles that you want to create. For example:

```
[AuthorizationRoles]
0=AdminRole
1=UserRole
```

4. Create a section for each authorization role that you listed. The section name must match the name that you set in the `[AuthorizationRoles]` list. For example:

```
[AdminRole]
```

5. In the section for each role, define the operations that you want the role to be able to perform. You can set `StandardRoles` to a list of appropriate values, or specify an explicit list of allowed actions by using `Actions`, `IndexActions`, and `ServiceActions`. For example:

```
[AdminRole]
StandardRoles=Admin,Index,ServiceControl,ServiceStatus

[UserRole]
Actions=GetVersion
ServiceActions=GetStatus
```

> **NOTE:** The standard roles do not overlap. If you want a particular role to be able to perform all actions, you must include all the standard roles, or ensure that the clients, SSL identities, and so on, are assigned to all relevant roles.

6. In the section for each role, define the access permissions for the role, by setting `Clients`, `SSLIdentities`, and `GSSPrincipals`, as appropriate. If an incoming connection matches one of the allowed clients, principals, or SSL identities, the user has permission to perform the operations allowed by the role. For example:

```
[AdminRole]
StandardRoles=Admin,Index,ServiceControl,ServiceStatus
Clients=localhost
SSLIdentities=admin.example.com
```

7. Save and close the configuration file.

8. Restart IDOL Server for your changes to take effect.

> **IMPORTANT:** If you do not provide any authorization roles for a standard role, IDOL Server uses the default client authorization for the role (localhost for `Admin` and `ServiceControl`, all clients for `Query`, `Index`, and `ServiceStatus`). If you define authorization only by actions, Micro Focus recommends that you configure an authorization role that disallows all users for all roles by default. For example:

```
[ForbidAllRoles]
StandardRoles=*
Clients=""
```

> This configuration ensures that IDOL Server uses only your action-based authorizations.

Set up an SSL Connection

There are several ways to set up Secure Socket Layer (SSL) connections for IDOL Server. For example, you can:

- Configure an SSL *gateway*. You configure incoming communications to a unified IDOL Server to use SSL connections, but communications between components are plain.

- Configure SSL between all IDOL components in a unified IDOL Server. All communications into IDOL, and between components, are configured with SSL connections.

- Configure SSL between stand-alone IDOL components.

In all cases the basic principle of configuring SSL is the same, but the exact configuration varies.

To configure SSL connections

1. Set the `SSLConfig` parameter to the name of the section in which you define SSL options. The configuration sections where you set `SSLConfig` vary depending on your setup. In general:

    - For incoming ACI calls, set the `SSLConfig` parameter in the `[Server]` section.

    - For incoming Index actions, set the `SSLConfig` parameter in the `[IndexServer]` section.

    - For incoming Service actions, set the `SSLConfig` parameter in the `[Service]` section.

    - For outgoing ACI calls to IDOL components, set the `SSLConfig` parameter in each component section. For example, `[AgentDRE]`.

    For example:

    ```
    [Server]
    SSLConfig=SSLOption1
    ```

2. For each `SSLOption` you define, create a new configuration section to contain the SSL options.

    For example:

    ```
    [SSLOption1]
    ```

3. Within each SSL options section, you can specify the following SSL parameters:

`SSLMethod`	Determines which SSL protocol to use: TLSV1.3, TLSV1.2, TLSV1, SSLV3, or Negotiate, which uses the highest protocol supported by both client and server. Micro Focus recommends that you use `TLSV1.3`, unless interoperability with older systems requires use of less secure protocols.
`SSLCertificate`	The SSL Certificate file to use to identify this component to a peer. The certificate can be in either ASN1 or PEM format. Micro Focus recommends that you use the PEM format. This parameter requires a matching `SSLPrivateKey` value.
`SSLPrivateKey`	The private security key for the SSL certificate. The security key can be in either ASN1 or PEM format. This parameter requires a matching `SSLCertificate` value. The private key can be password protected. See `SSLPrivateKeyPassword`.
`SSLCACertificate`	The Certificate Authority certificate indicating that this component trusts only communication with a peer that offers a certificate signed by the specified CAs.

SSLCheckCertificate	Requests a certificate signed by a trusted authority from peers.
	Setting `SSLCACertificate` implicitly sets this parameter to **True**. If you set `SSLCACertificate` to **False**, IDOL Server encrypts communications, but does not request certificates from peers.
SSLCheckCommonName	Determines whether the host name listed in the peer certificate (that is, the CommonName or "CN" attribute) resolves to the same IP address as the peer itself, as determined by the network connection. This parameter helps verify the identity of the peer.
	For example, if the host name in a certificate is `eip.example.com` and resolves to an IP address of `12.3.4.56`, the peer must share the same IP address.
SSLPrivateKeyPassword	If the file defined in `SSLPrivateKey` is password protected, use this parameter to specify the password. The password can be in plain text or in basic or AES encryption format.

Related Topics

- Password Encryption, on page 507

Set up SSL between IDOL components

If you are using a unified IDOL Server configuration, you can enable SSL communication between IDOL components. Set the `SSLIDOLComponents` parameter to **True** in the `[Server]` section.

You can configure Secure Socket Layer (SSL) connections for communication between the following components and other IDOL components:

- Agentstore
- Category
- Community
- Content
- IDOL Proxy
- View

You can set `SSLConfig` in the following configuration sections for SSL communications between IDOL components:

- `[Server]` to configure SSL communications for incoming ACI calls for all components.
- `[IndexServer]` to configure incoming SSL communications to the IDOL Server index port. This option implicitly includes any indexing components (such as Content).
- `[Service]` to configure incoming SSL communications to the IDOL Server service port.

- [Agent] to configure outgoing SSL communications from the IDOL Category component to the IDOL Content component where the IDOL Server agent index is stored (Agentstore).

- [AgentDRE] to configure outgoing SSL communications from IDOL components to the IDOL Content component where the IDOL Server agent index is stored (Agentstore).

- [CatDRE] to configure outgoing SSL communications from IDOL components to the IDOL Content component where the IDOL Server category index is stored (Agentstore).

- [DataDRE] to configure outgoing SSL communications from IDOL components to the IDOL Content component where the IDOL Server data index is stored (Content).

NOTE: For SSL communication with the IDOL Agentstore component, you must also configure SSL settings in the IDOL Agentstore component configuration file.

For example:

```
[Server]
SSLConfig=SSLOptions1
...

[AgentDRE]
SSLConfig=SSLOptions2
...

[DataDRE]
SSLConfig=SSLOptions2
...
```

For Omni Group Servers:

```
[Note]
GroupServerHost=...
GroupServerPort=...
SSLConfig=SSLOptions2

[SSLOptions1]              //SSL options for incoming connections
SSLMethod=TLSV1.3
SSLCertificate=host1.crt
SSLPrivateKey=host1.key
SSLCACertificate=trusted.crt

[SSLOptions2]             //SSL options for outgoing connections
SSLMethod=TLSV1.3
SSLCertificate=host2.crt
SSLPrivateKey=9s7BxMjD2d3M3t7awt/J8A
SSLCACertificate=trusted.crt
```

Set up SSL for Shared Communications

To define one set of SSL options to share between multiple communications, define only one `SSLOption` section. Set all the `SSLConfig` parameters to the single `SSLOption` section in each configuration section. For example:

```
[Server]
SSLConfig=SSLOptions
...

[AgentDRE]
SSLConfig=SSLOptions
...

[DataDRE]
SSLConfig=SSLOptions
...
```

For Omni Group Servers:

```
...
[Note]
GroupServerHost=hostname
GroupServerPort=1.2.3.4
SSLConfig=SSLOptions

[SSLOptions]
...
```

Set up SSL for Mailer

You can set up Secure Socket Layer (SSL) connections between the Mailer component and Community and Category.

To configure SSL for Mailer, set the following parameters in the `[Email]` section:

SSLConfig	The SSL options for a secure connection with Community.
ClassificationSSLConfig	The SSL options for a secure connection with Category.

For example:

```
[Email]
SSLConfig=SSLOption
ClassificationSSLConfig=CatSSLConfig
```

Set up SSL for the IDOL View component

To configure SSL for the IDOL View component, you can set the following parameters in the
`[Viewing]` section:

`SSLConfig`	SSL options for a secure connection to the IDOL Content component to use for highlighting.
`OutgoingSSLConfig`	SSL options for a secure connection to other applications. View uses this option if someone requests to view a document that exists on a server that is secured by SSL.

For example,

```
[Viewing]
IdolHost=localhost
IdolPort=9010
SSLConfig=SSLOption1
OutgoingSSLConfig=SSLOption2
```

Set up SSL for Communications to Remote Servers

You can use SSL connections to the remote server when you use the `DREEXPORTREMOTE` index action.

You must configure an `SSLOptions` configuration section in the IDOL Content component configuration file. For example:

```
[SSLOptions1]
SSLMethod=TLSV1.2
SSLCertificate=host1.crt
SSLPrivateKey=host1.key
SSLCACertificate=trusted.crt
```

When you send the `DREEXPORTREMOTE` index action, set the `SSLConfig` action parameter to the name of the configuration section where you configure the SSL options. For example:

```
http://localhost:20001/DREEXPORTREMOTE?&TargetEngineHost=banff&TargetEnginePort=202
00&Delete=True&Blocking=True&BatchSize=10000&SSLConfig=SSLOptions1
```

In this example, the IDOL Content component uses the SSL configuration options from the
`[SSLOptions1]` configuration section to contact the remote server.

Log SSL Settings

You can define a log file for logging SSL setting details:

```
[LOGGING]
0=...
1=SSL_LOG_STREAM
```

```
[SSL_LOG_STREAM]
LogFile=ssl.log
LogTypeCSVS=SSL
LogLevel=Normal
LogTime=True
LogEcho=False
LogMaxSizeKBs=1024
```

Check SSL Status

You can check whether your service port and index port SSL settings are correct by using ACI actions.

- The `GetChildren` action returns the `<autn:serviceport_ssl_enabled>` tag. The value is **True** when SSL is enabled for the service port.

- The `GetStatus` action returns the `<indexport_ssl_enabled>` tag. The value is **True** when SSL is enabled for the index port.

Chapter 20: Add Users to IDOL Server

This section describes how to create and manage IDOL user accounts.

Create IDOL Users

You must store users in the IDOL Community component to use any of the following IDOL Server features:

- **Agents**. Users can store queries in the form of agents to ensure that they always have the latest available information. Users can edit and retrain their agents.

- **Profiling**. A profile is a set of agents that are trained using the documents that the user is looking at, and that return data that matches their interests. You can set up your application so that every time a user looks at a document, the profile determines whether the document is relevant to its agent training. The profile then either updates the training with the document content, or creates a new profile agent for the user.

- **Collaboration**. You can match users that have common agents or similar profiles.

- **Alerting**. When IDOL Server receives new content that matches user agents, the server immediately notifies the user by email or a third-party system (for example, by SMS or a pager).

- **Mailing**. The IDOL Community component matches the agents and profiles against its document content at regular intervals, and automatically notifies users of documents that match their agents and profiles by sending them an email message.

- **Expertise**. The IDOL Community component accepts a natural language or Boolean search string and returns users who own matching agents or profiles. This process allows instant identification of experts in any subjects at hand.

You can create a user structure that is either flat (all users have equal roles and responsibilities) or hierarchical (you assign roles to users, which can hierarchically relate to other roles).

Flat Structure

To create a flat user structure, use the `UserAdd` action to create users. For example:

`http://`*`CommunityHost`*`:`*`port`*`/action=UserAdd&UserName=JaneBrown&Password=Sesame`

where:

CommunityHost	is the IP address or host name of the machine where the IDOL Community component is installed.
port	is the IDOL Community component action port (specified as the `Port` value in the `[Server]` section of the IDOL Community component configuration file).

Hierarchical Structure

To use a hierarchical user structure, you must create a hierarchical set of roles. You can then assign users to the roles.

To create a hierarchical user structure

1. Decide which groups you want to use to structure your users. You can, for example, group them according to roles and responsibilities.

2. Use the `RoleAdd` action to create a role for each user group. For example:

   `action=RoleAdd&RoleName=Sales`

3. Use the `RoleAddRoleToRole` action to create a hierarchical structure of roles. For example:

   `action=RoleAddRoleToRole&RoleName=Sales&ParentRoleName=TeleSales`

4. Use the `UserAdd` action to create individual IDOL users. For example:

   `action=UserAdd&UserName=JaneBrown&Password=Sesame`

5. Use the `RoleAddUserToRole` action to associate each user with a role. For example:

   `action=RoleAddUserToRole&UserName=JaneBrown&RoleName=TeleSales`

Create Users in IDOL Admin

As an alternative to using actions, you can use the IDOL Admin interface to create users.

To create users in IDOL Admin

1. In the **Control** menu, click **Users**.

2. Click **+Add**.

   The Add User dialog box opens.

3. Type a user name and password. Type the password again in the **Confirm Password** field.

4. Click **Add User**.

   The Add User dialog box closes and the user appears in the user list.

Manage Roles in IDOL Admin

As an alternative to submitting actions, you can create roles and assign them to users by using the IDOL Admin interface.

To manage roles in IDOL Admin

1. In the Control menu, click **Roles**.

2. Click **Add Role**.

   The Add Role dialog box opens.

3. Type the name of the role that you want to create, then click **Add Role**.

   IDOL Admin creates the role and adds it to the list of roles.

4. Select the role that you want to associate with the user, then click **Add User**.

   The Add User(s) To Role dialog box opens, displaying a list of IDOL users.

5. Select the check boxes beside the users to add to the role. To select all users, select the check box alongside **User**. The check boxes next to all users are automatically selected.

6. Click **Ok**.

   The Add User(s) to Role dialog box closes and the users are added to the user list.

To remove a user from the selected role, click **X** beside the user. The Delete Role User Confirmation dialog box opens; click **Delete** to remove the user from the role.

Integrate with a Third-Party User Structure

The `DeferLogin` option allows you to integrate the IDOL Community component with a third-party system (such as SiteMinder, Windows NT, LDAP, or Lotus Notes) to manage authentication. The entitlements of the users are set to the ones given to the IDOL Community component default (root) role.

To use the `DeferLogin` option

1. Set `DeferLogin` to **True** in the `[Server]` section of the IDOL Community component configuration file, and restart the IDOL Community component.

2. Add `DeferLogin=True` to any user action that you send.

When you turn on `DeferLogin`, Community accesses the third-party system to manage the user authentication. When a user logs onto the system for the first time, Community creates a user with the appropriate name, and allocates the default role permissions and settings to this user.

For example, if you send a user action (such as `UserRead` or `ProfileUser`) with `DeferLogin` for a user that does not currently exist in Community, Community automatically creates the user in the system, based on the user name from the third-party system.

Implement User Account Security

The IDOL Community component provides several security options for user accounts.

- In addition to passwords, you can assign PIN codes to users for authentication with Community

- You can set a minimum length for user names and passwords, and a minimum strength for user passwords.

- You can also specify how often users must change passwords and PIN codes, and ensure that they do not use the same passwords again.

- You can set up Community to protect against *brute force attacks*. Community can lock user accounts if there are too many incorrect logon attempts from a user in a certain time period.

Create User PIN Codes

You can create user PIN codes to use for authentication in addition to passwords. The IDOL Community component can lock users out if they fail to authenticate using the PIN code.

To configure the IDOL Community component to use PIN codes

1. Open the IDOL Community component configuration file in a text editor.

2. Find the `[User]` section, or create one if it does not exist.

3. Set the `PincodeLength` parameter to the length of PIN code that you want to use. Created PIN codes must have the specified length, and must consist of alphanumeric characters. For example:

   ```
   [User]
   PincodeLength=8
   ```

4. Save and close the configuration file.

5. Restart the IDOL Community componentfor your changes to take effect.

Add a PIN Code for a User

When you enable PIN codes in the configuration file, you can assign them to users by sending a `UserPin` action.

To add or change a PIN code

- Send the following action to the IDOL Community component:

  ```
  http://
  CommunityHost:CommunityPort/action=UserPin&UserName=Username&Pincode=PinValue
  ```

  where:

`CommunityHost`	is the name or IP address of the host on which the IDOL Community component runs.
`CommunityPort`	is the IDOL Community component ACI port.
`Username`	is the name of the user for which you want to add a PIN code.
`PinValue`	is the value of the new pin.

Authenticate Users with PIN Codes

You authenticate users with PIN codes by using the `UserPin` action. This action can check certain characters in the PIN code, rather than the whole PIN code.

To authenticate a PIN code

- Send the following action to the IDOL Community component:

```
http://
CommunityHost
:
CommunityPort
/action=UserPin&UserName=Username&Positions=Positions&Values=Values
```

where:

`CommunityHost`	is the name or IP address of the host on which the IDOL Community component runs.
`CommunityPort`	is the IDOL Community component ACI port.
`Username`	is the user name of the user whose account you want to authenticate.
`Positions`	is a comma-separated list of the positions in the PIN code that you want to check.
`Values`	is a comma-separated list of the values that Community must check against the specified positions.

For example, to authenticate by using the third, fifth, and sixth characters from a PIN code, you must set `Positions` to **3,5,6**. The `Values` parameter then contains the values that the user provides, for example **y,9,a**. Community checks that the specified values occur at the specified positions in the PIN code.

Set User Name and Password Restrictions

You can specify certain restrictions on user account names and passwords to ensure that it is difficult to gain unauthorized access.

To configure user account name and password restrictions

1. Open the IDOL Community component configuration file in a text editor.

2. Find the `[User]` section, or create one if it does not exist.

3. Set the `MinUsernameLength` parameter to the minimum number of characters allowed for user names. For example:

   `MinUsernameLength=6`

4. Set the `MinPasswordLength` parameter to the minimum number of characters allowed for passwords. For example:

   `MinPasswordLength=6`

5. Set the `PasswordStrength` parameter to the minimum allowed strength of passwords. This parameter uses a scale from 1 (weakest) to 10 (strongest). For example:

   `PasswordStrength=6`

   To increase password strength, users must create passwords that contain a mixture of upper- and lowercase letters, numbers, and punctuation.

   > **TIP:** You can also create custom password strength requirements (or use an external Web service) by creating a Lua script. In this case, you must set the `UserLuaScript` configuration parameter to the path to the script to use. The script must define the `password_complexity_check` function, which Community calls to check the password rules. In this case, Community uses the Lua script, and then also checks the password against your `PasswordStrength` criteria (if set).
   >
   > For more information, refer to the *IDOL Server Reference*.

6. Save and close the configuration file.

7. Restart the IDOL Community component for your changes to take effect.

Enable Password and PIN Code Time Restrictions

To help prevent unauthorized access, you can ensure that users change their passwords and PIN codes regularly.

You can ensure that users do not re-use passwords. The IDOL Community component can store a list of used passwords, and prevent users from re-using a stored password.

Community can also ensure that users keep passwords for a minimum duration. This option prevents users from immediately changing their password several times to return to a previous password.

> **NOTE:** You can use the `UserReadUserListDetails` action to check the time remaining (in seconds) until password expiration for a user. If the password has expired, `-1` is displayed.

To configure password and PIN code time restrictions

1. Open the IDOL Community component configuration file in a text editor.

2. Find the `[User]` section, or create one if it does not exist.

3. Set the `PasswordChangeDuration` parameter to the time interval after which users must change their password. For example:

   ```
   PasswordChangeDuration=60days
   ```

4. Set the `PincodeChangeDuration` parameter to the time interval after which users must change their PIN code. For example:

   ```
   PincodeChangeDuration=60days
   ```

5. Set the `MaxNumPasswordPerUser` parameter to the number of passwords that you want to store in Community. Users cannot change their passwords to any of the stored passwords. For example:

   ```
   MaxNumPasswordPerUser=5
   ```

6. Set the `KeepPasswordDuration` parameter to the length of time that users must keep a password before they are allowed to change it again. For example:

   ```
   KeepPasswordDuration=5days
   ```

7. Save and close the configuration file.

8. Restart the IDOL Community component for your changes to take effect.

9. Notify your users of your password and PIN policies.

Set Maximum Login Attempts

To protect against brute force attacks on user accounts, you can configure the IDOL Community component to lock user accounts when there are too many incorrect login attempts within a specified time period.

To set a maximum number of login attempts

1. Open the IDOL Community component configuration file in a text editor.

2. Find the `[User]` section, or create one if it does not exist.

3. Set the `LoginMaxAttempts` parameter to the maximum number of incorrect login attempts to allow in the time period.

4. Set the `LoginExpiryTime` parameter to the time (in seconds) before the current number of login attempts resets. Community locks the user account if there are too many incorrect login attempts within this time period. For example:

```
LoginMaxAttempts=3
LoginExpiryTime=60
```

In this example, the user account locks if there are three incorrect login attempts within 60 seconds of each other.

5. To automatically unlock users, set the `LockRemovalDuration` parameter to the length of time that the user remains locked. For example:

```
LockRemovalDuration=24hours
```

Set `LockRemovalDuration` to **-1** to disable it.

6. Save and close the configuration file.

7. Restart the IDOL Community component for your changes to take effect.

8. Notify your users of your password and PIN policies.

Users must contact a system administrator to unlock their accounts, unless you configure the `LockRemovalDuration` parameter.

Lock and Unlock User Accounts

If genuine users are locked out, you can reinstate them by using a `UserLock` action. You can also use this action to lock user accounts manually.

> **TIP:** You can also click **Lock** or **Unlock** in the User Details page in the IDOL Admin interface to lock and unlock users.

To unlock a user account

- Send the following action to the IDOL Community component:

```
http://
CommunityHost:CommunityPort/action=UserLock&UserName=Username&Unlock=True
```

where:

CommunityHost	is the name or IP address of the host on which the IDOL Community component runs.
CommunityPort	is the IDOL Community component ACI port.
Username	is the user name of the user whose account you want to unlock.

To lock a user account

- Send the following action to the IDOL Community component:

```
http://
```

CommunityHost:*CommunityPort*/action=UserLock&UserName=*Username*&Unlock=False

where:

CommunityHost	is the name or IP address of the host on which the IDOL Community component runs.
CommunityPort	is the IDOL Community component ACI port.
Username	is the user name of the user whose account you want to lock.

Chapter 21: Mail

The IDOL Community component can automatically match user agents and subscription channels against its document content at regular intervals. It automatically sends users email to notify them of documents that match their agents and the channels that they are subscribed to. You determine the format of the emails by using customizable templates.

Automatically Email Agent and Channel Results

You can configure the IDOL Community component to automatically email users the results that their agents and channels produce. You can schedule the emailing of agent and channel results and optionally store lists of sent results to prevent the duplication of email.

To set up automatic email of agent and channel results

1. Open the IDOL Community component configuration file in a text editor.

2. Find the `[UserCustom]` section. This section lists all the custom processes that Community runs.

3. Check whether the `[UserCustom]` section lists a section for emailing. If it does not, add one. For example:

   ```
   [UserCustom]
   0=Email
   ```

4. Create a configuration file section for the emailing process that you listed. For example:

   ```
   [Email]
   ```

5. In your new section, set `Library` to ***InstallDir*/community/modules/user_email**. Set `RunMailer` to **True** and `DefaultSendEmail` to **True** to enable the Community mailing operation.

6. Specify a `TestUser`. While you are configuring mailing, Community sends all mail to the `TestUser` email address.

7. If you are using a proxy server, specify the `ProxyHost`, `ProxyPort`, your `ProxyUsername`, and your `ProxyPassword`.

8. Use `SMTPHost` and `SMTPPort` to specify the details of your mail server.

9. Use `Cycles` and `Interval` to determine how many times the mailing operation must run, and the time span that you want to elapse between the sending of email. Set `StartTime` to `now`, so that you can test the mailing operation immediately when you start Community.

10. Set `Retries` to the number of times that Community attempts to connect to its agent index before it times out. Set `TimeoutMS` to how long each of these attempts can take.

11. Use `From`, `FromHost`, and `FromName` to set the details to display as the sender of email that the mailing operation sends. You can also use `FromField` and `FromNameField` to specify a user field that contains the sender details.

12. Specify the `DefaultSubject` to display as the mail subject line.

13. Use `XSLTemplate` to specify the template to use for the email. The `DefaultEmailFormat` and `DefaultEmailResultsType` settings allow you to specify the email format and whether to send results individually or in sets.

14. Set `DefaultAddSetToReadDocuments` to **True** to automatically add the documents in the email to the list of documents that the user has viewed. Set `DefaultExcludeReadDocuments` to **True** to exclude documents that the user has recently viewed from the email (so that they receive each result only once).

    You must set `DreTemplateReferenceStart` and `DreTemplateReferenceEnd` to ensure that Community can extract document references and determine if they were viewed.

15. To include channel results in the email that the mailing operation sends, configure the following parameters:

    - `ClassificationServerHost`. The host name or IP address of the IDOL Category component to use.

    - `ClassificationServerPort`. The ACI port of the IDOL Category component.

    - `ClassificationServerXSLTemplate`. The template to use to display channel results.

    - `ClassificationServerNumResults`. The maximum number of channel results to include in the email.

    - `ClassificationServerThreshold`. The quality of channel results to include in the email.

    - `ClassificationServerParams`. Parameters that must be included in the channels query that the mailing operation sends to the category index.

    - `ClassificationServerValues`. The values of the specified `ClassificationServerParams` parameters.

    - `ClassificationServerRetries`. The number of times that the mailing operation attempts to connect to the category index.

    - `ClassificationServerTimeout`. Specifies how long each of the `ClassificationServerRetries` can take, before the mailing operation times out.

    > **NOTE:** Users receive channel results only for categories that they subscribe to. You can subscribe a user to one or more categories by sending a `UserEdit` action to Community. Use the `CategorySubscribe` action parameter to specify the categories whose results you want to mail to the user. (You can unsubscribe a user by using a `UserEdit` action with

> the `CategoryUnsubscribe` action parameter set to the categories whose results must no longer be included in the email to the user.)

16. To minimize the impact that the mailing operation has on your system resources, you can set `SleepBetweenRequests` and `MaxEmailsPerUser` to values that are appropriate for your environment.

17. Save the IDOL Community component configuration file and restart Community. The mailing operation starts immediately because you set `StartTime` to **now**. It sends mail to the `TestUser` address that you specified. Ensure that the mail process works smoothly.

18. Make any adjustments to your settings that you need, then save the configuration file again and restart the IDOL Community component. You can enable `VerboseLogging` if you experience problems with the mailing operation.

When you are satisfied with the mailing operation:

1. Open the IDOL Community component configuration file in a text editor.

2. Find the `[UserCustom]` section.

3. Delete the email address that you specified for `TestUser`, and set `StartTime` to the time when you want the mailing operation to start.

4. Save and close the configuration file.

5. Restart the IDOL Community component for your changes to take effect.

Related Topics

Send Custom Emails

You can configure the IDOL Community component to send an email that contains details of a specific single document to a user.

To configure the IDOL Community component to send custom emails

1. Open the IDOL Community component configuration file in a text editor.

2. Find the `[UserCustom]` section.

   If you already added a custom section to automatically email results to users, the same settings enable the sending of custom emails. If you are using this existing section, ensure that you specify the template to use for custom emails by using the `EmailActionXSLTemplate` parameter. Continue with Step 7.

   If you want Community to send custom emails without enabling automatic emailing of the agent and channel results, specify a new custom section. Continue with Step 2.

   > **NOTE:** For details of configuration parameters, refer to the *IDOL Server Reference*.

3. Add a section to the configuration file with the name that you specified in the [UserCustom] section.

4. In your new section, set `Library` to *InstallDir*/**community/modules/user_email** to enable the mailing operation.

5. If you are using a proxy server, specify the `ProxyHost`, `ProxyPort`, your `ProxyUsername`, and your `ProxyPassword`.

6. Use `SMTPHost` and `SMTPPort` to specify the details of your mail server.

7. Specify the template to use to create custom emails as the value of the `EmailActionXSLTemplate` parameter.

8. Save and close the configuration file.

9. Restart the IDOL Community component for your changes to take effect.

10. Send a `Custom` action to IDOL Server:

   - Set `Function` to **email**.

   - Set `Library` to the name of the custom section in the IDOL Server configuration file that sets up the mailing operation.

   Refer to the *IDOL Server Reference* for details on the `Custom` action.

The mailing operation uses the template that you specified as the value of the `EmailActionXSLTemplate` parameter to create the email that it sends to the specified user.

Related Topics

- Display Online Help, on page 30
- Automatically Email Agent and Channel Results, on page 409

Send Emails in Batches

By default, the IDOL Community component sends emails individually. To speed up the email process, you can configure Community to send emails in batches. The mailer can continue generating emails while sending batches of emails.

To set up batch emails

1. Open the IDOL Community component configuration file in a text editor.

2. Find the [Email] section.

3. Add the following configuration parameter:

MaxEmailsToSendBatchProcess	Specify the maximum number of emails to store in the queue before sending as a batch. The default value is 20.

4. You can slow the email process by using the following configuration parameter:

`SleepBetweenSentEmails`	Specify a sleep time (in milliseconds) after the mailer sends each email. The default value is **0**.

For example:

```
[Email]
MaxEmailsToSendBatchProcess=100
SleepBetweenSentEmails=50
```

Mailer Templates

The following table describes the XSL templates that the IDOL Community component installation includes for the mailing operation.

For automatically emailing agent and channel results:	
`email.xss`	The main template that the `user_email` library uses for results emails. `email.xss` specifies the overall structure of emails, and includes specific instructions for displaying individual agent results.
`channels.xss`	The template that the `user_email` library uses for formatting the channel results that the `email.xss` template includes.
For sending custom emails:	
`ondemand.xss`	The template that specifies how to display the e-mails that Community sends in response to a `Custom` action.

You can modify these templates to customize email layout.

Edit Templates

The XSL templates use XPath and XSLT to identify and display fields from the XML returned in response to actions sent to the IDOL Community component.

You identify the XML fields that the template uses to create emails by using the `select` attribute in the template `XSL` tags. To identify the XML fields that a template can use, use a Web browser to send the HTTP action for which Community uses the template to display results. You can then determine available field names from the `autn` tags in the XML that returns. The action to send depends on the template that you are editing.

Template	Action
email.xss	AgentGetResults
channels.xss	CategoryQuery
ondemand.xss	Custom

For details of how to send these actions, refer to the *IDOL Server Reference*.

Related Topics

For example, if you send an AgentGetResults action to the IDOL Community component, the following XML might return:

```
<?xml version='1.0' encoding='UTF-8' ?>
<autnresponse xmlns:autn='http://schemas.autonomy.com/aci/'>
   <action>AGENTGETRESULTS</action>
   <response>SUCCESS</response>
   <responsedata>
      <autn:agent>
         <autn:aid>2-A2</autn:aid>
         <autn:training />
         <autn:parent>2</autn:parent>
         <autn:agentname>agent21</autn:agentname>
         <autn:fields>
      <retrained>true</retrained>
         <private>false</private>
         <fromdocument>true</fromdocument>
      </autn:fields>
      <autn:results>
         <autn:numhits>1</autn:numhits>
         <autn:hit>

            <autn:reference>http://193.115.251.40/ArchiveData/encarta/38000/msdata3
9439.htm</autn:reference>
            <autn:id>1254</autn:id>
            <autn:section>0</autn:section>
            <autn:weight>70.77</autn:weight>

            <autn:links>TAPESTRI,REVIV,WEAV,REACH,EUROPEAN,OCCUR,PRACTIC,TRADIT,REM
AIN,EUROP,EXAMPL,WESTERN,ALTHOUGH,EAR</autn:links>
            <autn:database>News</autn:database>
            <autn:title>Tapestry Tapestry weaving may have been practiced in Europe
as...</autn:title>
            <autn:summary>Tapestry Tapestry weaving may have been practiced in
Europe as ... . Tapestry Tapestry weaving may have been practiced in Europe as
early as the 8th century, although no examples remain. Western European tapestry
```

```
reached its greatest development between the 14th and 18th centuries. During the
19th and 20th centuries, however, revivals of the tapestry tradition occurred. .
</autn:summary>
                <autn:content>
                    <DOCUMENT>

                    <DREREFERENCE>http://193.115.251.40/ArchiveData/encarta/38000/msdata
39439.htm</DREREFERENCE>
                    <DRETITLE>Tapestry   Tapestry weaving may have been practiced in
Europe as ... </DRETITLE>
                    <BLANK />
                    <IMAGE>archiv</IMAGE>
                    <PAPER />
                    <SUMMARY>Tapestry    Tapestry weaving may have been practiced in
Europe as early as the 8th century, although no examples remain. Western European
tapestry reached its greatest development between the 14th and 18th
centuries</SUMMARY>
                    <DOCTYPE>ARCHIVE</DOCTYPE>
                    <DREDATE>907347778</DREDATE>
                    <DREDBNAME>ARCHIVE</DREDBNAME>
                    <DRECONTENT>Tapestry Tapestry weaving may have been practiced in
Europe as early as the 8th century, although no examples remain. Western European
tapestry reached its greatest development between the 14th and 18th centuries.
During the 19th and 20th centuries, however, revivals of the tapestry tradition
occurred. </DRECONTENT>
                <autn:content>
            </autn:hit>
        </autn:results>
    </autn:agent>
</responsedata>
</autnresponse>
```

In this example, you can see from the XML that Community returns that the following fields are
available as values for the `select` attribute:

agent	private	section
aid	fromdocument	weight
training	results	links
parent	numhits	database
agentname	hit	title
fields	reference	summary
retrained	id	content

You can include these fields as values in the XSL tags. For example, to display the value of the `<autn:
title>` tag for each result document, include these lines in your template:

```
<xsl:for-each select=responsedata/hit">
<xsl:value-of select="title">
</xsl:for-each>
```

You must remove the `autn:` part of the tag from the XSL tag that you specify. For example, if the XML that Community returns contains a tag called `autn:title`, specify the tag as `title` (as in `select="title",` in the example here).

Chapter 22: Administer IDOL Server

You can administer IDOL Server by performing the tasks described in this section.

Enable Configuration Changes

When you change an IDOL component configuration file, you must stop and restart the component so that it reads the new configuration.

To stop and restart an IDOL component

- **For Windows**. Display the Windows Services dialog box, stop the component service, then start it again.

- **For UNIX**. Use the `Stop.sh` stop script to stop the component, then use the `Start.sh` script to start it again. These scripts are supplied with the IDOL Server installation.

- **For any platform**. In the Console page in IDOL Admin, click the **Service Control** tab, then click **Stop Service**. Then manually restart the service by using the start script (UNIX), or the Windows Services dialog box (Windows).

Some IDOL Content component configuration changes require you to reindex your data. In these cases, Micro Focus recommends that you use the `DREEXPORTIDX` or `DREEXPORTXML` index actions to export your content to an IDX or XML document. You can then initialize your index, make your configuration changes, and index the IDX or XML file back into the IDOL Content component.

Related Topics

- Export IDX Documents from the IDOL Content component

- Export XML Documents from the IDOL Content component

Delete and Restore Documents

This section describes how to delete documents from the IDOL Content component data index, and how to restore documents that were deleted by using the `DREDELETEDOC` index action.

Delete Documents by Reference

If you identify one or more documents by their reference field, you can delete them from the IDOL Content component data index by sending a `DREDELETEREF` action (case sensitive) from your Web browser:

```
http://
ContentHost
:indexPort/DREDELETEREF?Docs=docReferences&Field=fields&DREDbName=databaseName
```

where:

ContentHost	is the IP address or host name of the machine on which the IDOL Content component is installed.
indexPort	is the IDOL Content component index port (specified as `IndexPort` in the `[Server]` section of the IDOL Content component configuration file).
docReferences	is a list of the (percent-encoded) references of the documents that you want to delete. If the references contain plus symbols (+) or spaces, percent-encode each reference, then separate multiple references with plus symbols (there must be no space before or after a plus symbol).
fields	is the field that contains the reference specified in the `Docs` parameter when a document has more than one reference. Separate multiple references with commas or spaces (there must be no space before or after a comma). For example, the following action: `DREDELETEREF?Docs=myref1.txt&Field=REFFIELD1` deletes a document that contains the following references: `#DREFIELD REFFIELD1="myref1.txt"` `#DREFIELD REFFIELD2="myref2.txt"` but does not delete a document that contains the following references: `#DREFIELD REFFIELD1="myref2.txt"` `#DREFIELD REFFIELD2="myref1.txt"`
databaseName	is the name of the database that the documents that you want to delete belong to. This parameter is optional. If you do not specify a database, Content deletes the document from all databases that contain it.

For example:

```
http://12.3.4.56:20001/DREDELETEREF?Docs=http%3A%2F%2Fnews%2Enewssite%2Ecom%2Findex
%2Ehtml+http%3A%2F%2Fnews%2Enewssite%2Ecom%2Fcoverstory%2Ehtml
```

This action uses port `20001` to delete the documents with the specified URLs from Content, which is located on a machine with the IP address `12.3.4.56`.

Delete Documents and Ranges of Documents

If you identify individual documents or ranges of documents by their ID, you can delete them from the IDOL Content component data index by sending a `DREDELETEDOC` action (case sensitive) from your Web browser:

`http://ContentHost:indexPort/DREDELETEDOC?Docs=docIDs`

where:

ContentHost	is the IP address or host name of the machine on which the IDOL Content component is installed.
indexPort	is the IDOL Content component index port (specified as `IndexPort` in the `[Server]` section of the IDOL Content component configuration file).
docIDs	is one or more individual document IDs or ranges of document IDs that specify the documents that you want to delete. Use one or a combination of the following two formats: • *docID*. Specify the IDs of one or more documents. To specify multiple document IDs, separate them with plus symbols (there must be no space before or after a plus symbol). • `range=[min_docID,max_docID]`. Enter the document ID of the first and last document in a range of documents that you want to delete. If you want to combine the formats, separate them with plus symbols (there must be no space before or after a plus symbol).

For example:

`http://MyHost:20001/DREDELETEDOC?Docs=3+5+range=[7,10]`

This action uses port `20001` to delete the documents with the DocIDs `3`, `5`, `7`, `8`, `9`, and `10` from the IDOL Content component, which is located on a machine with the host name `MyHost`.

Delete Documents in IDOL Admin

After you use the Test Action tab in the Console page in the Control section of IDOL Admin, you can delete individual documents or sets of documents from the results list in the IDOL Admin interface.

To delete a set of documents in IDOL Admin

1. In the results list, click **Delete these documents**.

   The Delete Documents dialog box opens.

2. Click **Delete Documents**.

To delete an individual document in IDOL Admin

1. In the results list, click **Options** next to the document that you want to delete.

   The Edit query result dialog box opens.

2. Click **Delete**.

   The Delete dialog box opens.

3. Click **Delete**.

Restore Deleted Documents

If you used a DREDELETEDOC action to delete documents from the IDOL Content component data index, you can use a DREUNDELETEDOC action (case sensitive) to restore some or all the individual documents that you deleted. You can restore documents only if you have not run a DRECOMPACT index action since they were deleted (DRECOMPACT permanently removes unused documents and space from the index).

http://*ContentHost*:*indexPort*/DREUNDELETEDOC?Docs=*docIDs*

where:

ContentHost	is the IP address or host name of the machine on which the IDOL Content component is installed.
indexPort	is the IDOL Content component index port (specified as IndexPort in the [Server] section of the IDOL Content component configuration file).
docIDs	is one or more individual document IDs or ranges of document IDs that specify deleted documents that you want to restore. Use one or a combination of the following two formats: • *docID*. Specify the IDs of one or more deleted documents. If you want to specify multiple document IDs, separate them with plus symbols (there must be no space before or after a plus symbol). • range=[*min_docID,max_docID*]. Enter the document ID of the first and last document in a range of deleted documents that you want to restore. If you want to combine the formats, separate them with plus symbols (there must be no space before or after a plus symbol).

For example:

http://MyHost:20001/DREUNDELETEDOC?Docs=3+5+range=[7,10]

This action uses port 20001 to restore the documents with the DocIDs 3, 5, 7, 8, 9, and 10 to the IDOL Content component, which is located on a machine with the host name MyHost.

Locate Duplicate Documents

You can locate duplicate documents in the data index after indexing has taken place by using the DREDUPLICATE index action. This action locates duplicates in a specified subset of the content, and then removes them, tags a field, or moves the duplicate documents to another database.

```
http://
ContentHost:indexPort/DREDUPLICATE?ReferenceField=Field&DuplicateAction=Action
```

where:

ContentHost	is the IP address or host name of the machine on which the IDOL Content component is installed.
indexPort	is the IDOL Content component index port (specified as IndexPort in the [Server] section of the IDOL Content component configuration file).
Field	is a ReferenceType field used as the initial determination of whether two documents are a match.
Action	is the action to perform on a duplicate. The following options are available: • Delete. Deletes all duplicate documents. • Database. Moves all duplicate documents to a database. If you select the Database action, you must specify the database in the Database parameter. • Tag. Tags a specified field in the duplicate document. You must specify the field in the TagField index parameter. You can also specify a value to tag the field with by using the TagValue parameter. If you do not specify a TagValue, the field is tagged with the value **1**.

For example:

```
http://MyHost:20001/DREDUPLICATE?ReferenceField=DOCUMENT/DREREFERENCE&DuplicateActi
on=Database&Database=Duplicates
```

This action uses port 20001 to remove duplicates from the IDOL Content component that is located on the machine with the host name MyHost. Content uses the DREREFERENCE field to identify duplicate documents, and moves them to the Duplicates database.

```
http://MyHost:20001/DREDUPLICATE?ReferenceField=DOCUMENT/DREREFERENCE&DuplicateActi
on=Tag&TagField=DOCUMENT/DRETITLE&TagValue=Duplicate
```

In this example, Content initially uses the DREREFERENCE field to identify the duplicate documents, and then it changes the DRETITLE field to the value **Duplicate**.

To prevent Content from indexing duplicate documents, use the KillDuplicates parameter with the DREADD and DREADDDATA index actions.

For details on the other parameters that are available for the DREDUPLICATE index action, refer to the *IDOL Server Reference*.

Related Topics

Create and Delete Databases

This section describes how to create and delete databases in the IDOL Content component.

Create a New Database

You can create a new database in the IDOL Content component by using one of the following methods:

- Send a `DRECREATEDBASE` index action to Content.

- Add a database to the IDOL Content component configuration file.

- Create a new database in IDOL Admin.

> **NOTE:** The database name must not contain the following characters, which are syntax characters in other parameters that use database names:
>
> - ampersand (&)
>
> - plus sign (+)
>
> - percent (%)
>
> - comma (,)
>
> In addition, Micro Focus strongly recommends that you do not use white space characters (including \t, \r, \n, and so on) in your database names.

Send a DRECREATEDBASE Index Action

To create a new database by this method, send a `DRECREATEDBASE` action (case sensitive) from your Web browser:

`http://ContentHost:indexPort/DRECREATEDBASE?DREDbName=databaseName`

where:

`ContentHost`	is the IP address or host name of the machine on which the IDOL Content component is installed.
`indexPort`	is the IDOL Content component index port (specified as `IndexPort` in the `[Server]` section of the IDOL Content component configuration file).
`databaseName`	is the name of the database that you want to create.

For example:

```
http://12.3.4.56:20001/DRECREATEDBASE?DREDbName=Archive
```

This action uses port `20001` to create a new database named `Archive` on the IDOL Content component on the machine with the IP address `12.3.4.56`.

> **NOTE:** This action does not complete if the configuration file is read-only.

For information about other available parameters, refer to the *IDOL Server Reference*.

Add a Database to the IDOL Content component Configuration File

Use the following procedure to add a database to the IDOL Content component configuration file.

To add a new database to the IDOL Content component configuration file

1. Open the IDOL Content component configuration file in a text editor.

2. Find the `[Databases]` section. This section contains the `NumDBs` setting, which indicates how many databases Content currently contains. It also contains a section for each of these databases, with settings that apply to them. The names of the individual database sections use the format `DatabaseN`, where *N* numbers the databases in consecutive order, starting from 0 (zero).

   For example:

   ```
   [Databases]
   NumDBs=2

   [Database0]
   Name=News

   [Database1]
   Name=Archive
   ```

3. Increase the `NumDBs` setting by one.

   For example:

   ```
   [Databases]
   NumDBs=3
   ```

4. Create a section for the new database. The name of the section must use the format `DatabaseN`, where *N* numbers the databases in consecutive order, starting from 0 (zero).

   Use the `Name` parameter to specify a name for your new database.

   For example:

   ```
   [Databases]
   NumDBs=3

   [Database0]
   Name=News

   [Database1]
   ```

```
Name=Archive

[Database2]
Name=myNewDatabase
```

5. Add any other optional parameters for the database. For more information, refer to the *IDOL Server Reference*.

6. Save and close the configuration file.

7. Restart the IDOL Content component for your changes to take effect.

Create a new database in IDOL Admin

You can add a new database for the Content or Agentstore components, or IDOL Proxy, from the **Databases** page in the Control section of the IDOL Admin interface.

To add a new database to the IDOL Content component configuration file

1. Click **+ Add New Database**.The Add New Database dialog box opens.

2. In the **Database Name** box, type a name for the database.

3. Select the **Internal** check box if you want the database to be internal, and therefore hidden.

4. Select the **Read Only** check box if you want to prevent documents being added to the database.

5. Select a priority for the index action. This determines how IDOL Server queues the action.

6. Click **Create**.

Related Topics

- Display Online Help, on page 30

Modify a Database Configuration

You can modify the configuration for an IDOL Content component database by updating the parameters for the database in the configuration file, or by using the DREDBSETOPTIONS index action. For more information, refer to the *IDOL Server Reference*.

Alternatively, you can modify the configuration for a database by using the IDOL Admin interface.

To edit a database in IDOL Admin

1. In the **Control** menu, click **Databases**.

2. In the **Database Control** column, click [icon] next to the database that you want to edit.

   The Edit Database Properties dialog box opens.

3. Update the settings as required, then click **Edit**.

Delete a Database and All its Documents

You can delete an IDOL Content component database and all the documents that it contains either by sending a DREREMOVEDBASE action, or by using the IDOL Admin interface.

Delete a Database and All its Documents by Sending an Action

You can delete an IDOL Content component database and all the documents that it contains by sending a DREREMOVEDBASE action (case sensitive) from your Web browser:

`http://ContentHost:indexPort/DREREMOVEDBASE?DREDbName=databaseName`

where:

ContentHost	is the IP address or host name of the machine on which the IDOL Content component is installed.
indexPort	is the IDOL Content component index port (specified as IndexPort in the [Server] section of the IDOL Content component configuration file).
databaseName	is the name of the database that you want to delete. The documents in this database are deleted as well.

For example:

`http://MyHost:20001/DREREMOVEDBASE?DREDbName=Archive`

This action uses port 20001 to delete the Archive database and all documents that it contains from the IDOL Content component on the machine with host name MyHost.

> **NOTE:** This index action does not complete if the configuration file is read-only.

Delete a Database and All its Documents by using IDOL Admin

Use the following procedure to delete a database and all its documents by using the IDOL Admin interface.

To delete a database and all its documents by using IDOL Admin

1. In the **Control** menu, click **Databases**.

2. In the **Database Control** column, click next to the database that you want to delete.

   The Delete Database dialog box opens.

3. Click **Delete**.

Delete All Documents from a Database

You can delete all documents from an IDOL Content component database either by sending a DREDELBASE action, or by using the IDOL Admin interface.

Delete All Documents from a Database by Sending an Action

You can delete all documents from an IDOL Content component database by sending a DREDELBASE action (case sensitive) from your Web browser:

`http://ContentHost:indexPort/DREDELDBASE?DREDbName=databaseName`

where:

ContentHost	is the IP address or host name of the machine on which the IDOL Content component is installed.
indexPort	is the IDOL Content component index port (specified as IndexPort in the [Server] section of the IDOL Content component configuration file).
databaseName	is the name of the database whose documents you want to delete.

For example:

`http://12.3.4.56:20001/DREDELDBASE?DREDbName=Archive`

This action uses port 20001 to delete all documents from the Archive database on a machine with the IP address 12.3.4.56.

> **NOTE:** This index action does not complete if the configuration file is read-only.

Delete All Documents from a Database by using IDOL Admin

Use the following procedure to delete all documents from a database by using the IDOL Admin interface.

To delete all documents from a database by using IDOL Admin

1. In the **Control** menu, click **Databases**.

2. In the **Database Control** column, click next to the database from which you want to delete documents.

    The Delete Documents dialog box opens.

3. Click **Delete**.

Expire Documents

To ensure that the documents in your IDOL Content component are current, you can run an expiration operation, which deletes or archives documents that reach a specific age. You can expire documents:

- immediately, by sending a `DREEXPIRE` action

- at regular intervals, by setting up a scheduled expiration operation

By default, Content deletes documents when they expire. To archive them instead, enter the name of the database to use for archiving as the value of the `ExpireIntoDatabase` parameter in each of the IDOL Content component configuration file database sections.

Content can read the date that determines when a document expires from:

- a field in the document.

- the expiration time that you set for the database that contains the document.

Content does not expire the document if it is unable to determine whether a document must expire (for example, because it does not contain the expiration field, and the database has no expiration time).

Set up a Field Process for Expiration Dates

Use the following procedure to set up fields that determine when documents expire.

To set up fields that determine when documents expire

1. Open the IDOL Content component configuration file in a text editor.

2. Find the `[FieldProcessing]` section.

3. Add a new field process to the list of field processes that the `[FieldProcessing]` section contains.

   For example:

   ```
   [FieldProcessing]
   0=IndexFields
   1=IndexAndWeightHigher
   ```

4. To add a new field process, add a new line to the `[FieldProcessing]` section:

   ```
   [FieldProcessing]
   0=IndexFields
   1=IndexAndWeightHigher
   2=ExpireDateFields
   ```

   The listed field processes are numbered in consecutive order, starting from 0 (zero).

5. Create a section for your new field process in the configuration file. Create a property for the new process, and use the `PropertyFieldCSVs` parameter to identify the document fields that

determine whether documents expire (a document expires when the time in this field elapses).

For example:

```
[ExpireDateFields]
Property=SetExpireDate
PropertyFieldCSVs=*/DREEXPIRE,*/valid_time
```

6. Create a section for your new property in the configuration file, and set the `ExpireDateType` to **true** to indicate that the associated `PropertyFieldCSVs` fields hold the document expiration date.

   To include an extra offset from the expiration date in the field, set `ExpireAfterDelay` to the number of hours (positive or negative) by which to offset the field expiration date.

   For example:

   ```
   [SetExpireDate]
   ExpireDateType=True
   ExpireAfterDelay=12
   ```

Expire Immediately

To expire documents immediately, send a `DREEXPIRE` index action (case sensitive) from your Web browser:

`http://ContentHost:indexPort/DREEXPIRE`

where:

ContentHost	is the IP address or host name of the machine on which the IDOL Content component is installed.
indexPort	is the IDOL Content component index port (specified as `IndexPort` in the `[Server]` section of the IDOL Content component configuration file).

For example:

`http://MyHost:20001/DREEXPIRE`

This action uses port `20001` to expire the documents from the IDOL Content component located on a machine with host name `MyHost`.

Expire at Regular Intervals

Use the following procedure to expire documents at regular intervals.

To expire documents at regular intervals

1. Open the IDOL Content component configuration file in a text editor.

2. Find the `[Schedule]` section. If the configuration file does not contain a `[Schedule]` section, add one.

3. Specify the following parameters in the [`Schedule`] section:

   - `Expire`. Set to **True** to enable an expiration schedule.

   - `ExpireTime`. Specify the time (`hh:mm`) at which you want the expiration operation to start.

   - `ExpireInterval`. Specify the number of hours that elapse between individual expiration operations. Set to 0 (zero) to perform the operation daily.

   For example:

   ```
   [Schedule]
   Expire=True
   ExpireTime=00:00
   ExpireInterval=24
   ```

4. Set the following parameter in the individual database sections to specify where to archive the database documents when they expire:

   - `ExpireIntoDatabase`. Specify the name of the database in which to archive expired documents. To delete documents when they expire, do not add this setting.

   For example:

   ```
   [News]
   ExpireIntoDatabase=Archive
   ```

5. Save and close the configuration file.

6. Restart the IDOL Content component for your changes to take effect.

Change Document Metadata

Send a `DRECHANGEMETA` index action (case sensitive) from your Web browser to change the metadata (index date, expire date, or database) of documents. You can identify individual documents or document ranges by their references or IDs. (You can specify ranges only with IDs.)

```
http://
ContentHost
:indexPort/DRECHANGEMETA?Type=type&Refs=docReferences&Docs=docIDs&NewValue=value
```

where:

ContentHost	is the IP address or host name of the machine on which the IDOL Content component is installed.
indexPort	is the IDOL Content component index port (specified as `IndexPort` in the [`Server`] section of the IDOL Content component configuration file).
type	is the type of metadata whose value you are changing: - `date`. The index date to assign to the documents. The date you specify must be in a format that you set for `DateFormatCSVs` in the IDOL Content

	component configuration file.
	• `expiredate`. The expire date to assign to the documents. The date you specify must be in a format that you set for `DateFormatCSVs` in the IDOL Content component configuration file.
	• `database`. The database to move the documents to.
docReferences	is a list of (percent-encoded) references to the documents whose metadata is to change. If the references contain plus symbols (+) or spaces, percent-encode each reference, then separate multiple references with plus symbols (there must be no space before or after a plus symbol) and percent-encode the whole string again (using Internet Explorer or the ACI client).
docIDs	is one or more individual document IDs or a ranges of document IDs for the documents whose metadata is to change. Use one or a combination of the following formats:
	• *docID*. Specify the IDs of one or more documents. To specify multiple document IDs, separate them with plus symbols (there must be no space before or after a plus symbol).
	• `range=[`*min_docID,max_docID*`]`. Enter the document ID of the first and last document in a range of documents.
	To combine the formats, separate them with plus symbols (there must be no space before or after a plus symbol).
value	is the new metadata value to write into all the specified documents.

For example:

```
http://12.3.4.56:20001/DRECHANGEMETA?Type=database&Docs=3+5+range=
[7,10]&NewValue=Archive
```

This action uses port `20001` to reassign the documents with the IDs `3`, `5`, `7`, `8`, `9`, and `10` to the `Archive` database. Content is located on a machine with the IP address `12.3.4.56`.

> **NOTE:** By default, when you set the `Refs` parameter, Content searches all document references for the specified value. If your documents have multiple reference fields, and you want to search only one field name, set the `Field` parameter to the name of the reference field that you want to search.

Change Document Field Values

To change field values in documents, use the `DREREPLACE` index action, or the IDOL Admin interface.

Related Topics

- [About Fields, on page 81](#)

Change Document Field Values by Running an Index Action

You can change field values in documents by using the following index action:

`DREREPLACE?DatabaseMatch=Databases\n\nData#DREENDDATA\n\n`

> **NOTE:** This action requires a POST request method.

where:

Databases	is a list of one or more databases in which to replace the specified data. If you specify multiple databases, separate them with plus symbols (there must be no space before or after a plus symbol).

Data	is a list of field values to replace or delete in the IDOL Content component. Specify the fields and field values as follows:
	
	`#DREALL`
	`#DREDBNAME databaseName`
	`#DREDOCID id` or `#DREDOCREF url` or `#DRESTATEID stateID`
	`#DREFIELDNAME fieldName`
	`#DREFIELDVALUE fieldValue`
	`#DREFIELDVALUEIFNOTFOUND fieldValue`
	`#DREDELETEFIELDVALUE fieldValue`
	`#DREDELETESINGLEFIELDVALUE fieldValue`
	`#DREDELETEFIELD fieldName`
	`#DREFIELDBITOR value` (`#DREFIELDBITAND value` or `#DREFIELDBITXOR value`)
	
	The data fields are described in the table below.

`#DREENDDATA`	must exist at the end of an action parameter block to signify the end of the action.

`DREREPLACE` **Data Fields**

Field	Description
`#DREALL`	Match all documents. This option is restricted only by the `DatabaseMatch` parameter supplied as part of the `DREREPLACE` request, or by database read-only status.
`#DREDBNAME databaseName`	The databases in which to match all documents. To specify more than one database, separate them with plus signs (+). Content ignores the `DatabaseMatch` parameter for the documents identified by `#DREDBNAME`.
`#DREDOCID id`	The ID of the document that contains the field to change or delete.

DREREPLACE Data Fields, continued

Field	Description
#DREDOCREF *url*	The reference of the document that contains the field to change or delete.
#DRESTATEID *stateID*	The stored state ID of an array of documents that contains the field to change or delete. The state ID returns from a query where StoreState=True.
#DREFIELDNAME *fieldName*	The name of the field whose value you want to change or delete. Note that in XML documents, *fieldName* must be fully qualified (for example, XML/DOCUMENT/MYFIELD).
#DREFIELDVALUE *fieldValue*	The new value for the field specified by *fieldName* .
#DREFIELDVALUEIFNOTFOUND *fieldValue*	The new value for the field specified by *fieldName* if the *fieldName*/*fieldValue* pair does not already exist.
#DREFIELDVALUEIFNOFIELD *fieldValue*	The new value for the field specified by *fieldName* if the *fieldName* does not already exist in the document.
#DREDELETEFIELDVALUE *fieldValue* #DREDELETENOCASEFIELDVALUE *fieldValue*	The value that the field specified by *fieldName* must contain for the field to be deleted. By default, the value matching is not case sensitive (the NOCASE specifier is optional), but you can use #DREDELETECASEFIELDVALUE to match case sensitively.
#DREDELETECASEFIELDVALUE *fieldValue*	The value (case sensitive) that the field specified by *fieldName* must contain for the field to be deleted.
#DREDELETESINGLEFIELDVALUE *fieldValue* #DREDELETESINGLENOCASEFIELDVALUE *fieldValue*	The value that the field specified by *fieldName* must contain for IDOL Server to delete it. This option deletes only the first instance of the *fieldName*/*fieldValue* pair. By default, the value matching is not case sensitive (the NOCASE specifier is optional), but you can use #DREDELETESINGLECASEFIELDVALUE to match case sensitively.
#DREDELETESINGLECASEFIELDVALUE *fieldValue*	The value (case sensitive) that the field specified by *fieldName* must contain for IDOL Server to delete it. This option deletes only the first instance of the *fieldName*/*fieldValue* pair.
#DREDELETEWILDFIELDVALUE *fieldValue* #DREDELETEWILDCASEFIELDVALUE	The Wildcard value that the field specified by

DREREPLACE Data Fields, continued

Field	Description
fieldValue	*fieldName* must contain for the field to be deleted. This operation matches Wildcards for UTF-8 (that is, ? matches a single UTF-8 character). By default, the value matching is case sensitive (the CASE specifier is optional), but you can use #DREDELETEWILDNOCASEFIELDVALUE to match case insensitively.
#DREDELETEWILDNOCASEFIELDVALUE *fieldValue*	The Wildcard value (case insensitive) that the field specified by *fieldName* must contain for the field to be deleted. This operation matches Wildcards for UTF-8 (that is, ? matches a single UTF-8 character).
#DREDELETESINGLEWILDFIELDVALUE *fieldValue* #DREDELETESINGLEWILDCASEFIELDVALUE *fieldValue*	The Wildcard value that the field specified by *fieldName* must contain for the first instance of the field to be deleted. This operation deletes only a single instance of the *fieldName*/*fieldValue* pair. It matches Wildcards for UTF-8 (that is, ? matches a single UTF-8 character). By default, the value matching is case sensitive (the CASE specifier is optional), but you can use #DREDELETESINGLEWILDNOCASEFIELDVALUE to match case insensitively.
#DREDELETESINGLEWILDNOCASEFIELDVALUE *fieldValue*	The Wildcard value (case insensitive) that the field specified by *fieldName* must contain for the first instance of the field to be deleted. This operation deletes only a single instance of the *fieldName*/*fieldValue* pair. It matches Wildcards for UTF-8 (that is, ? matches a single UTF-8 character).
#DREDELETEFIELD *fieldName*	The field to delete. You do not require a #DRE*VALUE parameter.
#DREFIELDBITOR *value*	Generally a 32-bit decimal integer. - If this field is a NumericType field with the NumericIntegerOnly property, the value is a signed 64-bit integer. - If the field is a BitFieldType field, the value must be a hexadecimal value. Content performs the bitwise OR operation on the field identified by the previous #DREFIELDNAME. For NumericType fields, if the field is not present in the

DREREPLACE Data Fields, continued

Field	Description
	document, it adds a field with value `0000000000` first, and then applies the bit operation.
`#DREXMLFIELDVALUE`	A block of XML data to insert into an XML document under the previous `#DREFIELDNAME`.

To specify multiple instances of `#DREDOCID`, `#DREDOCREF`, or `#DRESTATEID`, separate the entries with a percent-encoded plus sign or comma.

For example: `#DREDOCID 1,3,7-10` or `#DREDOCID REF1+REF2+REF3`. You must also set `ReplaceAllRefs` to **True** to perform actions on multiple documents.

> **NOTE:** You cannot use `DREREPLACE` to update the values in `SecurityType` fields.

> **NOTE:** Micro Focus generally recommends that you do not replace values in `Index` fields, because Content must reindex the documents being changed, which can slow the update process. Changes to numerical fields, numerical date fields, or fields with other properties, occur without reindexing, and so occur quickly.

Related Topics

- Send Data with a POST Method, on page 58

Change Document Field Values by Using IDOL Admin

In the **Replace** tab on the Console page in the Control section of IDOL Admin, you can use the wizard to change field values or delete fields from indexed documents. For more information, refer to the *IDOL Admin User Guide*.

> **CAUTION:** You cannot recover your original data after you have replaced or deleted values from indexed documents.

To replace or delete field values in indexed documents

1. On the **Data Source** page, click an option to specify the fields to replace or delete:

   a. **Build the data to replace or delete manually**. Click this option to add a new data source.

   b. **UTF-8 encoded local file containing DREREPLACE parameters**. Click **Choose Files** to browse to a file that already specifies the preceding information.

   c. **Introduce some data that defines the fields and values to replace or delete**. Copy and paste in text that specifies:

      i. the documents in which to delete or replace the field values

       ii.   the values to substitute, or whether to delete the field

      iii.   `#DREENDDATANOOP` to mark the end of the index action parameters

2. Click **Next**.

   a. If you chose to build the data to replace or delete manually, the Choose Data page opens. You can add a new data source and specify the documents and field values that you want to edit.

   b. If you chose to replace or delete data from local files, or to copy and paste text that specifies the fields and values to replace or delete, the Choose Database page opens. Select one or more databases in which to change the data. Alternatively, you can read the target database from the data.

3. Click **Next**. The Summary page opens.

4. Select a priority for the `DREREPLACE` action, which determines how IDOL Server queues the action. The Summary section displays the action that will be sent to the server. The action is automatically constructed as you complete the wizard. It is shown for reference only; you cannot edit the action directly.

5. Click **Replace**.

IDOL Server performs the replace action on the specified documents. The changes are not visible in index searches until the index cache is flushed to disk. Click **Sync** to flush the index cache.

If you need to edit any of your settings, you can click **Previous** to move back through the pages of the wizard, or click **Reset** to clear all settings and start again from the beginning.

Examples

```
DREREPLACE?DATABASEMATCH=News+Archive HTTP/1.0\n
Content-Length:203\n\n

#DREDOCID 1\n
#DREFIELDNAME Price\n
#DREFIELDVALUE 10\n
#DREFIELDNAME Color\n
#DREFIELDVALUE Red\n

#DREDOCREF http://www.example.com/example/dynamic/autopage442.shtml\n
#DREFIELDNAME Country\n
#DREFIELDVALUE UK\n
#DREFIELDNAME Region\n
#DREFIELDVALUE South East\n
#DREFIELDNAME OnSale\n
#DREDELETEFIELDVALUE Yes\n

#DRESTATEID abcdefg-6\n
#DREFIELDNAME Fruit\n
#DREFIELDVALUEIFNOTFOUND mango\n
#DREDELETESINGLEFIELDVALUE apple\n
```

```
#DREDELETEFIELD XML/DOC/DELETEME\n
```

```
#DREENDDATANOOP\n\n
```

In this example, the DREREPLACE makes these changes in the News and Archive databases.

- In the document with the ID 1:

    ○ The value of the Price field changes to 10.

    ○ The value of the Color field changes to Red.

- In the document with the reference
  http://www.example.com/example/dynamic/autopage442.shtml:

    ○ The value of the Country field changes to UK.

    ○ The value of the Region field changes to South East.

    ○ The field OnSale is removed from the document if it has the value Yes.

- In the documents referenced in the state ID abcdefg-6:

    ○ The value of the Fruit field changes to mango if the Fruit/mango pair does not already exist.

    ○ If the Fruit field contains the value apple, a single instance of the Fruit/apple pair is deleted.

    ○ All instances of the XML/DOC/DELETEME field are deleted.

- #DREENDDATANOOP signifies the end of the action parameters.

Edit BitField Values

The following example demonstrates how to use a DREREPLACE action to edit the set information held in BitFieldType fields.

```
DREREPLACE?ReplaceAllRefs=True HTTP/1.0\n
Content-Length:203\n\n

#DREALL \n
#DREFIELDNAME BitField\n
#DREFIELDVALUEIFNOFIELD A008\n
#DREFIELDBITOR A008\n

#DREDOCID 1274\n
#DREFIELDNAME Workbook\n
#DREFIELDBITXOR 98\n

#DREDBNAME News\n
#DREFIELDNAME NewsSets\n
#DREFIELDBITAND D9\n

#DREENDDATANOOP\n\n
```

> **NOTE:** The `BitField`, `Workbook`, and `NewsSets` fields used in this example must be `BitFieldType` fields.

In this example, the `DREREPLACE` makes these changes.

- In all documents in the IDOL Content component:

  - If there is no `BitField` field, Content adds one with the value `A008`.

  - Content changes the value in the `BitField` field by performing a bitwise OR operation between the current value of the field and the value `A008`.

    For example, if the current value of the field is `8080`:

Hexadecimal	Binary	
8080	1000 0000 1000 0000	
A008	1010 0000 0000 1000	
	1010 0000 1000 1000	The new field value is **A088**

    The result of the bitwise OR is that any binary digits that were set to **1** (indicating that the document was part of the set) are left as they are. In addition, all the documents have the bits set to **1** for the third, thirteenth, and fifteenth sets, if it was not already set. In this way, all the documents are marked as part of sets 3, 13, and 15.

    In this example, the document is marked as part of set 3, while remaining a part of sets 7, 11, 13, and 15.

- In the document with ID 1274:

  - Content changes the value in the `Workbook` field by performing a bitwise XOR operation between the current value of the field and the value `98`. For example, if the current value of the field is `6C`:

Hexadecimal	Binary	
6C	0110 1100	
98	1001 1000	
	1111 0100	The new field value is **F4**

    The result of the bitwise XOR is that any binary digits that are set to **1** in either number (but not in both) are set to **1** in the result. If the document was not already part of sets 3, 4, or 7, Content adds it to those sets. However, if the bit is set to **1** or **0** in both numbers, it is set to **0** in the result. If a document is already part of sets 3, 4, or 7, it is removed from them.

    In this example, Content adds the document to sets 4 and 7. It remains part of sets 2, 5, and 6, but Content removes it from set 3.

- In all documents in the `News` database:

○ Content changes the value in the `NewsSets` field by performing a bitwise AND operation between the current value of the field and the value `D9`.

For example, if the current value of the field is `6C`:

Hexadecimal	Binary	
6C	0110 1100	
D9	1101 1001	
	0100 1000	The new field value is **48**

The result of the bitwise AND is that any binary digits that are set to **1** in both numbers are set to **1** in the result. If a document is already part of set 0, 3, 4, 6, or 7, it remains a part of those sets, but it is not be added if it was not originally part of those sets. It is also removed from any other sets it was initially part of.

In this example, the document remains in set 3 and set 6, but Content removes it from sets 2 and 5.

- `#DREENDDATANOOP` signifies the end of the action parameters.

Related Topics

- BitFieldType Fields, on page 103

Edit the Spelling Correction Cache

You can use the `DRESPELLCHECKCACHE` index action to edit entries in the IDOL Content component spelling correction cache without stopping the service. This action can edit or remove entries in the cache, or write the cache to disk.

`http://ContentHost:indexPort/DRESPELLCHECKCACHE?Action=Action`

where:

`ContentHost`	is the IP address or host name of the IDOL Content component.
`indexPort`	is the IDOL Content component index port (specified as `IndexPort` in the `[Server]` section of the IDOL Content component configuration file).
`Action`	is the action to perform in the spelling correction cache. The following options are available: - `Update`. Add a correction for the spelling, overwriting any current entry. Set `Original` to the incorrect spelling, and `Corrected` to the correct spelling for this word. If you do not specify the `Corrected` parameter, Content marks the word in the cache as not to be corrected. - `Remove`. Remove any entry for the specified incorrect spelling. Set the `Original` parameter to the incorrect spelling to remove.

| | • `Clear`. Remove all entries from the cache. |
| | • `Flush`. Write the spelling correction cache (`prx.db`) to disk. |

For example:

`http://localhost:9001/DRESPELLCHECKCACHE?Action=Update&Original=Barnana&Corrected=Banana`

This action uses port `9001` to edit the spelling correction cache in the IDOL Content component which is located on the local machine. It adds `Banana` as the correction for `Barnana`, overwriting any current entry.

`http://localhost:9001/DRESPELLCHECKCACHE?Action=Remove&Original=Barnana`

This action uses port `9001` to edit the spelling correction cache in the IDOL Content component which is located on the local machine. It removes any entry for `barnana` from the cache.

For details on other parameters that are available for the `DRESPELLCHECKCACHE` index action, refer to the *IDOL Server Reference*.

Related Topics

Use a Custom Term Weight File

A custom term weight file defines a set of terms and weights to use when the IDOL Content component calculates relevance weighting at query time. Content calculates term weights at index time, and stores a list of term weights for the index. You can retrieve the term details by using the `TermGetInfo` action.

In a distributed system, you might want to use a custom term weight file to ensure that all your child servers have the same term weighting, to ensure consistent results and relevant weighting across your indexes.

When the custom file exists, Content uses it for all queries unless you set the `CustomWeight` action parameter to **False** for an action. This action parameter is available for the `Query`, `Suggest`, `SuggestOnText`, `GetQueryTagValues`, `Summarize`, `TermGetBest`, and `TermGetInfo` actions.

You can use the `DREMODIFYTERMWEIGHT` index action to add, modify, or delete a custom term weight file.

`http://ContentHost:indexPort/DREMODIFYTERMWEIGHT?Filename=CustomWeightFile`

where:

`ContentHost`	is the IP address or host name of the IDOL Content component.
`indexPort`	is the IDOL Content component index port (specified as `IndexPort` in the `[Server]` section of the IDOL Content component configuration file).
`CustomWeightFile`	is the file name and path to the XML file that contains the stemmed terms and

	custom weight values to use.
	You can use the output from the `TermGetInfo` action, or an XML file with the same structure. Content extracts the term from the value of each `<autn:term>` tag, and the weight from the value of the `apcm_weight` attribute in the tag.
	For example:
	`<autn:term apcm_weight="nn">TERM</autn:term>`
	Content applies weight *nn* to `TERM` (`TERM` must be stemmed).

For example:

`http://localhost:9001/DREMODIFYTERMWEIGHT?Filename=C:\IDOL\Files\CustomWeight.xml`

By default, if you have an existing custom term weight file, Content updates the file with the values in the new one. Content adds any terms that do not exist in the current file, and updates the weight for any terms that do exist. You can set the `KeepExisting` parameter to **False** to delete the existing file and replace it with the new one.

You can also set `KeepExisting` to **False** without setting the `Filename` parameter. In this case, Content deletes the custom term weight file and uses the default index weights. For example:

`http://localhost:9001/DREMODIFYTERMWEIGHT?KeepExisting=False`

For more information about the `DREMODIFYTERMWEIGHT` index action, refer to the *IDOL Server Reference*.

Related Topics

- Display Online Help, on page 30

Compact the Data Index

You can reduce the space that the documents in the IDOL Content component data index take up by running a compact operation. In this operation, new documents fill up the space that has been created through the deletion of documents (similar to defragmentation).

You can compact the IDOL Content component data index

- immediately, by sending a `DRECOMPACT` action.

- at regular intervals, by setting up a scheduled `Compact` operation.

> **NOTE:** You can automatically back up the IDOL Content component data index whenever you send a `DRECOMPACT` action. Micro Focus recommends that you back up Content before you compact.

Related Topics

- Back up the Data Index Automatically, on page 452

- Back up the Entire IDOL Content component Data Index, on page 449

Compact the Data Index Immediately

You can compact the data index either by sending an action from your Web browser, or by using the IDOL Admin interface.

To compact the data index by sending an action from your Web browser

Send a `DRECOMPACT` action (case sensitive) from your Web browser:

`http://ContentHost:indexPort/DRECOMPACT`

where:

`ContentHost`	is the IP address or host name of the machine on which the IDOL Content component is installed.
`indexPort`	is the IDOL Content component index port (specified as `IndexPort` in the `[Server]` section of the IDOL Content component configuration file).

For example:

`http://MyHost:20001/DRECOMPACT`

This action uses port `20001` to compact the data content of an IDOL Content component that is located on a machine with host name `MyHost`.

You can restrict compaction to the `NodeTable` (document storage) stage by using the index parameter `Type`.

For example:

`http://MyHost:20001/DRECOMPACT?Type=NodeTable`

After you start a `DRECOMPACT` operation, you can pause the index action by sending an `IndexerGetStatus` action with the `IndexAction` parameter set to **pause**. You can also cancel the index action by sending an `IndexerGetStatus` action with the `IndexAction` parameter set to **Cancel**. In this case, the next `DRECOMPACT` operation that you send resumes where the previous compaction stopped.

To compact the data index by using IDOL Admin

1. In IDOL Admin, go to **Control > Console > Service Control**.

2. Click **Compact**.

   A confirmation message appears.

3. Click **Compact**.

You can monitor the progress of the `DRECOMPACT` action in the Recent Tasks panel.

Compact the Data Index at Regular Intervals

Use the following procedure to compact the data index at regular intervals.

To set up a schedule for compaction

1. Open the IDOL Content component configuration file in a text editor.

2. Find the `[Schedule]` section. If the configuration file does not contain a `[Schedule]` section, add one.

3. Specify the following parameters in the `[Schedule]` section:

    -   `Compact`. Set to **True** to enable a compacting schedule.

    -   `CompactTime`. Specify the time (`hh:mm`) when you want the `Compact` operation to start.

    -   `CompactInterval`. Specify the number of hours that elapse between individual compaction operations. Set to `0` (zero) if you want the operation to take place daily.

For example:

```
[Schedule]
Compact=True
CompactTime=00:00
CompactInterval=24
```

Initialize the Data Index

You can discard the data that your IDOL Content component data index contains and reset it to the state it was in when you first installed the IDOL Content component. This operation also clears the contents of the index queue, including any queued index actions. Your configuration file is not reset to its initial state.

You can initialize the data index either by sending an action from your Web browser, or by using the IDOL Admin interface.

To initialize the data index by sending an action from your Web browser

Send a `DREINITIAL` action (case sensitive) from your Web browser:

`http://`*ContentHost*`:`*indexPort*`/DREINITIAL?`

where:

ContentHost	is the IP address or host name of the machine on which the IDOL Content component is installed.
indexPort	is the IDOL Content component index port (specified as `IndexPort` in the `[Server]` section of the IDOL Content component configuration file).

Micro Focus also recommends adding the `InitialID` parameter to the `DREINITIAL` index action. The `DREINITIAL` index action resets the index, so it always returns its index ID as 1. When you set an `InitialID`, the IDOL Server `GetStatus` action reports the `InitialID` of the last `DREINITIAL` that it processed, which allows you to check whether the `DREINITIAL` index action has been completed. The `InitialID` also ensures that if the server receives the same request multiple times, it processes only the first one.

For example:

```
http://12.3.4.56:20001/DREINITIAL?InitialID=12345
```

This action uses port `20001` to reset the data index of an IDOL Content component that is located on a machine with the IP address `12.3.4.56` to its original state.

To initialize the data index by using IDOL Admin

1. In the **Service Control** tab in the Console page, click **Initialize IDOL Content**.

   A confirmation message appears.

2. Click **Delete it**.

You can monitor the progress of the `DREINITIAL` action in the Recent Tasks panel.

Validate the Data Index

You can check that your data indexes are correctly populated by validating the index. Validation checks the consistency of each of the subindexes in IDOL (node table, numeric, and so on) and logs information about any issues in the application log.

You can perform validation:

- at any time, using an index action or the IDOL Admin interface.

- on startup, by configuring validation parameters.

- automatically, following a server interruption.

> **NOTE:** The validation checks can take some time to run, depending on the size of your index.

Validate the Data Index Immediately

You can validate the data index either by sending an action from your Web browser, or by using the IDOL Admin interface.

To validate the data index by sending an action from your Web browser

To validate part of the data index immediately, send a `DREVALIDATE` index action (case sensitive) from your Web browser:

```
http://ContentHost:indexPort/DREVALIDATE?&Type=type
```

where:

`ContentHost`	is the IP address or host name of the machine on which the IDOL Content component is installed.
`indexPort`	is the IDOL Content component index port (specified as `IndexPort` in the `[Server]` section of the IDOL Content component configuration file).

type	is the type of subindex that you want to validate. You can specify only one subindex for each action. Use one of the following values: `DiskIndex` `Geospatial` `NodeTable` `Numeric` `PhraseIndex` `RefIndex` `SecIndex` `Unstemmed`

For example:

`http://12.3.4.56:20001/DREVALIDATE?&Type=SecIndex`

This index action uses port `20001` to validate the Content security index files of an IDOL Content component that is located on a machine with the IP address `12.3.4.56`.

To validate the data index by using IDOL Admin

1. On the **Service Control** tab of the Console page, click **Validate**.

   The Validate dialog box opens.

2. Select the check boxes next to the subindexes that you want to validate.

3. If you want to fail validation as soon as an error is found, select the **Fail validation immediately** check box.

4. Select the **Index Action Priority**, which determines how the action is queued.

5. Click **Validate**.

You can use the Recent Tasks panel to monitor the progress of the validation.

Validate Subindexes On Startup

You can configure the IDOL Content component to validate specified subindexes on startup. You can set the validation configuration parameters in the `[Server]` section, to specify which subindexes to validate every time you restart Content.

For more information, refer to the *IDOL Server Reference*.

Validate the Data Index Automatically

When the IDOL Content component restarts after an indexing interruption, it automatically runs all the internal validation checks on its subindexes.

These checks can take some time to run. If this is not desirable, you can turn off this validation check by setting the `ValidateOnInterruption` configuration parameter to **False**.

```
[Server]
ValidateOnInterruption=False
```

You can also use the `ValidateOptions` configuration parameter to configure other methods for speeding up the validation process. For more information, refer to the *IDOL Server Reference*.

Repair an Index After Validation Fails

You can regenerate a data index if the validation step fails. There are three ways to regenerate an index:

1. Use the `RegenerateIndex` configuration parameters and restart the server.

2. Use the `DREREGENERATE` index action while the server is running.

3. Use the IDOL Admin interface. This option provides a user interface for the `DREREGENERATE` index action.

All these methods recreate all the files for a particular data index without reindexing all your content.

> **NOTE:** You can also use these methods to regenerate an index after you make a field configuration change. However, you cannot use the `RegenerateRefIndex` or `DREREGENERATE RefIndex` option to change the configuration for `ReferenceType` fields.

Regenerate with a Server Restart

The `RegenerateIndex` configuration parameters allow you to recreate the data indexes.

To regenerate an index by using the Regenerate configuration parameters

1. Open the IDOL Content component configuration file in a text editor.

2. In the `[Server]` section, add the `RegenerateIndex` configuration parameter for the index that you want to regenerate, and set it to **True**. The following parameters regenerate the indexes that correspond to some of the validation steps.

   - `RegenerateNumericIndex`

   - `RegenerateRefIndex`

   - `RegenerateSecIndex`

   - `RegenerateUnstemmedIndex`

   For more information about these parameters, refer to the *IDOL Server Reference*.

   > **NOTE:** If validation fails for the `nodetable` or `diskindex` stages, you must reindex your content to repair the index.

3. Save and close the configuration file.

4. Restart the IDOL Content component for your changes to take effect.

On startup, Content regenerates the specified index.

5. In the configuration file, delete the Regenerate*Index* parameter, or set it to **False**. This step ensures that Content does not waste time regenerating the index again when you next restart.

Regenerate with an Index Action

The DREREGENERATE action allows you to recreate the data indexes while the server is running. Use this method if you need the IDOL Content component to be accessible while you regenerate.

> **NOTE:** You cannot regenerate the unstemmed index by using DREREGENERATE. In this case, you must use the RegenerateUnstemmedIndex configuration parameter.

To regenerate the data index immediately, send a DREREGENERATE index action (case sensitive) from your Web browser:

http://*ContentHost*:*indexPort*/DREREGENERATE?&Type=*type*

where:

ContentHost	is the IP address or host name of the machine on which the IDOL Content component is installed.
indexPort	is the IDOL Content component index port (specified as IndexPort in the [Server] section of the IDOL Content component configuration file).
type	is the type of subindex that you want to regenerate. You can specify multiple indexes in a comma-separated list. The following values are available for repairing a validation failure: • Numeric • RefIndex • SecIndex > **NOTE:** The DREREGENERATE index action also has options that allow you to regenerate field indexes to update the field configuration. See Update Field Configuration, on page 87.

For example:

http://12.3.4.56:20001/DREREGENERATE?&Type=Numeric,RefIndex

This index action uses port 20001 to regenerate the NumericType and ReferenceType field indexes of an IDOL Content component that is located on a machine with the IP address 12.3.4.56.

Regenerate with IDOL Admin

You can also regenerate the data index by using the **Regenerate** function in the **Service Control** tab in the Console page of the IDOL Admin interface.

To regenerate the data index by using IDOL Admin

1. In the **Service Control** tab in the Console page, click **Regenerate**.

   The Regenerate dialog box opens.

2. In the **Type** list, select the type of subindex that you want to regenerate.

3. In the **Priority** list, select the priority for the action, which determines how the action is queued.

4. Click **Regenerate**.

Chapter 23: Back up IDOL Server

Micro Focus recommends that you back up your IDOL components at regular intervals to ensure that you always have current copies of the data that IDOL Server stores.

Back up Content

The IDOL Content component stores content data in its data index. You can back up the entire data index, or export IDX and XML documents from specific IDOL databases.

> **NOTE:** To restore backed up content, see Restore Content, on page 458.

Back up the Entire IDOL Content component Data Index

You can back up the entire IDOL Content component data index in several ways:

- Immediately, using a `DREBACKUP` index action.

  > **NOTE:** You can also back up Content by using the `BackupServer` ACI action. However, Micro Focus generally recommends that you use the index action, because `BackupServer` uses an ACI thread, which reduces the number of other processes that Content can run.

- At regular intervals, by setting up a scheduled backup.

- Automatically, whenever you send a `DRECOMPACT` index action.

- Dynamically, by taking a snapshot of the data.

Related Topics

- Back up the Data Index Immediately, on the next page

Back up the Data Index Immediately

Send a `DREBACKUP` action (case sensitive) from your Web browser to copy all the IDOL Content component data index `*.DB` files to a new location.

```
http://ContentHost:indexPort/DREBACKUP?Path=path
```

where:

ContentHost	is the IP address or host name of the Content machine.
indexPort	is the Content index port (specified by the `IndexPort` parameter in the `[Server]` section of the Content configuration file).
path	is the path to the location where you want to create the backup. In Windows, you can use a Unicode path.

For example:

```
http://MyHost:20001/DREBACKUP?Path=E:\Backup
```

This action uses port `20001` to create a backup of the Content data index on `E:\Backup`. The Content whose data index is backed up is located on a machine with the host name `MyHost`.

You can set the optional parameter `CheckIndexUpdates` to **True** if you want to check for updates to the index since the last backup. In this case, Content skips the backup if the index has not been modified.

`DREBACKUP` also accepts the optional parameter `HostDetails`. Set `HostDetails` to **True** to write the backup to a subdirectory in the specified directory path. Content names this directory using its host and port. In this way, a series of servers under a Distributed Index Handler (DIH) can all accept the same directory as input, and they write to a subdirectory named with their own host and port.

Back up the Data Index at Regular Intervals

Use the following procedure to back up the data index at regular intervals.

To back up the data index at regular intervals

1. Open the IDOL Content component configuration file in a text editor.

2. Find the `[Schedule]` section. If the configuration file does not contain a `[Schedule]` section, add one.

3. Specify the following parameters in the `[Schedule]` section:

`Backup`	Set to **True** to enable a schedule backup.
`BackupCheckIndexUpdates`	Set to **True** to check for updates to the index before

	Content runs a backup. When the index has not changed, Content skips the scheduled backup.
BackupCompression	Set to **True** to compress the Content data index before the backup.
BackupTime	Specify the time (hh:mm) when you want the backup to start.
BackupInterval	Specify the number of hours that elapse between individual backups. Set to 0 (zero) to back up daily.
BackupDir*N*	Set to the path to the location where you want to create the data index backup. You must specify one directory for each of the NumberOfBackups.
BackupMaintainDirStructure	Set to **True** to maintain the directory structure when the data index is backed up.
NumberOfBackups	Specify the number of times that you want to back up the Content data index. You can cycle the backing up procedure by specifying multiple backups (the number of backups you specify must correspond to the number of BackupDir*N* directories that you specify). Content runs multiple backups in the following way: • It creates the first backup at the specified BackupTime. • It creates the next one after the specified BackupInterval, and so on. • After the data index has been backed up as many times as you specified for NumberOfBackups, it overwrites the first backup the next time that the specified BackupInterval elapses. This process means that you always have a current set of data index backups.
BackupRetryAttempts	Specify the number of times that you want Content to reattempt a backup if it fails. After this number of attempts, Content cancels the operation.
BackupRetryPause	The number of seconds that you want Content to wait between backup attempts.

For example:

```
[Schedule]
Backup=True
BackupCompression=True
BackupTime=00:00
BackupInterval=24
```

```
BackupMaintainStructure=True
BackupRetryAttempts=3
BackupRetryPause=5
NumberOfBackups=3
BackupDir0=E:\DataIndex_Backup0
BackupDir1=E:\DataIndex_Backup1
BackupDir2=E:\DataIndex_Backup2
```

Back up the Data Index Automatically

Use the following procedure to back up the data index automatically whenever you send a DRECOMPACT action.

To back up the data index automatically

1. Open the IDOL Content component configuration file in a text editor.

2. Find the [Schedule] section. If the configuration file does not contain a [Schedule] section, add one.

3. Specify the following parameters in the [Schedule] section:

PreCompactionBackup	Set this parameter to **True** to perform a backup of key files automatically whenever a DRECOMPACT action is sent (either from a Web browser, or by an IDOL Content component schedule). Content backs up the files before it compacts the data index, so that you can restore them. True is the default value.
	The backup maintains the Content directory structure. You can compress the backup by setting BackupCompression to **True** and setting BackupCompressionLevel.
PreCompactionBackupPath	If you set PreCompactionBackup to **True**, you can use PreCompactionBackupPath to specify the path to the directory where you want to back up files. By default, it uses the internalbackup directory in the Content data directory.
	If Content shuts down without completing a compaction, it uses the contents of this directory to restore itself.

Back up the Data Index Dynamically

If your IDOL Content component storage is a SAN with disk-snapshot capabilities, you can perform a hot backup (snapshot) of the data index.

To back up the data index dynamically

1. Send the DREFLUSHANDPAUSE action to prepare Content for the hot backup by flushing all files to disk and pausing indexing:

   http://*IDOLhost*:*indexPort*/DREFLUSHANDPAUSE?

2. Note the index ID returned from this action (for example, `INDEXID=41`); you need this ID later to restart the paused indexing queue.

3. Poll Content with the `IndexerGetStatus` action to view the status of the flush and pause process:

```
<autnresponse>
<action>INDEXERGETSTATUS</action>
<response>SUCCESS</response>
<responsedata>
<item>
    <id>41</id>
    <origin_ip>127.0.0.1</origin_ip>
    <received_time>2006/10/06 13:47:15</received_time>
    <start_time>2006/10/06 13:47:16</start_time>
    <end_time>Finished</end_time>
    <duration_secs>0</duration_secs>
    <percentage_processed>0</percentage_processed>
    <status>-16</status>
    <description>Indexing Paused</description>*
    <index_command>/DREFLUSHANDPAUSE? HTTP/1.1</index_command>
</item>
</responsedata>
</autnresponse>
```

When the returned status is `-16` (`Indexing Paused`), you can perform the hot backup.

4. Take the snapshot according to the procedures defined by your SAN.

5. After the snapshot completes, restart the indexing queue. Issue another `IndexerGetStatus` action, this time specifying a `Restart` action:

    `action=IndexerGetStatus&Index=`*indexNum*`&IndexAction=restart`

    where *indexNum* is the index ID (for example, 41) returned from `DREFLUSHANDPAUSE`.

Export IDX Documents from the IDOL Content component

You can send a `DREEXPORTIDX` action (case sensitive) from your Web browser to export IDX documents from one or more IDOL Content component databases. (Use `DREEXPORTXML` to export XML documents.) Use the following syntax:

```
http://
ContentHost
:
indexPort
/DREEXPORTIDX?FileName=
fileName
&Compress=True|False&DatabaseMatch=
databaseCSV&BatchSize=size&MinDate=minDate&MaxDate=maxDate
```

where:

ContentHost	The IP address or name of the IDOL Content component machine.
indexPort	The IDOL Content component index port (the value of `IndexPort` in the `[Server]` section of the IDOL Content component configuration file).
fileName	The path to the location where you store the IDX files to export. Double-byte file names are acceptable. The path must include a basic file name, which Content postfixes with incremental numbers and an appropriate extension. If you do not specify a file name, Content exports the files to the current working directory (`IDOLserver\IDOL\content`), and Content creates a file name in the format `AUTN-IDX-EXPORT-host-port-date>-time-incrementalNumber.extension`.
`Compress`	Whether to compress the exported files. Set `Compress` to **False** if you do not want to compress the files.
databaseCSV	If you do not want to export documents from all Content databases, enter one or more databases to restrict the export to. If you want to specify multiple databases, separate them with plus symbols, commas, or spaces (there must be no space before or after plus symbols or commas).
size	The maximum number of document sections that you want to export to one IDX file. The default value is 100,000 sections.
minDate	The earliest creation date or time that a document can have to export.
maxDate	The latest creation date or time that a document can have to export.

> **NOTE:** For additional action parameters, refer to the *IDOL Server Reference*.

Example 1:

```
http://12.3.4.56:20001/DREEXPORTIDX?FileName=/export/data/backup/output&Compress=True&DatabaseMatch=News,Archive&BatchSize=1000&MinDate=01/01/2003&MaxDate=01/01/2004
```

In this example, Content exports all IDX documents that have dates between the first of January 2003 and the first of January 2004. It exports from the `News` and `Archive` databases to a series of compressed files in the `/export/data/backup` directory. It names the files created in this directory `output-0.idx.gz`, `output-1.idx.gz` and so on.

Example 2:

```
http://12.3.4.56:20001/DREEXPORTIDX?
```

In this example, the IDOL Content component exports all IDX documents to a series of compressed files in the current Content working directory (`IDOLserver\IDOL\content`). It names the files created in this directory `AUTN-IDX-EXPORT-12.04.2005-02.15.41-0.idx.gz`, `AUTN-IDX-EXPORT-12.04.2005-02.15.41-1.idx.gz` and so on.

> **NOTE:** Content does not split the sections of a multiple section document across batches. If the exact `BatchSize` splits a multiple-section document, Content does not use the exact value.

> **NOTE:** You do not need to uncompress compressed IDX files before you index them. For example, the action `DREADD?output-0.idx.gz` indexes the `output-0.idx.gz` file correctly without you having to uncompress the file first.

Export XML Documents from the IDOL Content component

You can send a `DREEXPORTXML` action (case sensitive) from your Web browser to export XML documents from one or more Content databases (use `DREEXPORTIDX` to export IDX documents):

```
http://
ContentHost
:
indexPort
/DREEXPORTIDX?FileName=
fileName
&Compress=True|False&DatabaseMatch=
databaseCSV&BatchSize=size&MinDate=minDate&MaxDate=maxDate
```

where:

ContentHost	The IP address or name of the IDOL Content component machine.
indexPort	The IDOL Content component index port (the value of `IndexPort` in the `[Server]` section of the IDOL Content component configuration file).
fileName	The path to the location where you store the XML files to export. Double-byte file names are acceptable. The path must include a basic file name which Content postfixes with incremental numbers and an appropriate extension. If you do not specify a file name, Content exports the files to the current working directory (`IDOLserver\IDOL\content`), and Content creates a file name in the format `AUTN-XML-EXPORT-host-port-date-time-incrementalNumber.extension`.
`Compress`	Whether to compress the exported files. Set `Compress` to **False** if you do not want to compress the files.
databaseCSV	One or more databases to export documents from. By default, Content exports from all databases. To specify multiple databases, separate them with plus symbols, commas, or spaces (there must be no space before or after plus symbols or commas).
size	The maximum number of document sections to export to one IDX file. The default value is 100,000 sections.
minDate	The earliest creation date or time that a document can have to export.
maxDate	The latest creation date or time that a document can have to export.

> **NOTE:** For additional action parameters, refer to the *IDOL Server Reference*.

Example 1:

```
http://12.3.4.56:20001/DREEXPORTXML?FileName=/export/data/backup/output&Compress=True&DatabaseMatch=News,Archive&BatchSize=1000&MinDate=01/01/2003&MaxDate=01/01/2004
```

In this example, Content exports all XML documents that have dates between 1 January 2003 and 1 January 2004. It exports from the `News` and `Archive` databases to a series of compressed files in the `/export/data/backup` directory. Content names the files created in this directory `output-0.xml.gz`, `output-1.xml.gz` and so on.

Example 2:

```
http://12.3.4.56:20001/DREEXPORTXML?
```

In this example, Content exports all XML documents to a series of compressed files in the Content current working directory (`IDOLserver\IDOL\content`). It names the files created in this directory `AUTN-XML-EXPORT-12.04.2005-02.15.41-0.xml.gz`, `AUTN-XML-EXPORT-12.04.2005-02.15.41-1.xml.gz` and so on.

> **NOTE:** Content does not split multiple-section documents across chunks, so the specified `BatchSize` is not used exactly if it splits a multiple-section document.

> **NOTE:** You do not need to uncompress compressed XML files before you index them. For example, the action `DREADD?output-0.xml.gz` indexes the `output-0.xml.gz` file correctly without you having to uncompress the file first.

Use IDOL Admin to Export Indexed Documents

You can export IDX or XML documents from one or more IDOL Server databases. You can also export the entire data index.

To export indexed documents

1. In the **Control** menu, click **Console**.

2. Click the **Export** tab.

3. In the **Export Location** box, type the path to the directory in which to store the exported documents.

4. In the **Export Index Action** box, select the type of files you want to export–either **IDX** or **XML**. This selects either the `DREEXPORTIDX` or `DREEXPORTXML` action, which is displayed and automatically updated as you fill in the form.

5. Click **Select Databases**, then select the check boxes next to all the databases you want to export documents from. If you do not select any databases, IDOL Server exports documents from all databases.

6. Complete the following fields to specify the export task in detail, including which documents to export.

Index Action Priority	Change or set the priority of the exporting job.
Compress	Select whether to compress the exported files.
Delete Documents	Select whether to delete the documents from IDOL Server after exporting them. (Documents are deleted only if the export is successful.)
Add Host Details	Select whether to add host details to the exported file name.
Batch Size	Type the number of document sections to export to one file. Click **X** to delete the value.
Max Date	Select the latest creation date and time that a document can have to be exported. Click **Now** in the calendar to automatically select the current date and time. Click **X** to delete the value.
Min Date	Select the earliest creation date and time that a document can have to be exported. Click **Now** in the calendar to automatically select the current date and time. Click **X** to delete the value.
Max ID	Type the maximum document ID number (inclusive) to export. Click **X** to delete the value.
Min ID	Type the minimum document ID number (inclusive) to export. Click **X** to delete the value.
Match ID	If you do not want to export all documents, type the IDs of documents to export. Separate multiple IDs with spaces or plus symbols. There must be no space before or after a plus symbol. You can also specify a range of IDs, using a hyphen. Click **X** to delete all values.
Match Reference	If you do not want to export all IDX documents, type the references of the documents to export.
	If the references contain plus symbols (+) or spaces, percent-encode each reference, then separate multiple references with plus symbols (there must be no space before or after a plus symbol) and percent-encode the whole string again (using Internet Explorer or the ACI client). Click **X** to delete all values.
State Match ID	Type the state token of a list of documents to export. If you specify the token name only, all documents listed in the token are exported. If you add a (zero-based) index range, or individual numbers separated by plus symbols (+), in square brackets after the token name, only that range or set of documents are exported.
	Click **X** to delete all values.

7. Click **Export** to export the documents to the specified location.

Archive Index Actions

You can configure the IDOL Content component to store a record of all index actions that it runs. You can use the archive of index actions to replay index actions, for example after you restore to a backup.

To configure the IDOL Content component to archive index actions

1. Open the IDOL Content component configuration file in a text editor.

2. In the `[Paths]` section, set the `ArchivePath` parameter to the path where you want to store the archive. For example:

   ```
   [Paths]
   ArchivePath=./archive
   ```

3. Save and close the configuration file.

4. Restart the IDOL Content component for your changes to take effect.

When you have configured `ArchivePath`, Content saves all index actions to the specified path. It also archives any state tokens that have been referenced in the `DREREPLACE`, `DRECHANGEMETA`, `DREDELETEDOC`, `DREEXPORTIDX`, `DREEXPORTXML`, and `DREEXPORTREMOTE` index actions.

During a `DREINITIAL` index action, Content stores the contents of the archive directory in a subdirectory, so that you can replay actions after you restore from a backup. If you restore to a time, Content automatically replays archived index actions between the last backup file and the restore time.

> **TIP:** When you configure an `ArchivePath`, you can optionally stop Content from indexing a particular index action by using the `NoArchive` index action parameter for any index action.
>
> However, if you use the `NoArchive` parameter for an index action that changes the state of the index, and then use `DREINITIAL` and `RestoreTime` to restore the index, the restore process does not include these changes, and the final index state might be different.

Related Topics

- [Restore to a Time](#)

Restore Content

This section describes how to restore your IDOL Content component data index from a backup file or an exported file.

Return a List of Backup Files

To find out what backup files are available for your IDOL Content component, you can send the `GetBackupData` action. This action returns a list of the backup files that you can use to restore your

index.

`http://ContentHhost:ACIport/action=GetBackupData`

where:

`ContentHost`	is the IP address or host name of the machine on which Content is installed.
`ACIPort`	is the Content ACI port (specified by the `Port` parameter in the `[Server]` section of the Content configuration file).

To find out which backup file Content would use to restore to a particular backup time, you can set the `GetBackupForRestoreTime` parameter to the backup time that you want. In this case, the action returns the single backup file that is used to restore to this time. For a list of available date formats, see .

Restore from a Backup File

To restore a data index to an IDOL Content component, you can send a `DREINITIAL` index action (case sensitive) from your Web browser, including the path to a backup file that you created by using the `DREBACKUP` index action.

> **TIP:** You can also restore Content by using the `RestoreServer` ACI action.

`http://ContentHost:indexPort/DREINITIAL?Path=path`

where:

`ContentHost`	is the IP address or host name of the machine on which Content is installed.
`indexPort`	is the Content index port (specified by the `IndexPort` parameter in the `[Server]` section of the Content configuration file).
`path`	is the path to the location of the backup, created by using the `DREBACKUP` index action. In Windows, you can use a Unicode path.

For example:

`http://12.3.4.56:20001/DREINITIAL?Path=E:\DataIndex_Backup`

This action uses port `20001` to restore the files backed up on `E:\DataIndex_Backup` to a Content located on a machine with the IP address `12.3.4.56`.

If you create a backup by using the `HostDetails` parameter, you can restore these backups by using the `HostDetails` parameter with your `DREINITIAL`:

`http://IDOLhost:port/DREINITIAL?Path=path&HostDetails=True`

> **NOTE:** When you restore from a backup by using `DREINITIAL`, Content does not reindex the data.
>
> When you want to update a configuration parameter that requires you to reindex your data, you can use `DREEXPORTIDX` or `DREEXPORTXML` to export an IDX or XML file. You can then initialize your index, make your configuration changes, and index the IDX or XML file back into Content.

Related Topics

- Export IDX Documents from the IDOL Content component
- Export XML Documents from the IDOL Content component

Restore to a Time

To restore your index to a particular time, you can send a `DREINITIAL` index action (case sensitive), and set the `RestoreTime` parameter.

`http://`*`ContentHost`*`:`*`indexPort`*`/DREINITIAL?RestoreTime=`*`Time`*

where:

`ContentHost`	is the IP address or host name of the machine on which Content is installed.
`indexPort`	is the Content index port (specified by the `IndexPort` parameter in the `[Server]` section of the Content configuration file).
`Time`	is the time that you want to restore to. For a list of available time and date formats, see Date formats , on page 249. **NOTE:** If you use a format that contains a space (such as a date and time), you must percent-encode the parameter value. Content processes date and time values according to the timezone of the server. To avoid confusion or inconsistent results, Micro Focus recommends that you use the epoch seconds format where possible. The `GetBackupData` action returns the available backup times in epoch seconds, and the `IndexerGetStatus` action returns the times for index jobs in epoch seconds if you set `EpochTime` to **True**, so that you can find the time that you want to restore to.

When you restore to a time, Content:

- restores to the latest backup from before the specified time.
- runs any archived index actions from between the backup time and the specified restore time. If you have not configured an `ArchivePath`, the index remains in the backup state.

To find out which backup file Content would use to back up to a particular time, use the `GetBackupData` action, with the `GetBackupForRestoreTime` parameter. See Return a List of Backup Files, on page 458.

Restore Exported Content

To restore index data from an exported IDX or XML file, created with the `DREEXPORTIDX` or `DREEXPORTXML` index actions, you can index the file by using the `DREADD` index action.

You can use this method to reindex content after you make a configuration change that requires a reindex.

Back up Categories, Taxonomies, and Cluster Jobs

You can back up categories, taxonomies, and cluster jobs (snapshots, cluster results, and spectrographs) either by submitting an action, or by using the IDOL Admin interface.

To back up the IDOL Category component categories, taxonomies, and cluster jobs by submitting an action

- Run a `BackupServer` action, with the `Path` parameter set to the path to the directory where you want to store the backup for your categories, taxonomies, and cluster jobs:

    `http://`
    `CategoryHost:CategoryPort/action=BackupServer&Path=C:\Backups\Category\`

    where:

CategoryHost	is the IP address or name of the machine on which the IDOL Category component is installed.
CategoryPort	is the ACI port of the IDOL Category component. **NOTE:** In a unified IDOL Server, you must send the `BackupServer` action directly to the IDOL Category component (not to the default IDOL Proxy port).

NOTE: During the backup process, Category stores temporary files in the directory that you can configure with the `BackupDir` configuration parameter. It stores the final backup file in the path that you specify.

NOTE: While Category is processing a `BackupServer` action, you cannot add, change, or remove categories, or run cluster jobs. You can, however, run actions that do not affect the data that is backed up. For example, you can use the `ClusterResults`, `ClusterSGDataServe`, `CategoryQuery` actions.

To back up IDOL Category component categories, taxonomies, and cluster jobs by using IDOL Admin

- On the **Category Backup/Restore** tab, click **Backup**.

    The `Backup` maintenance action backs up the category data to the backup directory that you specified by using the `BackupDir` parameter in the [`Server`] section of the `category.cfg` configuration file. The **Backup Action String** field shows the action that is sent to IDOL Server.

Related Topics

- Restore Categories, Taxonomies, and Cluster Jobs, below

Restore Categories, Taxonomies, and Cluster Jobs

You can restore backups of categories, taxonomies, and cluster jobs either by running a `RestoreServer` action, or by using the IDOL Admin interface.

To restore data from a backup by running an action

1. Send the `RestoreServer` action to the IDOL Category component. Set the `FileName` parameter to the name of the backup file that you want to restore from.

   NOTE: In a unified IDOL Server, you must send the `RestoreServer` action directly to the IDOL Category component (not to the default IDOL Proxy port).

   For example:

   ```
   http://localhost:9020/action=RestoreServer&FileName=C:\Backups\Category\myba
   ckup.zip
   ```

2. Send the `CategorySyncCatDRE` action to synchronize the restored IDOL Category component with the IDOL Agentstore component. For example:

   ```
   http://localhost:9020/action=CategorySyncCatDRE
   ```

NOTE: While the IDOL Category component is processing a `RestoreServer` action, you cannot run any category, cluster, or taxonomy actions.

To restore data from a backup by using IDOL Admin

- On the **Category Backup/Restore** tab in the Console page, click **Restore**.

  Category restores the category data from your most recent backup file, and displays a confirmation message. The **Restore Action String** field shows the action that is sent to IDOL Server.

Related Topics

- Back up Categories, Taxonomies, and Cluster Jobs, on the previous page

Back up Users, Roles, Agents, and Profiles

You can back up users, roles, agents, and profiles, either by submitting an action, or by using the IDOL Admin interface.

The backup includes all the data in the IDOL Community component, including a `UserExport` XML file and the configuration file.

To back up the Community users, roles, agents, and profiles by submitting an action

- Run a `BackupServer` action, with the `Path` parameter set to the path to the directory where you want to store the backup for your categories, taxonomies, and cluster jobs:

  ```
  http://
  CommunityHost:CommunityPort/action=BackupServer&Path=C:\Backups\Category\
  ```

  where:

`CommunityHost`	is the IP address or name of the machine on which Community is installed.
`CommunityPort`	is the Community ACI port. **NOTE:** In a unified IDOL Server, you must send the `BackupServer` action directly to Community (not to the default IDOL Proxy port).

NOTE: While Community is processing a `BackupServer` action, you cannot add, change, or remove community data. You can, however, run actions that do not affect the data that is backed up (such as `UserRead`, `Community`, and `Security`).

To back up Community users, roles, agents, and profiles by using IDOL Admin

1. On the **Backup/Restore** tab (in **Control** section), optionally update the backup directory, and then click **Backup**.

   The `BackupServer` maintenance action backs up the community data to the specified backup directory.

Related Topics

-

Restore Users, Roles, Agents, and Profiles

You can restore backups of users, roles, agents, and profiles either by running a `RestoreServer` action, or by using the IDOL Admin interface.

IMPORTANT: The restore action does not restore the configuration file from the backup. However, the backup file does include the configuration file, which you can restore manually, if required.

To restore data from a backup by running an action

1. Send the `RestoreServer` action to Community. Set the `FileName` parameter to the name of the backup file that you want to restore from.

> **NOTE:** In a unified IDOL Server, you must send the `RestoreServer` action directly to Community (not to the default IDOL Proxy port).

For example:

```
http://localhost:9030/action=RestoreServer&FileName=C:\Backups\Community\myb
ackup.zip
```

> **NOTE:** While Community is processing a `RestoreServer` action, you cannot add, change, or remove community data. You can, however, run actions that do not affect the data that is backed up (such as `UserRead`, `Community`, and `Security`).

To back up Community users, roles, agents, and profiles by using IDOL Admin

- On the **Backup/Restore** tab (in **Control** section), optionally update the backup directory to the directory that you want to restore from, and then click **Restore**.

   The `RestoreServer` maintenance action restores the community data from the specified backup directory.

Related Topics

-

Export Users, Roles, Agents, and Profiles

You can export IDOL Community component users, roles, agents, and profiles to a specified XML file from where they can be imported into a Community again. For example, you can use this process to transfer Community to a different platform.

```
http://CommunityHost:ACIPort/action=UserExport&ExportFileName=fileName
```

where:

CommunityHost	is the IP address or name of the Community machine.
ACIPort	is the Community ACI port (the value of `Port` in the `[Server]` section of the Community configuration file).
fileName	is the name of the XML file to export Community users, roles, agents and profiles to. You must specify the path to the file if the XML file is not stored in the same directory as Community.

For example:

```
http://12.3.4.56:20000/action=UserExport&ExportFileName=MyFile.xml
```

This action uses port `20000` to export Community users, roles, agents, and profiles to the file `MyFile.xml`.

Import Users, Roles, Agents, and Profiles

You can import IDOL Community component users, roles, agents, and profiles from a specified XML file (into which you previously exported them from Community by using the `UserExport` action). You can use this process, for example, to transfer Community to a different platform.

```
http://
CommunityHost:ACIPort/action=UserImport&ImportFileName=fileName&UserFields=fieldCSV
```

where:

CommunityHost	is the IP address or name of the Community machine.
ACIPort	is the Community ACI port (the value of `Port` in the `[Server]` section of the Community configuration file).
fileName	is the name of the XML file that contains the users, roles, agents, and profiles that you want to import. If the XML file is not stored in the same directory as Community, specify the path to the file as well.
fieldCSV	(optional) allows you to restrict the import of user fields by specifying a Wildcard list of fields to import. If you specify multiple fields, separate them with commas (there must be no space before or after a comma). Community imports only user fields that match a listed field.

For example:

```
http://12.3.4.56:20000/action=UserImport&ImportFileName=MyFile.xml
```

This action uses port `20000` to import Community users, roles, agents, and profiles from the file `MyFile.xml`.

Back up Other IDOL Components

Most IDOL components do not store additional data, and so do not need to be backed up. For example, the IDOL View component does not store any data except for its cache, which it can build up again from a clean installation.

However, Micro Focus recommends that you back up your component configuration files, to ensure that you can restore a particular custom configuration.

In addition, IDOL Connectors store information about the files they have previously retrieved from a repository. You can back up your connectors by using a `BackupServer` action. For more information, refer to the documentation for the appropriate connector.

Chapter 24: Troubleshoot IDOL Server

To troubleshoot IDOL Server, consult the log files. The entries in these files provide details of possible problems or incorrect configurations. They are useful in tracking and fixing, for example, configuration errors.

IDOL Server Log Files

By default IDOL Server creates the following log files:

`agentstore_application.log`	Logs general application errors, warnings, and information relating to the IDOL Server agent index.
`agentstore_index.log`	Logs messages relating to the indexing, deletion, and updating of agents.
`agentstore_query.log`	Logs messages relating to the querying of agents.
`application.log`	Logs general application errors, warnings, and information relating to IDOL Server indexes.
`community_application.log`	Logs general application errors, warnings, and information relating to user settings (for example, preferences and authentication).
`community_term.log`	Logs terms as they are added to profiles and agents.
`community_user.log`	Logs when users and agents are added or deleted (or when this fails).
`category_application.log`	Logs general application errors, warnings, and information relating to the IDOL Category component category index.
`category_category.log`	Logs messages relating to category actions that read or manipulate the categories, including errors, warnings, and progress information.
`category_cluster.log`	Logs messages relating to cluster actions, including errors, warnings, and progress information.
`category_schedule.log`	Logs messages relating to the running of the analysis schedules that are

	specified in the configuration file.
category_ taxonomy.log	Logs messages relating to the `TaxonomyGenerate` action, including errors, warnings, and progress information.
content_ application.log	Logs general application errors, warnings, and information relating to the IDOL Content component data index.
content_ index.log	Logs messages relating to the indexing, deletion, and updating of documents.
content_ query.log	Logs messages relating to query processes.
content_ queryterms.log	Logs the query terms.
index.log	Logs the index actions that IDOL Server receives.
mailer. log	Logs messages relating to the mailer and scheduled mail tasks.
query.log	Logs all the requests that IDOL Server receives.
stats_index.log	Logs IDOL Server statistics.

You can use the IDOL Admin interface to view the Request log and any other log streams that are enabled.

To select a log stream to display

1. In the **Monitor** menu, click **Logs**.

2. Click the relevant tab.

If there is more than one log file available for that log stream, a list displays the options.

When you select a log stream, it is generated in the user interface.

> **NOTE:** IDOL Admin shows only the logs for the component that you are currently connected to.

To update the log stream at any point

*   Click **Refresh**. You can also enable Autorefresh by selecting a refresh interval in the **Refresh every [*number*] seconds** list. If Autorefresh is enabled, only the latest log entries are visible.

> **NOTE:** You can refresh only the current log file for each log stream. If you are viewing a previous log file (the log name is appended with a date), the refresh options are unavailable.

To search within the displayed log stream

*   Type a search term into the box in the top-right corner of the page.

    If the term appears in the log stream, the search box turns green and the instances in the text are highlighted. The log stream automatically scrolls to the first instance. If there are no results for the term, the search box turns red.

Warning or error entries in the log streams are written in orange text. The number of warning and error messages in a log stream is displayed beside its tab name. Hover over the number to open either a Warnings in [*logstream*] window or an Errors in [*logstream*] window, containing links to the relevant entries in the log stream.

Customize Logging

You can customize logging by setting up your own *log streams*. Each log stream creates a separate log file in which specific log message types (for example, action, index, application, or import) are logged.

To set up log streams

1. Open the IDOL Server configuration file in a text editor.

2. Find the `[Logging]` section. If the configuration file does not contain a `[Logging]` section, add one.

3. In the `[Logging]` section, create a list of the log streams that you want to set up, in the format *N=LogStreamName*. List the log streams in consecutive order, starting from 0 (zero). For example:

   ```
   [Logging]
   LogLevel=FULL
   LogDirectory=logs
   0=ApplicationLogStream
   1=ActionLogStream
   ```

   You can also use the `[Logging]` section to configure any default values for logging configuration parameters, such as `LogLevel`. For more information, see the *IDOL Server Reference*.

4. Create a new section for each of the log streams. Each section must have the same name as the log stream. For example:

   ```
   [ApplicationLogStream]
   [ActionLogStream]
   ```

5. Specify the settings for each log stream in the appropriate section. You can specify the type of logging to perform (for example, full logging), whether to display log messages on the console, the maximum size of log files, and so on. For example:

   ```
   [ApplicationLogStream]
   LogTypeCSVs=application
   LogFile=application.log
   LogHistorySize=50
   LogTime=True
   LogEcho=False
   LogMaxSizeKBs=1024

   [ActionLogStream]
   LogTypeCSVs=action
   LogFile=logs/action.log
   LogHistorySize=50
   ```

```
LogTime=True
LogEcho=False
LogMaxSizeKBs=1024
```

6. Save and close the configuration file. Restart the service for your changes to take effect.

Create a Diagnostics Package

You can automatically generate a .ZIP package that includes files that are useful for troubleshooting. When you need to report a problem to Micro Focus Big Data customer support, create a diagnostics package and send it to help them identify and solve your problem.

You can create a diagnostics package either by sending an action, or by using IDOL Admin.

To create a diagnostics package by sending an action

- Send the `Diagnostic` action to the service port of the component that you want to create the diagnostics package for.

  ○ Set the `File` parameter to the file name and path for the .ZIP file to create.

  ○ (Optional) Set the `Logs` parameter to one of the following options to determine the type of logs to include in the diagnostics package:

`Current`	Include only current logs.
`Full`	Include all logs, including archives.
`None`	Do not include logs.

  Micro Focus recommends that you include logs in the diagnostics package if you send it to Micro Focus Big Data customer support.

For example:

```
http://localhost:9012/action=Diagnostic&File=C:\Diagnostics\Content_
10.2.zip&Logs=Current
```

This action creates a diagnostic .ZIP file for the IDOL Content component on the local machine with service port `9012`.

To create a diagnostics package by using IDOL Admin

1. In the IDOL Admin interface, go to the **Diagnostics** tab on the Console page.

2. In the **File path on IDOL server** box, type a file path on your server to save the diagnostics bundle to. Clicking the icon on the right generates a unique file name using the format *server-install-directory/diagnostics-timestamp.zip*.

3. Select the level of log information to include in the diagnostics bundle:

Current	Include only current logs.
Full	Include all logs, including archives.
None	Do not include logs.

4. Click **Create Diagnostics Bundle** to export the diagnostics ZIP file to the specified location.

   A confirmation message appears.

IDOL Statistics Server

The IDOL Statistics server monitors interactions between one or more IDOL databases and an end user. Interactions can occur in two ways: a user can send actions directly to an IDOL Server through a Web browser; alternatively, a user can interact with IDOL through a front-end application such as IDOL Data Admin. The Statistics server monitors IDOL log files for queries or actions that users send to the database, then uses that data to report statistics.

For more information about Statistics Server, refer to the *Query Manipulation Server Administration Guide*.

Part VI: Appendixes

This section includes the following appendixes:

- IDOL Server Configuration File
- Password Encryption
- Languages and Language Files
- Manually Create IDX Files
- Category XML Format
- GetStatus Action Response
- Error Codes

Appendix A: IDOL Server Configuration File

This section describes the IDOL Server configuration file.

IDOL Server Configuration File

In a unified IDOL Server, the IDOL Server configuration file contains the parameters that determine how IDOL Server operates. You can modify the configuration parameters to customize IDOL Server. You can find this file in the following directory:

InstallDir\idolserver.cfg

where *InstallDir* is the installation directory for your IDOL Server.

You can validate the configuration file, or search for text within the configuration file, by using the Config page in the **Monitor** section of the IDOL Admin user interface. However, you cannot edit the configuration file by using IDOL Admin. For more information, refer to the *IDOL Admin User Guide*.

In an IDOL setup where you use stand-alone components, each component uses its own configuration file. For more information, see Edit IDOL Configuration Files, on page 32 and the *IDOL Getting Started Guide*.

The following sections give more information about editing configuration files, and provides details about the sections in the unified IDOL Server configuration file.

Related Topics

- Edit IDOL Configuration Files, on page 32

Modify Configuration Parameter Values

You modify IDOL Server configuration parameters by directly editing the parameters in the configuration file. When setting configuration parameter values, you must use UTF-8

> **NOTE:** You must stop and restart IDOL Server for new configuration settings to take effect.

This section describes how to enter parameter values in the configuration file.

Enter Boolean Values

The following settings for Boolean parameters are interchangeable:

```
TRUE = true = ON = on = Y = y = 1

FALSE = false = OFF = off = N = n = 0
```

Enter String Values

To enter a comma-separated list of strings when one of the strings contains a comma, you can indicate the start and the end of the string with quotation marks, for example:

ParameterName=**cat,dog,bird,"wing,beak",turtle**

Alternatively, you can escape the comma with a backslash:

ParameterName=**cat,dog,bird,wing\,beak,turtle**

If any string in a comma-separated list contains quotation marks, you must put this string into quotation marks and escape each quotation mark in the string by inserting a backslash before it. For example:

ParameterName=**"<font face=\"arial\" size=\"+1\"><b>","<p>"**

Here, quotation marks indicate the beginning and end of the string. All quotation marks that are contained in the string are escaped.

Include an External Configuration File

You can share configuration sections or parameters between ACI server configuration files. The following sections describe different ways to include content from an external configuration file.

You can include a configuration file in its entirety, specified configuration sections, or a single parameter.

When you include content from an external configuration file, the `GetConfig` and `ValidateConfig` actions operate on the combined configuration, after any external content is merged in.

> **CAUTION:** You cannot use shared configuration files if you use any actions or index actions that modify the configuration file (for example, the `DRECREATEDBASE` index action writes the new database configuration to the file). If IDOL Server updates the configuration file in this way, it overwrites the link to the shared configuration file with the imported content. This behavior prevents possible conflicts if the configuration change updates any shared configuration.

In the procedures in the following sections, you can specify external configuration file locations by using absolute paths, relative paths, and network locations. For example:

```
../sharedconfig.cfg
K:\sharedconfig\sharedsettings.cfg
\\example.com\shared\idol.cfg
file://example.com/shared/idol.cfg
```

Relative paths are relative to the primary configuration file.

> **NOTE:** You can use nested inclusions, for example, you can refer to a shared configuration file that references a third file. However, the external configuration files must not refer back to your original configuration file. These circular references result in an error, and IDOL Server does not start.

> Similarly, you cannot use any of these methods to refer to a different section in your primary configuration file.

Include the Whole External Configuration File

This method allows you to import the whole external configuration file at a specified point in your configuration file.

To include the whole external configuration file

1. Open your configuration file in a text editor.

2. Find the place in the configuration file where you want to add the external configuration file.

3. On a new line, type a left angle bracket (<), followed by the path to and name of the external configuration file, in quotation marks (""). You can use relative paths and network locations. For example:

   ```
   < "K:\sharedconfig\sharedsettings.cfg"
   ```

4. Save and close the configuration file.

Include Sections of an External Configuration File

This method allows you to import one or more configuration sections (including the section headings) from an external configuration file at a specified point in your configuration file. You can include a whole configuration section in this way, but the configuration section name in the external file must exactly match what you want to use in your file. If you want to use a configuration section from the external file with a different name, see Merge a Section from an External Configuration File, on the next page.

To include sections of an external configuration file

1. Open your configuration file in a text editor.

2. Find the place in the configuration file where you want to add the external configuration file section.

3. On a new line, type a left angle bracket (<), followed by the path of the external configuration file, in quotation marks (""). You can use relative paths and network locations. After the configuration file path, add the configuration section name that you want to include. For example:

   ```
   < "K:\sharedconfig\extrasettings.cfg" [License]
   ```

   > **NOTE:** You cannot include a section that already exists in your configuration file.

4. Save and close the configuration file.

Include Parameters from an External Configuration File

This method allows you to import one or more parameters from an external configuration file at a specified point in your configuration file. You can import a single parameter or use wildcards to specify

multiple parameters. The parameter values in the external file must match what you want to use in your file. This method does not import the section heading, such as `[License]` in the following examples.

To include parameters from an external configuration file

1. Open your configuration file in a text editor.

2. Find the place in the configuration file where you want to add the parameters from the external configuration file.

3. On a new line, type a left angle bracket (`<`), followed by the path of the external configuration file, in quotation marks (`""`). You can use relative paths and network locations. After the configuration file path, add the name of the section that contains the parameter, followed by the parameter name. For example:

   ```
   < "license.cfg" [License] LicenseServerHost
   ```

   To specify a default value for the parameter, in case it does not exist in the external configuration file, specify the configuration section, parameter name, and then an equals sign (`=`) followed by the default value. For example:

   ```
   < "license.cfg" [License] LicenseServerHost=localhost
   ```

   You can use wildcards to import multiple parameters, but this method does not support default values. The `*` wildcard matches zero or more characters. The `?` wildcard matches any single character. Use the pipe character | as a separator between wildcard strings. For example:

   ```
   < "license.cfg" [License] LicenseServer*
   ```

4. Save and close the configuration file.

Merge a Section from an External Configuration File

This method allows you to include a configuration section from an external configuration file as part of your IDOL Server configuration file. For example, you might want to specify a standard SSL configuration section in an external file and share it between several servers. You can use this method if the configuration section that you want to import has a different name to the one you want to use.

To merge a configuration section from an external configuration file

1. Open your configuration file in a text editor.

2. Find or create the configuration section that you want to include from an external file. For example:

   ```
   [SSLOptions1]
   ```

3. After the configuration section name, type a left angle bracket (`<`), followed by the path to and name of the external configuration file, in quotation marks (`""`). You can use relative paths and network locations. For example:

   ```
   [SSLOptions1] < "../sharedconfig/ssloptions.cfg"
   ```

If the configuration section name in the external configuration file does not match the name that you want to use in your configuration file, specify the section to import after the configuration file name. For example:

```
[SSLOptions1] < "../sharedconfig/ssloptions.cfg" [SharedSSLOptions]
```

In this example, IDOL Server uses the values in the `[SharedSSLOptions]` section of the external configuration file as the values in the `[SSLOptions1]` section of the IDOL Server configuration file.

> **NOTE:** You can include additional configuration parameters in the section in your file. If these parameters also exist in the imported external configuration file, IDOL Server uses the values in the local configuration file. For example:
>
> ```
> [SSLOptions1] < "ssloptions.cfg" [SharedSSLOptions]
> SSLCACertificatesPath=C:\IDOL\HTTPConnector\CACERTS\
> ```

4. Save and close the configuration file.

Configuration File Sections

The IDOL Server configuration file contains several sections, each of which represent a specific area that you can configure by setting appropriate configuration parameters. For details on all available configuration parameters, refer to the *IDOL Server Reference*.

The configuration file sections for each configuration parameter are listed in the *IDOL Server Reference* under **Configuration Sections**.

In a unified IDOL Server configuration, all components read settings from the same configuration. In a stand-alone component, you create a separate configuration file for each component. In most cases, the names of the sections are the same. Some sections are available for more than one component.

For information about the configuration sections and settings available for each individual component, refer to the Reference for that component (for example, the *IDOL Content component Reference*.

Related Topics

- Display Online Help, on page 30

The configuration file can contain the following sections:

`[ACIEncryption]`	`[IndexCache]`	`[Security]`
`[Agent]`	`[IndexQueue]`	`[Server]`
`[AgentDRE]`	`[IndexServer]`	`[Service]`
`[AnalysisSchedules]`	`[LanguageTypes]`	`[SSLOptionN]`
`[Category]`	`[License]`	`[Summary]`
`[CatDRE]`	`[Logging]`	`[Synonym]`
`[Cluster]`	`[MemoryCache]`	`[Taxonomy]`

[Community]	[*MyProperty*]	[TermCache]
[DRE]	[Paths]	[User]
[DataDRE]	[Profile]	[UserCustom]
[Databases]	[ProfileNamedAreas]	[UserSecurity]
[DistributionSettings]	[Role]	[UserSecurityFields]
[DocumentTracking]	[Schedule]	[Viewing]
[FieldProcessing]	[SectionBreaking]	

NOTE: Some of these sections might not be present in the IDOL Server configuration file immediately after installation. The sections that you require depend on the operations that you need your IDOL Server to carry out.

[ACIEncryption] Section

You can encrypt communications between ACI servers and applications that use the IDOL ACI API by configuring an [ACIEncryption] section in each of the application configuration files. For example:

```
[ACIEncryption]
CommsAllowUnencrypted=False
CommsEncryptionType=GSS
ServiceName=kerberos
```

[Agent] Section

The [Agent] section determines how agents operate. For example:

```
[Agent]
DynamicAgentFields=True
DreCombine=Simple
DreSentences=3
DreCharacters=300
DreSummary=Context
DontCopyAgentFields=EmailAddress
AgentIndexFieldCSVs=DRELanguageType
```

Related Topics

- Agents, on page 151

[AgentDRE] Section

The [AgentDRE] section contains details of the machine that hosts the IDOL agent index. The IDOL Community component stores agents and profiles in its agent index.

In a unified IDOL Server, this section is not required. By default, the IDOL Community component uses the IDOL Agentstore component on the same machine. You can configure `[AgentDRE]` if you want to use a different Agentstore.

This section is required in a stand-alone IDOL Community component configuration.

```
[AgentDRE]
ACIPort=9002
Host=1.23.45.6
Timeout=5000
```

[AnalysisSchedules] Section

The `[AnalysisSchedules]` section summarizes the number of classification schedules to run. You must create a subsection to specify details for each of these schedules. You can schedule the following actions:

- `ClusterSnapshot`

- `ClusterCluster`

- `ClusterSGDataGen`

- `TaxonomyGenerate`

For example:

```
[AnalysisSchedules]
RunMissedSchedules=True

[AnalysisSchedule0]
ScheduleStartTime=12:00
ScheduleInterval=1day
ScheduleCycles=-1
ScheduleAction=ClusterSnapshot
TargetJobname=myjob

[AnalysisSchedule1]
ScheduleStartTime=12:15
ScheduleInterval=1day
ScheduleCycles=-1
ScheduleAction=ClusterCluster
SourceJobName=myjob
TargetJobName=myjob_clusters
DoMapping=True

[AnalysisSchedule2]
ScheduleStartTime=12:15
ScheduleInterval=1day
ScheduleCycles=-1
ScheduleAction=ClusterCluster
SourceJobName=myjob
TargetJobName=myjob_clusters_new
```

```
WhatsNew=True
Interval=86400
```

Related Topics

- Set up Schedules, on page 205

[AuthorizationRoles] Section

The `[AuthorizationRoles]` defines roles that enable a particular set of actions for particular clients, SSL identities, and GSS principals. You must create a subsection for each authorization role that you define in the `[AuthorizationRoles]` configuration section.

For example:

```
[AuthorizationRoles]
0=AdminRole
1=UserRole

[AdminRole]
StandardRoles=Admin,Index,ServiceControl
Clients=localhost
SSLIdentities=admin.example.com

[UserRole]
StandardRoles=Query,ServiceStatus
SSLIdentities=admin.example.com,userserver.example.com
```

[CatDRE] Section

The `[CatDRE]` section contains details of the machine that hosts the IDOL category index. The IDOL Category component stores categories in its category index.

In a unified IDOL Server, this section is not required. By default, the IDOL Category component uses the IDOL Agentstore component on the same machine. You can configure `[CatDRE]` if you want to use a different Agentstore.

This section is required in a stand-alone IDOL Category component configuration.

```
[CatDRE]
ACIPort=9002
Host=1.23.45.6
```

[Category] Section

The `[Category]` section contains details for the categories that IDOL Server creates in its category index. For example:

```
[Category]
AuthorizeWithoutUser=True
```

Related Topics

-

[Cluster] Section

The [Cluster] section contains the details for clustering. For example:

```
[Cluster]
ResultExpiryDays=30
SnapshotExpiryDays=30
SGExpiryDays=30
DownloadDocAction=DREContents
TitleFromSummary=True
SummaryField=autn:summary
```

Related Topics

-

[Community] Section

The [Community] section determines how community queries operate. For example:

```
[Community]
DreMinScore=20
DreWeighFieldText=False
ExpandQuery=False
ExpandQueryLog=False
ExpandQueryMinScore=60
ExpandQueryMaxResults=30
ExpandQueryMaxScore=80
```

[DAHEngines] Section

Use this section when you are using a unified IDOL Server configuration with the Distributed Action Handler and Distributed Index Handler components for a distributed setup. In this case, you configure DAH and DIH in the unified IDOL Server configuration file, and distribute actions to child IDOL Content components.

The [DAHEngines] section sets the total number of child servers that DAH distributes to. You must also create a [DAHEngine*N*] subsection for each of the child servers, in which you specify the child server details.

Use this option to configure the child servers when you want to configure different child servers for the DAH and DIH. For example, you might be using a chained DIH and DAH architecture, such that the top-level DAH distributes actions to child DAH components, and the top-level DIH distributes actions to child DIH components.

In this configuration, you specify the DIH child servers in the [DIHEngines] section (see).

If your DAH and DIH distribute actions to the same child servers (for example, IDOL Content components), you can use `[DistributionIDOLServers]` and `[IDOLServerN]` to specify the child servers. See .

```
[DAHEngines]
Number=2
```

[DAHEngine*N*] Section

Use this section when you are using a unified IDOL Server configuration with the Distributed Action Handler and Distributed Index Handler components for a distributed setup. In this case, you configure DAH and DIH in the unified IDOL Server configuration file, and distribute actions to child IDOL Content components.

Each `[DAHEngineN]` section define the host and port of a child server that the DAH communicates with.

Use this option to configure the child servers when you want to configure different child servers for the DAH and DIH. For example, you might be using a chained DIH and DAH architecture, such that the top-level DAH distributes actions to child DAH components, and the top-level DIH distributes actions to child DIH components.

You must number child server sections in consecutive order, starting from 0 (zero).

For example:

```
[DAHEngines]
Number=2

[DAHEngine0]
Host=localhost
Port=9010

[DAHEngine1]
Host=12.3.4.56
Port=9010
```

[Databases] Section

The `[Databases]` section lists the databases in which IDOL Server stores its data, and contains a subsection for each of the databases, where you specify settings that apply only to this database. If you index documents in multiple languages, you do not need to create a database for each language. For example:

```
[Databases]
NumDBs=2

[Database0]
Name=News
```

```
[Database1]
Name=Archive
```

Related Topics

-

[DataDRE] Section

The [DataDRE] section contains details of the machine that hosts the IDOL Content component data index that the IDOL Community component and IDOL Category component use for operations that require the data (such as categorization or alerting).

In a unified IDOL Server, this section is not required. By default, Category and Community both use the IDOL Content component on the same machine (or the DIH and DAH if you use a distributed unified configuration). You can configure [DataDRE] if you want to use a different IDOL Content component.

This section is required in the configuration for a stand-alone IDOL Community component or IDOL Category component.

```
[DataDRE]
Host=7.89.01.2
ACIPort=6002
Timeout=5000
```

[DIHEngines] Section

Use this section when you are using a unified IDOL Server configuration with the Distributed Action Handler and Distributed Index Handler components for a distributed setup. In this case, you configure DAH and DIH in the unified IDOL Server configuration file, and distribute actions to child IDOL Content components.

The [DIHEngines] section sets the total number of child servers that DIH distributes to. You must also create a [DIHEngine*N*] subsection for each of the child servers, in which you specify the child server details.

Use this option to configure the child servers when you want to configure different child servers for the DAH and DIH. For example, you might be using a chained DIH and DAH architecture, such that the top-level DAH distributes actions to child DAH components, and the top-level DIH distributes actions to child DIH components.

In this configuration, you specify the DAH child servers in the [DAHEngines] section (see).

If your DAH and DIH distribute actions to the same child servers (for example, IDOL Content components), you can use [DistributionIDOLServers] and [IDOLServer*N*] to specify the child servers. See .

For example:

```
[DIHEngines]
Number=2
```

[DIHEngine*N*] Section

Use this section when you are using a unified IDOL Server configuration with the Distributed Action Handler and Distributed Index Handler components for a distributed setup. In this case, you configure DAH and DIH in the unified IDOL Server configuration file, and distribute actions to child IDOL Content components.

Each [DIHEngine*N*] section define the host and port of a child server that the DIH communicates with.

Use this option to configure the child servers when you want to configure different child servers for the DAH and DIH. For example, you might be using a chained DIH and DAH architecture, such that the top-level DAH distributes actions to child DAH components, and the top-level DIH distributes actions to child DIH components.

You must number child server sections in consecutive order, starting from 0 (zero).

For example:

```
[DIHEngines]
Number=2

[DIHEngine0]
Host=dih1.company.com
Port=16100

[DIHEngine1]
Host=12.3.4.56
Port=16200
```

[DistributionIDOLServers] Section

Use this section when you are using a unified IDOL Server configuration with the Distributed Action Handler and Distributed Index Handler components for a distributed setup. In this case, you configure DAH and DIH in the unified IDOL Server configuration file, and distribute actions to child IDOL Content components.

The [DistributionIDOLServers] section sets the total number of child IDOL Content components that DIH and DAH distribute to. You must also create an [IDOLServer*N*] subsection for each of the child IDOL Content components, in which you can specify the child server details.

When you use [DistributionIDOLServers] and [IDOLServer*N*], the child servers are the same for DAH and DIH and must handle both ACI and index actions.

You can also create separate child server configuration sections for the DAH and DIH by using [DAHEngines] and [DIHEngines] (see [DAHEngines] Section, on page 483 and [DIHEngines] Section, on the previous page).

For more information refer to the *Distributed Action Handler Administration Guide* and the *Distributed Index Handler Administration Guide.*

```
[DistributionIDOLServers]
Number=2
```

[DistributionSettings] Section

Use this section when you are using a unified IDOL Server configuration with the Distributed Action Handler and Distributed Index Handler components for a distributed setup. In this case, you configure DAH and DIH in the unified IDOL Server configuration file, and distribute actions to child IDOL Content components.

The `[DistributionSettings]` section contains the options that normally appear in the `[Server]` section of the `DAH.cfg` and `DIH.cfg` files in the stand-alone component configuration.

For more information, refer to the *Distributed Action Handler Administration Guide* and the *Distributed Index Handler Administration Guide*.

For example, a unified configuration with the DIH:

```
[DistributionSettings]
Port=16000
DIHPort=16001
MirrorMode=True
```

[DocumentTracking] Section

The `[DocumentTracking]` section contains parameters that enable the tracking of documents through the import and indexing process. For example:

```
[DocumentTracking]
Backend=IDOL
DatabaseName=DocTracking
TargetHost=IDOL1
TargetPort=9001
```

[DRE] Section

The `[DRE]` section allows you to list `Query`, `TermGetBest`, and `TermGetInfo` action parameters to enable for agent and profile queries. After you enable the parameters in the configuration file, you can set them for the DRE*QueryParameter* in the `[Agent]` and `[Profile]` section of the configuration file, or for the DRE*QueryParameter* parameter of `AgentGetResults` and `Community` actions. For example:

```
[DRE]
AdditionalDREQueryParameters=Characters,MaxDate
AdditionalDRETermGetBestParameters=Weights
AdditionalDRETermGetInfoParameters=OnlyExisting
```

Related Topics

- Agents, on page 151

- Profiles, on page 209

[FieldProcessing] Section

The [FieldProcesssing] section lists the processes to apply to fields, and contains a subsection for each of the processes, in which you define the process. For example:

```
[FieldProcessing]
0=SetIndexFields
1=SetIndexAndWeightHigher
2=SetSectionBreakFields
3=SetDateFields
4=SetDatabaseFields
5=SetReferenceFields
6=SetTitleFields
7=SetHighlightFields
8=SetSourceFields
9=DetectNT_V4Security
10=DetectNotes_V4Security
11=DetectNetware_V4Security
12=DetectExchange_V4Security
13=DetectDocumentum_V4Security
14=HideAutonomyMetaDataField
15=SetNumericFields
16=SetFieldCheckFields

[SetIndexFields]
Property=IndexFields
PropertyFieldCSVs=*/DRECONTENT,*/DRETITLE

[SetIndexAndWeightHigher]
Property=IndexWeightFields
PropertyFieldCSVs=*/SUMMARIES

[SetSectionBreakFields]
Property=SectionFields
PropertyFieldCSVs=*/DRESECTION

[SetDateFields]
Property=DateFields
PropertyFieldCSVs=*/DREDATE,*/DATE

[SetDatabaseFields]
Property=DatabaseFields
PropertyFieldCSVs=*/DREDBNAME,*/DATABASE

[SetReferenceFields]
Property=ReferenceFields
PropertyFieldCSVs=*/DREREFERENCE,*/REFERENCE
```

```
[SetTitleFields]
Property=TitleFields
PropertyFieldCSVs=*/DRETITLE,*/TITLE

[SetHighlightFields]
Property=HighlightFields
PropertyFieldCSVs=*/DRETITLE,*/DRECONTENT

[SetSourceFields]
Property=SourceFields
PropertyFieldCSVs=*/DRETITLE,*/DRECONTENT

[DetectNT_V4Security]
Property=SecurityNT_V4
PropertyFieldCSVs=*/SECURITYTYPE
PropertyMatch=nt

[DetectNotes_V4Security]
Property=SecurityNotes_V4
PropertyFieldCSVs=*/SECURITYTYPE
PropertyMatch=*notes_v4

[DetectNetware_V4Security]
Property=SecurityNetware_V4
PropertyFieldCSVs=*/SECURITYTYPE
PropertyMatch=*netware_v4

[DetectExchange_V4Security]
Property=SecurityExchange_V4
PropertyFieldCSVs=*/SECURITYTYPE
PropertyMatch=*exchange_v4

[DetectDocumentum_V4Security]
Property=SecurityDocumentum_V4
PropertyFieldCSVs=*/SECURITYTYPE
PropertyMatch=*documentum

[HideAutonomyMetadataField]
Property=HideMetaDataFields
PropertyFieldCSVs=*/AUTONOMYMETADATA

[SetNumericFields]
Property=NumericFields
PropertyFieldCSVs=*/NONEXISTENT

[SetFieldCheckFields]
Property=FieldCheckFields
PropertyFieldCSVs=*/NONEXISTENT
```

Related Topics

- Fields, on page 81

[FlushLock] Section

The [FlushLock] configuration section contains settings that control disk flushing and locking.

Flushing is the process of writing cached data to disk during indexing. It occurs periodically, and it is resource-intensive. It is important to prevent two flushes from occurring simultaneously.

If a lock file exists, a server writes information to it to notify other servers that it is flushing, and other servers wait until it is finished before writing to the lock file and performing their flushes.

You can configure a lock file by using the FlushLockFile configuration parameter in the [Server] section. However, Micro Focus does not recommend that you use a FlushLockFile if you are using networked storage.

For networked storage, you can configure the [FlushLock] and [*MyLockServer*] sections with the details of Redis servers. In this case, the Content component attempts to contact all the configured Redis servers to obtain a lock, and flushes only if it can obtain a lock from more than half of the Redis servers.

For failover purposes, Micro Focus recommends that you configure at least three Redis servers for locking. In this case, if one of the servers is unavailable, the Content component can still obtain more than half of the locks.

> **NOTE:** Your lock servers must use Redis version 2.8 or later.

```
[FlushLock]
0=LockServer1
1=LockServer2
2=LockServer3

[LockServer1]
Host=12.34.56.78
Port=6379

[LockServer2]
Host=12.34.56.89
Port=6389

[LockServer3]
Host=12.34.56.12
Port=6389
```

Related Topics

- [MyLockServer] Sections, on page 495

[IDOLServer*N*] Section

Use this section when you are using a unified IDOL Server configuration with the Distributed Action Handler and Distributed Index Handler components for a distributed setup. In this case, you configure DAH and DIH in the unified IDOL Server configuration file, and distribute actions to child IDOL Content components.

Each [`IDOLServerN`] section define the host and port of a child IDOL Content component that the DAH and DIH communicate with.

Use this section to define the child server configurations when you use [`DistributionIDOLServers`] and use the same child severs for DAH and DIH.

You must number child server sections in consecutive order, starting from 0 (zero).

For example:

```
[DistributionIDOLServers]
Number=2

[IDOLServer0]
Host=localhost
Port=9010

[IDOLServer1]
Host=1.23.45.6
Port=9010
```

[IndexCache] Section

The [`IndexCache`] section contains parameters that determine how much memory IDOL Server uses to cache data for indexing. For example:

```
[IndexCache]
IndexCacheMaxSize=102400
```

[IndexNotify] Section

The [`IndexNotify`] section contains parameters that control and enable the automatic generation of index job information for a specified host. For example:

```
[IndexNotify]
Host=10.1.1.10
ACIPort=9992
BatchSize=1
BatchTimeout=10000
ConnectRetries=1
ConnectTimeout=5000
```

[IndexQueue] Section

The [IndexQueue] section contains parameters that control the index queue. For example:

```
[IndexQueue]
IndexQueueInitialSize=30000
IndexQueueMaxHistory=4000
IndexQueueMaxPendingItems=100
```

[IndexServer] Section

The [IndexServer] section contains settings for the IDOL Server index port. Use this section to set Secure Socket Layer (SSL) communications for the index port. For example:

```
[IndexServer]
SSLConfig=SSLOption1
```

Related Topics

-

[LanguageTypes] Section

The [LanguageTypes] section lists the language types that you want to use. It contains a section for each language listed, in which you configure the parameters that determine how to handle each language. For example:

```
[LanguageTypes]
DefaultLanguageType=englishUTF8
DefaultEncoding=UTF8
LanguageDirectory=C:\IDOLServer\IDOL\langfiles
0=afrikaans
1=albanian
2=arabic
3=armenian
4=azeri
5=basque
6=belorussian
7=bengali
8=bosnian

[afrikaans]
Encodings=UTF8:afrikaansUTF8
IndexNumbers=1

[albanian]
EncodingsUTF8:albanianUTF8
IndexNumbers=1
```

```
[arabic]
Encodings=ARABIC_ISO:arabicARABIC_ISO,ARABIC:arabicARABIC,UTF8:arabicUTF8
IndexNumbers=1

[armenian]
Encodings=UTF8:armenianUTF8
IndexNumbers=1

[azeri]
Encodings=UTF8:azeriUTF8
IndexNumbers=1

[basque]
Encodings=UTF8:basqueUTF8
Stoplist=basque.dat
IndexNumbers=1

[belorussian]
Encodings=CYRILLIC:belorussianCYRILLIC,UTF8:belorussianUTF8
IndexNumbers=1

[bengali]
Encodings=UTF8:bengaliUTF8
IndexNumbers=1

[bosnian]
Encodings=UTF8:bosnianUTF8
IndexNumbers=1
```

Related Topics

[License] Section

The `[License]` section contains licensing details, which you must not change. For example:

```
[License]
LicenseServerHost=127.0.0.1
LicenseServerACIPort=20000
LicenseServerTimeout=600000
LicenseServerRetries=1
```

[Logging] Section

The `[Logging]` section lists the logging streams to set up to create separate log files for different log message types (query, index, and application). It also contains a subsection for each of the listed logging streams, in which you can configure the parameters that determine how each stream is logged. For example:

```
[Logging]
LogArchiveDirectory=C:\idol\IDOLserver\IDOL\logs\archive
LogDirectory=C:\idol\IDOLserver\IDOL\logs
LogTime=True
LogEcho=TRUE
LogLevel=TRUE
LogExpireAction=compress
LogOldAction=move
LogMaxSizeKbs=10240
0=ApplicationLogStream
1=QueryLogStream
2=IndexLogStream
3=QueryTermsLogStream
4=UserLogStream
5=CategoryLogStream
6=ClusterLogStream
7=TaxonomyLogStream
8=ScheduleLogStream
9=CommunityTermLogStream

[ApplicationLogStream]
LogFile=application.log
LogTypeCSVs=application

[QueryLogStream]
LogFile=query.log
LogTypeCSVs=query

[IndexLogStream]
LogFile=index.log
LogTypeCSVs=index

[QueryTermsLogStream]
LogFile=queryterms.log
LogTypeCSVs=queryterms

[UserLogStream]
LogFile=user.log
LogTypeCSVs=user

[CategoryLogStream]
```

```
LogFile=category.log
LogTypeCSVs=category

[ClusterLogStream]
LogFile=cluster.log
LogTypeCSVs=cluster

[TaxonomyLogStream]
LogFile=taxonomy.log
LogTypeCSVs=taxonomy

[ScheduleLogStream]
LogFile=schedule.log
LogTypeCSVs=schedule

[CommunityTermLogStream]
LogFile=term.log
LogTypeCSVs=term
```

NOTE: All queries are truncated to 4,000 characters in query logs.

Related Topics

- Troubleshoot IDOL Server, on page 467

[MemoryCache] Section

The [MemoryCache] section contains parameters that control caching of specified fields to memory. For example:

```
[MemoryCache]
EvictWhenFull=True
MaxSize=1073741
```

[*MyLockServer*] Sections

The [*MyLockServer*] configuration sections contain details of your Redis servers to use for flushing and locking.

Flushing is the process of writing cached data to disk during indexing. It occurs periodically, and it is resource-intensive. It is important to prevent two flushes from occurring simultaneously.

If a lock file exists, a server writes information to it to notify other servers that it is flushing, and other servers wait until it is finished before writing to the lock file and performing their flushes.

You can configure a lock file by using the FlushLockFile configuration parameter in the [Server] section. However, Micro Focus does not recommend that you use a FlushLockFile if you are using networked storage.

For networked storage, you can configure the [FlushLock] and [*MyLockServer*] sections with the details of Redis servers. In this case, the Content component attempts to contact all the configured Redis servers to obtain a lock, and flushes only if it can obtain a lock from more than half of the Redis servers.

For failover purposes, Micro Focus recommends that you configure at least three Redis servers for locking. In this case, if one of the servers is unavailable, the Content component can still obtain more than half of the locks.

> **NOTE:** Your lock servers must use Redis version 2.8 or later.

```
[FlushLock]
0=LockServer1
1=LockServer2
2=LockServer3

[LockServer1]
Host=12.34.56.78
Port=6379

[LockServer2]
Host=12.34.56.89
Port=6389

[LockServer3]
Host=12.34.56.12
Port=6389
```

Related Topics

- [FlushLock] Section, on page 490

[*MyProperty*] **Sections**

The [*MyProperty*] sections list the properties that you created for the processes that you listed in the [FieldProcessing] section. You must create a subsection for each property. Within this section, you set configuration parameters that you apply to associated fields. For example:

```
[IndexFields]
Index=True

[IndexWeightFields]
Index=True
Weight=2

[SectionFields]
SectionBreakType=True

[DateFields]
DateType=True
```

```
[DatabaseFields]
DatabaseType=True

[ReferenceFields]
ReferenceType=True
TrimSpaces=True

[TitleFields]
TitleType=True

[HighlightFields]
HighlightType=True

[SourceFields]
SourceType=True

[SecurityNT_V4]
SecurityType=NT_V4

[SecurityNotes_V4]
SecurityType=Notes_V4

[SecurityNetware_V4]
SecurityType=Netware_V4

[SecurityExchange_V4]
SecurityType=Exchange_V4

[SecurityDocumentum_V4]
SecurityType=Documentum_V4

[HideMetaDataFields]
HiddenType=True
ACLType=True

[NumericFields]
NumericType=True

[FieldCheckFields]
FieldCheckType=True
```

Related Topics

- Fields, on page 81

[Paths] Section

The [Paths] section contains parameters that allow you to split the database into multiple partitions, and parameters that indicate the location of files that IDOL Server uses. For example:

```
[Paths]
DyntermPath=./dynterm
NodetablePath=./nodetable
RefIndexPath=./refindex
MainPath=./main
StatusPath=./status
UserPath=./users
Modules=C:\idol\IDOLserver\IDOL\modules
ClusterDirectory=./cluster
TaxonomyDirectory=./taxonomy
CategoryDirectory=./category
ImExDirectory=./imex
TemplateDirectory=./templates
```

Related Topics

-

[Profile] Section

The [Profile] section contains parameters that determine how to match profiles. For example:

```
[Profile]
DreCombine=Simple
DreSentences=3
DreCharacters=300
DrePrint=All
DreSummary=Context
DreMaxQueryTerms=20
```

Related Topics

-

[ProfileNamedAreas] Section

The [ProfileNamedAreas] section determines the names of the areas that contain the profiles that IDOL Server creates when users read or write documents. For example:

```
[ProfileNamedAreas]
0=default
1=authored
```

[Role] Section

The [Role] section contains parameters that determine the default role and which database a role can access. For example:

```
[Role]
DefaultRolename=everyone
AutoSetDatabases=True
DatabasePrivilege=databases
```

Related Topics

- Add Users to IDOL Server, on page 399

[Schedule] Section

The `[Schedule]` section contains parameters that allow you to schedule when to compact IDOL Server and when to expire documents from databases. For example:

```
[Schedule]
Compact=True
Expire=True
CompactTime=00:00
CompactInterval=672
ExpireTime=00:00
ExpireInterval=24
```

Related Topics

- Compact the Data Index at Regular Intervals, on page 441

[SectionBreaking] Section

The `[SectionBreaking]` section contains parameters that determine the size of the sections that IDOL Server divides documents into before it indexes them. For example:

```
[SectionBreaking]
MinFieldLength=80
MaxSectionLength=2000
```

[Security] Section

The `[Security]` section lists the security modules that you are using, and contains a subsection for each of the security modules. Within each subsection, you can specify the settings to apply to each module. For example:

```
[Security]
SecurityInfoKeys=MyAESKeyFile.ky
0=NT_V4
1=Netware_V4
2=Notes_V4
3=Exchange_V4
4=Documentum_V4
```

```
[NT_V4]
SecurityCode=1
Library=C:\idol\IDOLserver\IDOL\modules\mapped_security
Type=AUTONOMY_SECURITY_V4_NT_MAPPED
ReferenceField=*/AUTONOMYMETADATA

[Netware_V4]
SecurityCode=2
Library=C:\idol\IDOLserver\IDOL\modules\mapped_security
Type=AUTONOMY_SECURITY_V4_NETWARE_MAPPED
ReferenceField=*/AUTONOMYMETADATA

[Notes_V4]
SecurityCode=3
Library=C:\idol\IDOLserver\IDOL\modules\mapped_security
Type=AUTONOMY_SECURITY_V4_NOTES_MAPPED
ReferenceField=*/AUTONOMYMETADATA

[Exchange_V4]
SecurityCode=4
Library=C:\idol\IDOLserver\IDOL\modules\mapped_security
Type=AUTONOMY_SECURITY_V4_EXCHANGE_GRPS_MAPPED
ReferenceField=*/AUTONOMYMETADATA

[Documentum_V4]
SecurityCode=5
Library=C:\idol\IDOLserver\IDOL\modules\mapped_security
Type=AUTONOMY_SECURITY_V4_DOCUMENTUM_MAPPED
ReferenceField=*/AUTONOMYMETADATA
```

NOTE: Use the [`FieldProcessing`] and [*MyProperty*] sections to identify fields that determine the security type of documents and the processes to apply to these fields or documents.

NOTE: If you run IDOL Server on a UNIX platform, specify the LD_LIBRARY_PATH to ensure that IDOL Server can find the shared objects that it requires to implement security.

Related Topics

- Set up Security on Documents, on page 387

[Server] Section

The [Server] section contains general parameters. For example:

```
[Server]
IndexPort=20001
Port=20000
Threads=4
MaxInputString=16000
```

```
DelayedSync=True
AutoDetectLanguagesAtIndex=True
XSLTemplates=False
DateFormatCSVs=SHORTMONTH#SD+#SYYYY,DD/MM/YYYY,YYYY/MM/DD,YYYY-MM-DD
KillDuplicates=*/DREREFERENCE
DocumentDelimiterCSVs=*/DOCUMENT

CantHaveFieldCSVs=*/CHECKSUM,*/DREWORDCOUNT,*/DRETYPE,*/IMPORTBODYLEN,*/IMPORTMETAL
EN,*/IMPORTLINKLEN,*/IMPORTTITLELEN,*/IMPORTQUALITY,*/DREPAGE,*/DREFILENAME,*/dredo
ctype
InactiveSchedules=all
```

[Service] Section

The [Service] section contains parameters that determine which machines are permitted to use and control the IDOL Server service. For example:

```
[Service]
ServicePort=40010
```

[SSLOption*N*] Section

The [SSLOption*N*] section contains settings that determine incoming or outgoing SSL connections for IDOL Server. For example:

```
[SSLOption0]
SSLMethod=TLSV1.3
SSLCertificate=host1.crt
SSLPrivateKey=host1.key

[SSLOption1]
SSLMethod=TLSV1.3
SSLCertificate=host2.crt
SSLPrivateKey=host2.key
SSLPrivateKeyPassword=sample1XQ
SSLCheckCommonName=True
```

> **NOTE:** You must create an SSLOption section for each unique value set by the SSLConfig parameter in the [Server] section, or other section.

Related Topics

- Set up an SSL Connection, on page 391

[Summary] Section

The [Summary] section contains parameters that determine how to generate summaries. For example:

```
[Summary]
MinWordsPerSentence=10
```

Related Topics

- Return Summaries with Query Results, on page 326

[Synonym] Section

The [Synonym] section lists the parameters that determine how IDOL Server handles synonym list queries (that is, queries with the Synonym parameter set to **True**). A synonym query returns results which are conceptually similar to the query terms or the synonyms that are available for those terms. For example:

```
[Synonym]
0=PC_Syn

[PC_Syn]
File=myfile.txt
MaxExpandLevel=1
```

> **NOTE:** To send synonym queries to IDOL Server, you must also set up a synonym file and add the Synonym action parameter to your query. See Enable Synonym Lists, on page 289.

Related Topics

- Synonym Search, on page 288

[Taxonomy] Section

The [Taxonomy] section contains parameters that determine how to generate taxonomies when you run a TaxonomyGenerate action. For example:

```
[Taxonomy]
MaxConcepts=100
RelevanceThreshold=20
DistributionThreshold=10
ConceptThreshold=400
MinConceptOccs=15
CompoundRelevance=40
SiblingStrength=20
MinChildren=1
OnlyMatchSubset=0
MaxQNum=5000
```

Related Topics

- Create Taxonomies, on page 168

[TermCache] Section

The `[TermCache]` section contains parameters that determine which query terms to store in memory, and how much memory to allocate for cached query terms.

For example:

```
[TermCache]
TermCachePersistentKB=100000
```

[User] Section

The `[User]` section contains parameters that determine how many agents each user can have, and which fields belong to these agents. For example:

```
[User]
MaxAgents=10
IndexFieldCSVs=DRELanguageType
```

Related Topics

- Add Users to IDOL Server, on page 399

[UserCustom] Section

The `[UserCustom]` section allows you to add custom functionality to IDOL Server. It lists the functionality that you add, and contains a subsection for each item listed. Within each subsection, you can specify the settings that apply to this functionality (for example, the shared library it uses). For example:

```
[UserCustom]
0=Email

[Email]
Library=C:\IDOLserver\IDOL\modules\user_email
FromHost=127.0.0.1
SMTPHost=smtp.company.com
SMTPPort=25
DrePrint=all
XSLTemplate=C:\IDOLserver\IDOL\templates\email.xss
EmailActionXSLTemplate=C:\IDOLserver\IDOL\templates\ondemand.xss
ClassificationServerXSLTemplate=C:\IDOLserver\IDOL\templates\channels.xss
RunMailer=False
Retries=2
TimeoutMS=15000
StartTime=9:00
Interval=1 day
Cycles=-1
FromName=IdolMailer
```

```
DefaultSendEmail=True
DefaultEmailFormat=text/html
DefaultExcludeReadDocuments=True
DefaultAddSetToReadDocuments=True
DefaultSubject=USERNAME's Results
DefaultMimeVersion=1.0
MaxEmailsPerUser=20
From=user@company.com
```

Related Topics

[UserSecurity] Section

The `[UserSecurity]` section lists your security repositories, specifies generic parameters for them, and contains a subsection for each listed security repository. Within each subsection, you specify the parameters that apply to that repository.

You can list up to eight security types. Each security type must have a matching configuration section.

> **NOTE:** If you rename a security type, IDOL Server treats it as a new security type.
>
> If you remove a security type, IDOL Server leaves the corresponding user security fields intact.

> **NOTE:** In a stand-alone IDOL Community component configuration, this section is called `[Security]`.

For example:

```
[UserSecurity]
DefaultSecurityType=0
DocumentSecurity=True
SyncRolesFromGroups=False
SecurityUsernameDefaultToLoginUsername=False
0=Autonomy
1=NT
2=Notes
3=LDAP
4=Documentum
5=Exchange
6=Netware

[Autonomy]
Library=C:\idol\IDOLserver\IDOL\modules\user_autnsecurity
EnableLogging=False
DocumentSecurity=False
SecurityFieldCSVs=none

[NT]
```

```
CaseSensitiveUserNames=False
CaseSensitiveGroupNames=False
Library=C:\idol\IDOLserver\IDOL\modules\user_ntsecurity
EnableLogging=False
DocumentSecurity=True
V4=True
SecurityFieldCSVs=username,domain
Domain=DOMAIN
DocumentSecurityType=NT_V4
```

[UserSecurityFields] Section

The [UserSecurityFields] section lists the security fields. For example:

```
[UserSecurityFields]
0=username
1=password
2=group
3=domain

[Notes]
Library=C:\idol\IDOLserver\IDOL\modules\user_notessecurity
EnableLogging=False
NotesAuthURL=http://notesserver/names.nsf
DocumentSecurity=True
CaseSensitiveUserNames=False
CaseSensitiveGroupNames=False
SecurityFieldCSVs=username
DocumentSecurityType=Notes_V4

[LDAP]
Library=C:\idol\IDOLserver\IDOL\modules\user_ldapsecurity
EnableLogging=False
RDNAttribute=CN
Group=OU=Users,O=Company
LDAPServer=127.0.0.1
LDAPPort=389
FieldCSVs=email,emailaddress,telephone
LDAPAllAttributeValues=True
LDAPAttributeValueSeparatorChar=,
SecurityFieldCSVs=none
DocumentSecurity=False
CaseSensitiveUserNames=False
CaseSensitiveGroupNames=False

[NT]
CaseSensitiveUserNames=False
CaseSensitiveGroupNames=False
Library=C:\idol\IDOLserver\IDOL\modules\user_ntsecurity
```

```
EnableLogging=False
DocumentSecurity=True
V4=True
SecurityFieldCSVs=username,domain
Domain=DOMAIN
DocumentSecurityType=NT_V4

[Documentum]
DocumentSecurity=True
SecurityFieldCSVs=username
DocumentSecurityType=Documentum_V4
CaseSensitiveUserNames=False
CaseSensitiveGroupNames=False

[Exchange]
DocumentSecurity=True
V4=False
SecurityFieldCSVs=username,domain
DocumentSecurityType=Exchange_V4
CaseSensitiveUserNames=False
CaseSensitiveGroupNames=False

[Netware]
DocumentSecurity=True
DocumentSecurityType=Netware_V4
SecurityFieldCSVs=username
CaseSensitiveUserNames=False
CaseSensitiveGroupNames=False
```

[Viewing] Section

The [Viewing] section contains parameters that control the viewing of retrieved document content as formatted HTML. For example:

```
[Viewing]
//whether caching of previous jobs is done case-sensitively
CaseSensitiveURLs=True
//Expire cached jobs older than this
CacheExpirySeconds=86400
//List of local directories containing documents that can be viewed.
ViewLocalDirectoriesCSVs=
//remove script tags
StripScript=True
//rewrite hyperlinks as IDOL actions
OriginalBaseURL=False
```

Related Topics

- View Documents, on page 361

Appendix B: Password Encryption

This appendix describes how to use the Autpassword command-line tool to encrypt passwords to use in configuration files.

Encrypt Passwords

Micro Focus recommends that you encrypt all passwords that you enter into a configuration file.

Create a Key File

A key file is required to use AES encryption. You can use this key file for password encryption, and you can use it in the `SecurityInfoKeys` configuration parameter to encrypt and decrypt IDOL Server security information.

To create a new key file

1. Open a command-line window and change directory to the IDOL Server installation folder.

2. At the command line, type:

   `autpassword -x -tAES -oKeyFile=./MyKeyFile.ky`

   A new key file is created with the name `MyKeyFile.ky`

> **CAUTION:** To keep your passwords secure, you must protect the key file. Set the permissions on the key file so that only authorized users and processes can read it. IDOL Server must be able to read the key file to decrypt passwords, so do not move or rename it.

Encrypt a Password

The following procedure describes how to encrypt a password.

To encrypt a password

1. Open a command-line window and change directory to the IDOL Server installation folder.

2. At the command line, type:

   `autpassword -e -tEncryptionType [-oKeyFile] [-cFILE -sSECTION -pPARAMETER] PasswordString`

   where:

Option	Description
`-t` *`EncryptionType`*	The type of encryption to use: • **Basic** • **AES** - AES256 For example: **-tAES** > **NOTE:** AES is more secure than basic encryption.
`-o`*`KeyFile`*	AES encryption requires a key file. This option specifies the path and file name of a key file. The key file must contain 64 hexadecimal characters. For example: **-oKeyFile=./key.ky** > **NOTE:** The full (absolute) path of the key file is included in the encrypted value, because IDOL Server requires the key to decrypt the password. If you move or rename the key file, this path becomes invalid and you must update the encrypted value.
`-cFILE -sSECTION -pPARAMETER`	(Optional) You can use these options to write the password directly into a configuration file. You must specify all three options. • **-c**. The configuration file in which to write the encrypted password. • **-s**. The name of the section in the configuration file in which to write the password. • **-p**. The name of the parameter in which to write the encrypted password. For example: **-c./Config.cfg -sMyTask -pPassword**
`PasswordString`	The password to encrypt.

For example:

autpassword -e -tBASIC *MyPassword*

autpassword -e -tAES -oKeyFile=./key.ky *MyPassword*

autpassword -e -tAES -oKeyFile=./key.ky -c./Config.cfg -sDefault -pPassword *MyPassword*

The password is returned, or written to the configuration file.

Decrypt a Password

The following procedure describes how to decrypt a password.

To decrypt a password

1. Open a command-line window and change directory to the IDOL Server installation folder.

2. At the command line, type:

   `autpassword -d -tEncryptionType PasswordString`

   where:

Option	Description
`-tEncryptionType`	The type of encryption: • **Basic** • **AES** - AES256 For example: **-tAES**
`PasswordString`	The password to decrypt.

   For example:

   `autpassword -d -tBASIC 9t3M3t7awt/J8A`

   `autpassword -d -tAES PasswordString`

   The password is returned in plain text.

Appendix C: Decrypt Security Info Strings

This appendix describes how to decrypt the user `SecurityInfo` token. This method is useful mainly for troubleshoot, for example if you need to determine why a user can or cannot access a particular document.

> **TIP:** In general, if you need to decrypt the security info token, Micro Focus recommends that you use the `UserDecryptSecurityInfo` action. For more information, refer to the *IDOL Server Reference*.

Decrypt AES SecurityInfo Strings

The following procedure describes the algorithm to use to decrypt a security info string that is encrypted with an AES key file, with optional HMAC validation.

> **NOTE:** To decrypt a security info string, you need the AES key file that was used to generate it. Micro Focus strongly recommends that you secure your AES key file so that only your IDOL components and authorized administrators can access it.
>
> The IDOL Content component, IDOL Community component and DAH need access to the key file.

To decrypt an AES SecurityInfo String

1. Base64 decode the SecurityInfo String.

2. Split the decoded string on the left-most pipe character (|).

   The left side is the data length. The right side is the data (with the digest, if you use HMAC validation).

3. Check that the data length is equal to the length of the data with digest. If this check is not successful, fail the decryption.

4. (Optional) Use the following steps for HMAC validation:

   a. Split the data with digest on the left-most pipe character (|).

      The left side is the digest hex string. The right side is the data.

   b. Generate the HMAC key by taking a SHA-1 hash of your AES hexadecimal key string.

      > **NOTE:** This operation makes the key string case-sensitive.

     c. Calculate the HMAC_SHA512 value of the data.

     d. Compare the digest hex string to the HMAC_SHA512 value to verify the digest. If this check is not successful, fail the decryption.

5. Select the first 16 bytes of the data. This is the AES initialization vector.

6. Use AES-CBC to decrypt the remaining data, by using the IV and the 256-bit key from your AES hexadecimal key string.

   The decrypted data has the prefix `AUTN:`. If this string is not present, fail the decryption.

7. Use zlib to decompress the data after the `AUTN:` prefix.

Appendix D: Languages and Language Files

This appendix provides reference information related to languages and language files.

Supported Languages and Common Encodings

This section lists the languages that IDOL Server supports, and the most common encodings for each language.

You can set all IDOL Server `Encodings` settings to `UTF8` or `UCS2`. The internal IDOL Server storage encoding is `UTF8`.

Acehnese

Acehnese		
	Script:	UTF8
	[*MyLanguage*] section name:	ACEHNESE
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Afrikaans

Afrikaans		
	Script:	Latin

	[*MyLanguage*] section name:	AFRIKAANS
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1252/ISO-8859-1 UTF-8	ASCII UTF8

Albanian

Albanian		
	Script:	Latin
	[*MyLanguage*] section name:	ALBANIAN
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1252/ISO-8859-1 UTF-8	ASCII UTF8

Amharic

Amharic		
	Script:	UTF8
	[*MyLanguage*] section name:	AMHARIC
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Arabic

Arabic [1]		
	Script:	Arabic
	[*MyLanguage*] section name:	ARABIC
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1256	ARABIC

[1]A stemming algorithm is available for this language and is applied by default. If you do not want to apply stemming to this language, set **Stemming** to **False** for this language.

	ISO-8859-6	ARABIC_ISO
	UTF-8	UTF8

Armenian

Armenian		
	Script:	UTF8
	[*MyLanguage*] section name:	ARMENIAN
	For encoding:	Set Encodings parameter to:
	UTF-8	UTF8

Azeri

Azeri		
	Script:	Cyrillic
	[*MyLanguage*] section name:	AZERI
	For encoding:	Set Encodings parameter to:
	Windows-CP1251	CYRILLIC
	KOI8-R	CYRILLIC_KOI8
	ISO-8859-5	CYRILLIC_ISO
	UTF-8	UTF8

Basque

Basque		
	Script:	Latin
	[*MyLanguage*] section name:	BASQUE
	For encoding:	Set Encodings parameter to:
	Windows-CP1252/ISO-8859-1	ASCII
	UTF-8	UTF8

Belorussian

Belorussian		
	Script:	Cyrillic
	[*MyLanguage*] section name:	BELORUSSIAN
	For encoding:	Set Encodings parameter to:
	Windows-CP1251	CYRILLIC
	KOI8-R	CYRILLIC_KOI8
	ISO-8859-5	CYRILLIC_ISO
	UTF-8	UTF8

Bengali

Bengali		
	Script:	UTF8
	[*MyLanguage*] section name:	BENGALI
	For encoding:	Set Encodings parameter to:
	UTF-8	UTF8

Berber

Berber		
	Script:	UTF8
	[*MyLanguage*] section name:	BERBER
	For encoding:	Set Encodings parameter to:
	UTF-8	UTF8

Bihari

Bihari		

	Script:	UTF8
	[*MyLanguage*] section name:	BIHARI
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Bikol

Bikol		
	Script:	UTF8
	[*MyLanguage*] section name:	BIKOL
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Bishnupriya

Bishnupriya		
	Script:	UTF8
	[*MyLanguage*] section name:	BISHNUPRIYA
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Bosnian

Bosnian		
	Script:	Latin
	[*MyLanguage*] section name:	BOSNIAN
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1250	EASTERNEUROPEAN
	ISO-8859-2	EASTERNEUROPEAN_ISO
	UTF-8	UTF8

Breton

Breton		
	Script:	Latin
	[*MyLanguage*] section name:	BRETON
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1252/ISO-8859-1 UTF-8	ASCII UTF8

Bulgarian

Bulgarian		
	Script:	Cyrillic
	[*MyLanguage*] section name:	BULGARIAN
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1251 KOI8-R ISO-8859-5 UTF-8	CYRILLIC CYRILLIC_KOI8 CYRILLIC_ISO UTF8

Burmese

Burmese		
	Script:	UTF8
	[*MyLanguage*] section name:	BURMESE
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Catalan

Catalan [1]		
	Script:	Latin
	[*MyLanguage*] section name:	CATALAN
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1252/ISO-8859-1 UTF-8	ASCII UTF8

Cebuano

Cebuano		
	Script:	UTF8
	[*MyLanguage*] section name:	CEBUANO
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Cherokee

Cherokee		
	Script:	UTF8
	[*MyLanguage*] section name:	CHEROKEE
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

[1]A stemming algorithm is available for this language and is applied by default. If you do not want to apply stemming to this language, set **Stemming** to **False** for this language.

Chinese Traditional

Chinese traditional[1]		
	Script:	Big-5
	[*MyLanguage*] section name:	CHINESE
	For encoding:	Set `Encodings` parameter to:
	Big-5 UTF-8	CHINESETRADITIONAL UTF8

Chinese Simplified

Chinese simplified[2]		
	Script:	GB2312-80
	[*MyLanguage*] section name:	CHINESE
	For encoding:	Set `Encodings` parameter to:
	gb2312 UTF-8	CHINESESIMPLIFIED UTF8

Chuvash

Chuvash		
	Script:	UTF8
	[*MyLanguage*] section name:	CHUVASH
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

[1]The language has stemming embedded in sentence breaking.

[2]The language has stemming embedded in sentence breaking.

Croatian

Croatian		
	Script:	Latin
	[*MyLanguage*] section name:	CROATIAN
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1250 ISO-8859-2 UTF-8	EASTERNEUROPEAN EASTERNEUROPEAN_ISO UTF8

Czech

Czech [1]		
	Script:	Latin
	[*MyLanguage*] section name:	CZECH
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1250 ISO-8859-2 UTF-8	EASTERNEUROPEAN EASTERNEUROPEAN_ISO UTF8

Danish

Danish [2]		
	Script:	Latin
	[*MyLanguage*] section name:	DANISH
	For encoding:	Set `Encodings` parameter to:

[1]A stemming algorithm is available for this language and is applied by default. If you do not want to apply stemming to this language, set **Stemming** to **False** for this language.

[2]A stemming algorithm is available for this language and is applied by default. If you do not want to apply stemming to this language, set **Stemming** to **False** for this language.

| | Windows-CP1252/ISO-8859-1 | ASCII |
| | UTF-8 | UTF8 |

Divehi

Divehi		
	Script:	UTF8
	[*MyLanguage*] section name:	DIVEHI
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Dutch

Dutch [1]		
	Script:	Latin
	[*MyLanguage*] section name:	DUTCH
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1252/ISO-8859-1	ASCII
	UTF-8	UTF8

English

English [2]		
	Script:	Latin
	[*MyLanguage*] section name:	ENGLISH
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1252/ISO-8859-1	ASCII
	UTF-8	UTF8

[1]A stemming algorithm is available for this language and is applied by default. If you do not want to apply stemming to this language, set **Stemming** to **False** for this language.

[2]A stemming algorithm is available for this language and is applied by default. If you do not want to apply stemming to this language, set **Stemming** to **False** for this language.

Erzya

Erzya		
	Script:	UTF8
	[*MyLanguage*] section name:	ERZYA
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Esperanto

Esperanto		
	Script:	UTF8
	[*MyLanguage*] section name:	ESPERANTO
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Estonian

Estonian		
	Script:	Latin
	[*MyLanguage*] section name:	ESTONIAN
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1257 ISO-8859-4 UTF-8	NORTHERNEUROPEAN NORTHERNEUROPEAN_ISO UTF8

Ethiopic

Ethiopic		
	Script:	UTF8

	[*MyLanguage*] section name:	ETHIOPIC
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Faroese

Faroese		
	Script:	Latin
	[*MyLanguage*] section name:	FAROESE
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1252/ISO-8859-1	ASCII
	UTF-8	UTF8

Finnish

Finnish [1]		
	Script:	Latin
	[*MyLanguage*] section name:	FINNISH
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1252/ISO-8859-1	ASCII
	UTF-8	UTF8

French

French [2]		
	Script:	Latin
	[*MyLanguage*] section name:	FRENCH

[1]A stemming algorithm is available for this language and is applied by default. If you do not want to apply stemming to this language, set **Stemming** to **False** for this language.

[2]A stemming algorithm is available for this language and is applied by default. If you do not want to apply stemming to this language, set **Stemming** to **False** for this language.

	For encoding:	Set Encodings parameter to:
	Windows-CP1252/ISO-8859-1	ASCII
	UTF-8	UTF8

Frisian

Frisian		
	Script:	UTF8
	[*MyLanguage*] section name:	FRISIAN
	For encoding:	Set Encodings parameter to:
	UTF-8	UTF8

Gaelic

Gaelic		
	Script:	Latin
	[*MyLanguage*] section name:	GAELIC
	For encoding:	Set Encodings parameter to:
	Windows-CP1252/ISO-8859-1	ASCII
	UTF-8	UTF8

Galician

Galician		
	Script:	Latin
	[*MyLanguage*] section name:	GALICIAN
	For encoding:	Set Encodings parameter to:
	Windows-CP1252/ISO-8859-1	ASCII
	UTF-8	UTF8

Georgian

Georgian		
	Script:	UTF8
	[*MyLanguage*] section name:	GEORGIAN
	For encoding:	Set Encodings parameter to:
	UTF-8	UTF8

German

German [1]		
	Script:	Latin
	[*MyLanguage*] section name:	GERMAN
	For encoding:	Set Encodings parameter to:
	Windows-CP1252/ISO-8859-1 UTF-8	ASCII UTF8

Gilaki

Gilaki		
	Script:	UTF8
	[*MyLanguage*] section name:	GILAKI
	For encoding:	Set Encodings parameter to:
	UTF-8	UTF8

[1]A stemming algorithm is available for this language and is applied by default. If you do not want to apply stemming to this language, set **Stemming** to **False** for this language.

Greek

Greek [1]		
	Script:	Greek
	[*MyLanguage*] section name:	GREEK
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1253	GREEK
	ISO-8859-7	GREEK_ISO
	UTF-8	UTF8

Greenlandic

Greenlandic		
	Script:	Latin
	[*MyLanguage*] section name:	GREENLANDIC
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1257	NORTHERNEUROPEAN
	ISO-8859-4	NORTHERNEUROPEAN_ISO
	UTF-8	UTF8

Guarani

Guarani		
	Script:	UTF8
	[*MyLanguage*] section name:	GUARANI
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

[1]A stemming algorithm is available for this language and is applied by default. If you do not want to apply stemming to this language, set **Stemming** to **False** for this language.

Gujarati

Gujarati		
	Script:	UTF8
	[*MyLanguage*] section name:	GUJARATI
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Haitian

Haitian		
	Script:	UTF8
	[*MyLanguage*] section name:	HAITIAN
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Hausa

Hausa		
	Script:	UTF8
	[*MyLanguage*] section name:	HAUSA
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Hawaiian

Hawaiian		
	Script:	UTF8
	[*MyLanguage*] section name:	HAWAIIAN
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Hebrew

Hebrew [1]		
	Script:	Hebrew
	[*MyLanguage*] section name:	HEBREW
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1255 ISO-8859-8 UTF-8	HEBREW HEBREW_ISO UTF8

Hindi

Hindi		
	Script:	UTF8
	[*MyLanguage*] section name:	HINDI
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Hungarian

Hungarian [2]		
	Script:	Latin
	[*MyLanguage*] section name:	HUNGARIAN
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1250 ISO-8859-2 UTF-8	EASTERNEUROPEAN EASTERNEUROPEAN_ISO UTF8

[1]A stemming algorithm is available for this language and is applied by default. If you do not want to apply stemming to this language, set **Stemming** to **False** for this language.

[2]A stemming algorithm is available for this language and is applied by default. If you do not want to apply stemming to this language, set **Stemming** to **False** for this language.

Icelandic

Icelandic		
	Script:	Latin
	[*MyLanguage*] section name:	ICELANDIC
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1252/ISO-8859-1 UTF-8	ASCII UTF8

Igbo

Igbo		
	Script:	UTF8
	[*MyLanguage*] section name:	IGBO
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Ilokano

Ilokano		
	Script:	UTF8
	[*MyLanguage*] section name:	ILOKANO
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Indonesian

Indonesian		
	Script:	Latin
	[*MyLanguage*] section name:	INDONESIAN

	For encoding:	Set `Encodings` parameter to:
	Windows-CP1252/ISO-8859-1 UTF-8	ASCII UTF8

Italian

Italian [1]		
	Script:	Latin
	[*MyLanguage*] section name:	ITALIAN
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1252/ISO-8859-1 UTF-8	ASCII UTF8

Japanese

Japanese [2]		
	Script:	Japanese
	[*MyLanguage*] section name:	JAPANESE
	For encoding:	Set `Encodings` parameter to:
	Shift-JIS EUC JIS UTF-8	SHIFTJIS EUC JIS UTF8

Javanese

Javanese		
	Script:	UTF8

---

[1]A stemming algorithm is available for this language and is applied by default. If you do not want to apply stemming to this language, set **Stemming** to **False** for this language.

[2]The language has stemming embedded in sentence breaking.

	[*MyLanguage*] section name:	JAVANESE
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Kalmyk

Kalmyk		
	Script:	UTF8
	[*MyLanguage*] section name:	KALMYK
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Kannada

Kannada		
	Script:	UTF8
	[*MyLanguage*] section name:	KANNADA
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Kapampangan

Kapampangan		
	Script:	UTF8
	[*MyLanguage*] section name:	KAPAMPANGAN
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Kazakh

Kazakh		

	Script:	Cyrillic
	[*MyLanguage*] section name:	KAZAKH
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1251	CYRILLIC
	KOI8-R	CYRILLIC_KOI8
	ISO-8859-5	CYRILLIC_ISO
	UTF-8	UTF8

Khmer

Khmer		
	Script:	UTF8
	[*MyLanguage*] section name:	KHMER
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Kikongo

Kikongo		
	Script:	UTF8
	[*MyLanguage*] section name:	KIKONGO
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Kinyarwanda

Kinyarwanda		
	Script:	UTF8
	[*MyLanguage*] section name:	KINYARWANDA
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Kirundi

Kirundi		
	Script:	UTF8
	[*MyLanguage*] section name:	KIRUNDI
	For encoding:	Set Encodings parameter to:
	UTF-8	UTF8

Komi

Komi		
	Script:	UTF8
	[*MyLanguage*] section name:	KOMI
	For encoding:	Set Encodings parameter to:
	UTF-8	UTF8

Korean

Korean [1]		
	Script:	Hangul
	[*MyLanguage*] section name:	KOREAN
	For encoding:	Set Encodings parameter to:
	KS C 5601-1987 KS C 5601-1992 UTF-8	KOREAN KOREAN UTF8

Kurdish

Kurdish		

[1]The language has stemming embedded in sentence breaking.

	Script:	Latin
	[*MyLanguage*] section name:	KURDISH
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1252/ISO-8859-1 UTF-8	ASCII UTF8

Kyrgyz

Kyrgyz		
	Script:	Cyrillic
	[*MyLanguage*] section name:	KYRGYZ
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1251 KOI8-R ISO-8859-5 UTF-8	CYRILLIC CYRILLIC_KOI8 CYRILLIC_ISO UTF8

Lao

Lao		
	Script:	UTF8
	[*MyLanguage*] section name:	LAO
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Lappish

Lappish		
	Script:	Latin
	[*MyLanguage*] section name:	LAPPISH
	For encoding:	Set `Encodings` parameter to:

	Windows-CP1257	NORTHERNEUROPEAN
	ISO-8859-4	NORTHERNEUROPEAN_ISO
	UTF-8	UTF8

Latin

Latin[1]		
	Script:	Latin
	[*MyLanguage*] section name:	LATIN
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1252/ISO-8859-1	ASCII
	UTF-8	UTF8

Latvian

Latvian		
	Script:	Latin
	[*MyLanguage*] section name:	LATVIAN
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1257	NORTHERNEUROPEAN
	ISO-8859-4	NORTHERNEUROPEAN_ISO
	UTF-8	UTF8

Lingala

Lingala		
	Script:	UTF8
	[*MyLanguage*] section name:	LINGALA
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

[1]A stemming algorithm is available for this language and is applied by default. If you do not want to apply stemming to this language, set **Stemming** to **False** for this language.

Lithuanian

Lithuanian		
	Script:	Latin
	[*MyLanguage*] section name:	LITHUANIAN
	For encoding:	Set Encodings parameter to:
	Windows-CP1257	NORTHERNEUROPEAN
	ISO-8859-4	NORTHERNEUROPEAN_ISO
	UTF-8	UTF8

Luxembourgish

Luxembourgish		
	Script:	Latin
	[*MyLanguage*] section name:	LUXEMBOURGISH
	For encoding:	Set Encodings parameter to:
	Windows-CP1252/ISO-8859-1	ASCII
	UTF-8	UTF8

Macedonian

Macedonian		
	Script:	Cyrillic
	[*MyLanguage*] section name:	MACEDONIAN
	For encoding:	Set Encodings parameter to:
	Windows-CP1251	CYRILLIC
	KOI8-R	CYRILLIC_KOI8
	ISO-8859-5	CYRILLIC_ISO
	UTF-8	UTF8

Malagasy

Malagasy		
	Script:	UTF8
	[*MyLanguage*] section name:	MALAGASY
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Malay

Malay		
	Script:	Latin
	[*MyLanguage*] section name:	MALAY
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1252/ISO-8859-1	ASCII
	UTF-8	UTF8

Malayalam

Malayalam		
	Script:	UTF8
	[*MyLanguage*] section name:	MALAYALAM
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Maltese

Maltese		
	Script:	UTF8
	[*MyLanguage*] section name:	MALTESE

	For encoding:		Set `Encodings` parameter to:
	UTF-8		UTF8

Manipuri

Manipuri			
	Script:		UTF8
	[*MyLanguage*] section name:		MANIPURI
	For encoding:		Set `Encodings` parameter to:
	UTF-8		UTF8

Maori

Maori			
	Script:		Latin1
	[*MyLanguage*] section name:		MAORI
	For encoding:		Set `Encodings` parameter to:
	Windows-CP1252/ISO-8859-1		ASCII
	UTF-8		UTF8

Marathi

Marathi			
	Script:		UTF8
	[*MyLanguage*] section name:		MARATHI
	For encoding:		Set `Encodings` parameter to:
	UTF-8		UTF8

Mazandarani

Mazandarani		

	Script:	UTF8
	[*MyLanguage*] section name:	MAZANDARANI
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Mirandese

Mirandese		
	Script:	UTF8
	[*MyLanguage*] section name:	MIRANDESE
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Mongolian

Mongolian		
	Script:	Cyrillic
	[*MyLanguage*] section name:	MONGOLIAN
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1251	CYRILLIC
	KOI8-R	CYRILLIC_KOI8
	ISO-8859-5	CYRILLIC_ISO
	UTF-8	UTF8

Nahuatl

Nahuatl		
	Script:	UTF8
	[*MyLanguage*] section name:	NAHUATL
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Navajo

Navajo		
	Script:	UTF8
	[*MyLanguage*] section name:	NAVAJO
	For encoding:	Set Encodings parameter to:
	UTF-8	UTF8

Ndebele

Ndebele		
	Script:	UTF8
	[*MyLanguage*] section name:	NDEBELE
	For encoding:	Set Encodings parameter to:
	UTF-8	UTF8

Nepali

Nepali		
	Script:	UTF8
	[*MyLanguage*] section name:	NEPALI
	For encoding:	Set Encodings parameter to:
	UTF-8	UTF8

Newari

Newari		
	Script:	UTF8
	[*MyLanguage*] section name:	NEWARI
	For encoding:	Set Encodings parameter to:
	UTF-8	UTF8

Norwegian

Norwegian [1]		
	Script:	Latin
	[*MyLanguage*] section name:	NORWEGIAN
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1252/ISO-8859-1 UTF-8	ASCII UTF8

Oriya

Oriya		
	Script:	UTF8
	[*MyLanguage*] section name:	ORIYA
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Ossetian

Ossetian		
	Script:	UTF8
	[*MyLanguage*] section name:	OSSETIAN
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Panjabi

Panjabi		

[1]A stemming algorithm is available for this language and is applied by default. If you do not want to apply stemming to this language, set **Stemming** to **False** for this language.

	Script:	UTF8
	[*MyLanguage*] section name:	PANJABI
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Papiamentu

Papiamentu		
	Script:	UTF8
	[*MyLanguage*] section name:	PAPIAMENTU
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Persian

Persian		
	Script:	UTF8
	[*MyLanguage*] section name:	PERSIAN
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Polish

Polish [1]		
	Script:	Latin
	[*MyLanguage*] section name:	POLISH
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1250	EASTERNEUROPEAN

[1]A stemming algorithm is available for this language and is applied by default. If you do not want to apply stemming to this language, set **Stemming** to **False** for this language.

	ISO-8859-2		EASTERNEUROPEAN_ISO
	UTF-8		UTF8

Portuguese

Portuguese [1]		
	Script:	Latin
	[*MyLanguage*] section name:	PORTUGUESE
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1252/ISO-8859-1	ASCII
	UTF-8	UTF8

Pushto

Pushto		
	Script:	UTF8
	[*MyLanguage*] section name:	PUSHTO
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Quechua

Quechua		
	Script:	UTF8
	[*MyLanguage*] section name:	QUECHUA
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

[1]A stemming algorithm is available for this language and is applied by default. If you do not want to apply stemming to this language, set **Stemming** to **False** for this language.

Rhaeto-Romance

Rhaeto-Romance		
	Script:	UTF8
	[*MyLanguage*] section name:	RHAETO-ROMANCE
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Romanian

Romanian [1]		
	Script:	Latin
	[*MyLanguage*] section name:	ROMANIAN
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1250	EASTERNEUROPEAN
	ISO-8859-2	EASTERNEUROPEAN_ISO
	UTF-8	UTF8

Russian

Russian [2]		
	Script:	Cyrillic
	[*MyLanguage*] section name:	RUSSIAN
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1251	CYRILLIC
	KOI8-R	CYRILLIC_KOI8

[1]A stemming algorithm is available for this language and is applied by default. If you do not want to apply stemming to this language, set **Stemming** to **False** for this language.

[2]A stemming algorithm is available for this language and is applied by default. If you do not want to apply stemming to this language, set **Stemming** to **False** for this language.

	ISO-8859-5	CYRILLIC_ISO
	UTF-8	UTF8

Sakha

Sakha		
	Script:	UTF8
	[*MyLanguage*] section name:	SAKHA
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Sami

Sami		
	Script:	UTF8
	[*MyLanguage*] section name:	SAMI
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Sanskrit

Sanskrit		
	Script:	UTF8
	[*MyLanguage*] section name:	SANSKRIT
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Serbian

Serbian		
	Script:	Cyrillic

	[*MyLanguage*] section name:	SERBIAN
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1251	CYRILLIC
	KOI8-R	CYRILLIC_KOI8
	ISO-8859-5	CYRILLIC_ISO
	UTF-8	UTF8

Sesotho

Sesotho		
	Script:	UTF8
	[*MyLanguage*] section name:	SESOTHO
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Sesotho sa Leboa

Sesotho sa Leboa		
	Script:	UTF8
	[*MyLanguage*] section name:	SESOTHOSALEBOA
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Singhalese

Singhalese		
	Script:	UTF8
	[*MyLanguage*] section name:	SINGHALESE
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Siswant

Siswant		
	Script:	UTF8
	[*MyLanguage*] section name:	SISWANT
	For encoding:	Set Encodings parameter to:
	UTF-8	UTF8

Slovak

Slovak [1]		
	Script:	Latin
	[*MyLanguage*] section name:	SLOVAK
	For encoding:	Set Encodings parameter to:
	Windows-CP1250	EASTERNEUROPEAN
	ISO-8859-2	EASTERNEUROPEAN_ISO
	UTF-8	UTF8

Slovenian

Slovenian		
	Script:	Latin
	[*MyLanguage*] section name:	SLOVENIAN
	For encoding:	Set Encodings parameter to:
	Windows-CP1250	EASTERNEUROPEAN
	ISO-8859-2	EASTERNEUROPEAN_ISO
	UTF-8	UTF8

[1]A stemming algorithm is available for this language and is applied by default. If you do not want to apply stemming to this language, set **Stemming** to **False** for this language.

Somali

Somali		
	Script:	Latin
	[*MyLanguage*] section name:	SOMALI
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1252/ISO-8859-1 UTF-8	ASCII UTF8

Sorbian

Sorbian		
	Script:	Latin
	[*MyLanguage*] section name:	SORBIAN
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1250 ISO-8859-2 UTF-8	EASTERNEUROPEAN EASTERNEUROPEAN_ISO UTF8

Spanish

Spanish [1]		
	Script:	Latin
	[*MyLanguage*] section name:	SPANISH
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1252/ISO-8859-1 UTF-8	ASCII UTF8

[1]A stemming algorithm is available for this language and is applied by default. If you do not want to apply stemming to this language, set **Stemming** to **False** for this language.

Sranan

Sranan		
	Script:	UTF8
	[*MyLanguage*] section name:	SRANAN
	For encoding:	Set Encodings parameter to:
	UTF-8	UTF8

Sundanese

Sundanese		
	Script:	UTF8
	[*MyLanguage*] section name:	SUNDANESE
	For encoding:	Set Encodings parameter to:
	UTF-8	UTF8

Swahili

Swahili		
	Script:	Latin
	[*MyLanguage*] section name:	SWAHILI
	For encoding:	Set Encodings parameter to:
	Windows-CP1252/ISO-8859-1 UTF-8	ASCII UTF8

Swedish

Swedish [1]		

[1]A stemming algorithm is available for this language and is applied by default. If you do not want to apply stemming to this language, set **Stemming** to **False** for this language.

	Script:		Latin
	[*MyLanguage*] section name:		SWEDISH
	For encoding:		Set `Encodings` parameter to:
	Windows-CP1252/ISO-8859-1		ASCII
	UTF-8		UTF8

Syriac

Syriac			
	Script:		UTF8
	[*MyLanguage*] section name:		SYRIAC
	For encoding:		Set `Encodings` parameter to:
	UTF-8		UTF8

Tagalog

Tagalog			
	Script:		Latin
	[*MyLanguage*] section name:		TAGALOG
	For encoding:		Set `Encodings` parameter to:
	Windows-CP1252/ISO-8859-1		ASCII
	UTF-8		UTF8

Tahitian

Tahitian			
	Script:		UTF8
	[*MyLanguage*] section name:		TAHITIAN
	For encoding:		Set `Encodings` parameter to:
	UTF-8		UTF8

Tajik

Tajik		
	Script:	Cyrillic
	[*MyLanguage*] section name:	TAJIK
	For encoding:	Set Encodings parameter to:
	Windows-CP1251 KOI8-R ISO-8859-5 UTF-8	CYRILLIC CYRILLIC_KOI8 CYRILLIC_ISO UTF8

Tamil

Tamil		
	Script:	UTF8
	[*MyLanguage*] section name:	TAMIL
	For encoding:	Set Encodings parameter to:
	UTF-8	UTF8

Tatar

Tatar		
	Script:	Cyrillic
	[*MyLanguage*] section name:	TATAR
	For encoding:	Set Encodings parameter to:
	Windows-CP1251 KOI8-R ISO-8859-5 UTF-8	CYRILLIC CYRILLIC_KOI8 CYRILLIC_ISO UTF8

Telugu

Telugu		
	Script:	UTF8
	[*MyLanguage*] section name:	TELUGU
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Tetum

Tetum		
	Script:	UTF8
	[*MyLanguage*] section name:	TETUM
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Thai

Thai		
	Script:	Thai
	[*MyLanguage*] section name:	THAI
	For encoding:	Set `Encodings` parameter to:
	Windows-CP874/ISO-8859-11	THAI
	UTF-8	UTF8

Tibetan

Tibetan		
	Script:	UTF8
	[*MyLanguage*] section name:	TIBETAN

	For encoding:	Set Encodings parameter to:
	UTF-8	UTF8

Tokpisin

Tokpisin		
	Script:	UTF8
	[*MyLanguage*] section name:	TOKPISIN
	For encoding:	Set Encodings parameter to:
	UTF-8	UTF8

Tongan

Tongan		
	Script:	UTF8
	[*MyLanguage*] section name:	TONGAN
	For encoding:	Set Encodings parameter to:
	UTF-8	UTF8

Tsonga

Tsonga		
	Script:	UTF8
	[*MyLanguage*] section name:	TSONGA
	For encoding:	Set Encodings parameter to:
	UTF-8	UTF8

Tswana

Tswana		

	Script:	UTF8
	[*MyLanguage*] section name:	TSWANA
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Turkish

Turkish		
	Script:	Latin
	[*MyLanguage*] section name:	TURKISH
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1254/ISO-8859-9 UTF-8	TURKISH UTF8

Turkmen

Turkmen		
	Script:	UTF8
	[*MyLanguage*] section name:	TURKMEN
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Ukrainian

Ukrainian		
	Script:	Cyrillic
	[*MyLanguage*] section name:	UKRAINIAN
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1251 KOI8-R ISO-8859-5	CYRILLIC CYRILLIC_KOI8 CYRILLIC_ISO

	UTF-8	UTF8

Urdu

Urdu		
	Script:	UTF8
	[*MyLanguage*] section name:	URDU
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Uyghur

Uyghur		
	Script:	UTF8
	[*MyLanguage*] section name:	UYGHUR
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Uzbek

Uzbek		
	Script:	Cyrillic
	[*MyLanguage*] section name:	UZBEK
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1251	CYRILLIC
	KOI8-R	CYRILLIC_KOI8
	ISO-8859-5	CYRILLIC_ISO
	UTF-8	UTF8

Valencian

Valencian		
	Script:	Latin
	[*MyLanguage*] section name:	VALENCIAN
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1252/ISO-8859-1 UTF-8	ASCII UTF8

Venda

Venda		
	Script:	UTF8
	[*MyLanguage*] section name:	VENDA
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Vietnamese

Vietnamese		
	Script:	Vietnamese
	[*MyLanguage*] section name:	VIETNAMESE
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1258 UTF-8	VIETNAMESE UTF8

Waraywaray

Waraywaray		
	Script:	UTF8

	[*MyLanguage*] section name:	WARAYWARAY
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Welsh

Welsh [1]		
	Script:	Latin
	[*MyLanguage*] section name:	WELSH
	For encoding:	Set `Encodings` parameter to:
	Windows-CP1252/ISO-8859-1 UTF-8	ASCII UTF8

Wolof

Wolof		
	Script:	UTF8
	[*MyLanguage*] section name:	WOLOF
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Xhosa

Xhosa		
	Script:	UTF8
	[*MyLanguage*] section name:	XHOSA
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

[1]A stemming algorithm is available for this language and is applied by default. If you do not want to apply stemming to this language, set **Stemming** to **False** for this language.

Yiddish

Yiddish		
	Script:	UTF8
	[*MyLanguage*] section name:	YIDDISH
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Yoruba

Yoruba		
	Script:	UTF8
	[*MyLanguage*] section name:	YORUBA
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Zulu

Zulu		
	Script:	UTF8
	[*MyLanguage*] section name:	ZULU
	For encoding:	Set `Encodings` parameter to:
	UTF-8	UTF8

Supported Encodings

The table below lists the encodings that the IDOL Content component supports for text processing and indexing. Not all encodings are valid for all supported languages. For a list of the most common supported encodings for each language, see Supported Languages and Common Encodings, on page 513.

Supported Encodings

IDOL name	XML name	ISO name
ARABIC	Windows-1256	CP_1256
ARABIC_ISO	ISO-8859-6	ISO8859-6
ARABIC_MAC	x-mac-arabic	CP_10004
ASCII	ISO-8859-1	CP_ACP
ASCII_IBM	IBM850	CP_850
CHINESESIMPLIFIED	GBK	CP_936
CHINESETRADITIONAL	Big5	CP_950
CYRILLIC	Windows-1251	CP_1251
CYRILLIC_DOS	IBM866	CP_866
CYRILLIC_ISO	ISO-8859-5	ISO8859-5
CYRILLIC_KOI8	KOI8-R	CP_21866
EASTERNEUROPEAN	Windows-1250	CP_1250
EASTERNEUROPEAN_ISO	ISO-8859-2	ISO8859-2
EUC	EUC-JP	
GREEK	Windows-1253	CP_1253
GREEK_ISO	ISO-8859-7	ISO8859-7
HEBREW	Windows-1255	CP_1255
HEBREW_ISO	ISO-8859-8	ISO8859-8
JIS	JIS_Encoding	
KOREAN	KS_C_5601-1987	CP_949
LATIN3	ISO-8859-3	ISO8859-3
LATIN5	ISO-8859-9	ISO8859-9
LATIN6	ISO-8859-14	ISO8859-14
LATIN7	ISO-8859-13	ISO8859-13
LATIN9	ISO-8859-15	ISO8859-15
NORTHERNEUROPEAN	Windows-1257	CP_1257

Supported Encodings , continued

IDOL name	XML name	ISO name
NORTHERNEUROPEAN_ISO	ISO-8859-4	ISO8859-4
SHIFTJIS	Shift_JIS	CP_932
THAI	TIS-620	CP_874
TURKISH	Windows-1254	CP_1254
UCS2	ISO-10646-UCS-2	ISO-10646
UTF8	UTF-8	CP_UTF8
VIETNAMESE	Windows-1258	CP_1258
WESTERNEUROPEAN	Windows-1252	CP_1252

TermSize Parameter

The `TermSize` parameter in the `[Server]` section of the IDOL Content component configuration file allows you to specify the maximum number of characters that any term in the Content data index can contain. The default value is 20.

For certain languages, this default term size might be too low. In this case, you can increase the value of the `TermSize` parameter.

> **NOTE:** You cannot apply the `TermSize` parameter individually to a language.

The table below shows the `TermSize` value that Micro Focus recommends for the specified languages. If you have content in more than one language, use the larger value.

Recommended `TermSize` value per language

Language	TermSize value
English and other European languages	20
Arabic	30
Chinese	30
Hebrew	30
Korean	30
Japanese	30

Recommended TermSize value per language, continued

Language	TermSize value
Thai	30
German	30
Greek	40

Per-Language Sentence-Breaking Files

For languages in which words are not delimited by spaces (Japanese, Chinese, Thai, and Korean), the IDOL Content component uses sentence-breaking libraries. In a default IDOL Content component installation, these files are stored in the `IDOL/langfiles` directory.

If you run Content on a UNIX platform, specify the `LD_LIBRARY_PATH` to ensure that Content can find the sentence-breaking files that it requires.

The following tables list the files that the individual languages require.

- **Japanese**

NT	UNIX
`japanesebreaking.dll`	`japanesebreaking.so`
`\jpn-cha\cforms.cha`	`/jpn-cha/cforms.cha`
`\jpn-cha\chadic.da`	`/jpn-cha/chadic.da`
`\jpn-cha\chadic.lex`	`/jpn-cha/chadic.lex`
`\jpn-cha\chasenrc`	`/jpn-cha/chasenrc`
`\jpn-cha\connect.cha`	`/jpn-cha/connect.cha`
`\jpn-cha\ctypes.cha`	`/jpn-cha/ctypes.cha`
`\jpn-cha\grammar.cha`	`/jpn-cha/grammar.cha`
`\jpn-cha\matrix.cha`	`/jpn-cha/matrix.cha`
`\jpn-cha\table.cha`	`/jpn-cha/table.cha`
`libchasen.dll`	

- **Traditional Chinese**

NT	UNIX
`chinesebreaking.dll`	`chinesebreaking.so`

NT	UNIX
big5togb.txt	big5togb.txt
wordlist.txt	wordlist.txt
chineseconvlist.txt	chineseconvlist.txt

- **Simplified Chinese**

NT	UNIX
chinesebreaking.dll	chinesebreaking.so
big5togb.txt	big5togb.txt
wordlist.txt	wordlist.txt
chineseconvlist.txt	chineseconvlist.txt

- **Thai**

NT	UNIX
thaibreaking.dll	thaibreaking.so
thaidict.txt	thaidict.txt
thaiconvlist.txt	thaiconvlist.txt

- **Korean**

NT	UNIX
koreanbreaking.dll	koreanbreaking.so
main.dat	main.dat
prob.dat	prob.dat
main.fst	main.fst
prob.fst	prob.fst
pos.nam	pos.nam
tag.nam	tag.nam
tagout.nam	tagout.nam
connection.txt	connection.txt
StopPosNam.txt	StopPosNam.txt
TagName.txt	TagName.txt
koreanconvlist.txt	koreanconvlist.txt

Stop Word Lists for Supported Languages

A stop word list (stop list) is a list of common words that the IDOL Content component does not index. Words such as *the* or *a* occur too frequently to carry any significance, and Content does not require them to understand the concept of text. Using a stop list to remove these words can improve query results and performance, and save index space.

Each language that Content supports needs a stop list; if the Content installer does not include a stop word list for the language that you want to use, you can create one.

You can use a standard text editor to create or edit a stop list. Stop word lists are located in the Content `IDOL/langfiles` directory. For example, you might want to add any words that occur in most or all of your documents, and which you do not need to search for.

For all operations, Content recognizes words as stop words irrespective of the encoding they are in. For example, in Russian you can list a stop word in the UTF-8 encoding in the stop word list file, and Content recognizes it if it occurs in a document in KOI8 encoding.

For simplicity, Micro Focus recommends that you type all the terms in the stop list in UTF-8 encoding. However, you can list the words in the stop list in any of the valid encodings for that language. For example, in Russian you can specify stop words in KOI8, UTF8, ISO, and so on.

You can specify words in uppercase or lowercase, and you can separate them with spaces or new lines.

> **NOTE:** If necessary, you can use different encodings in the same stop list file. You need to specify each word only once; that is, you do not need to specify the same word in several different encodings.
>
> For each encoding that you want to use, create a section in your stop list file. Give the section the same name as the language type that you are using (for example, `cyrillic_koi8`, `cyrillic_utf8`).

For example:

```
[cyrillic_utf8]
ДАЖЕ
ДЛЯ ДО
ЕЕ ЕГО
А БЕЗ БОЛЕЕ БЫ БЫЛs
```

Related Topics

- Supported Languages and Common Encodings, on page 513

Appendix E: Manually Create IDX Files

To manually create an IDX file, you create a text file that contains the data that you want to index into the IDOL Content component, formatted in IDOL fields. The fields store the data in a format that Content can index.

IDX Format

IDOL fields in IDX files

`#DREREFERENCE`	A unique reference string for the document. Usually this reference is a file name, URL, or a unique code number.
`#DRETITLE`	The title of the document. You can enter multiple lines.
`#DRECONTENT`	The content of the document. You can enter multiple lines. (This parameter is optional. However, if you do not enter `#DRECONTENT`, you must specify one or more `#DREFIELD` *NameN* fields. Otherwise, the document does not contain any content).

`#DREFIELD` *NameN*	The name of each `DREFIELD` that you are defining, and enter an appropriate value for it. You must enclose the field value with quotation marks ("). For example, to index customer details:
	
	`#DREFIELD surname1="Smith"`
	`#DREFIELD forename1="Peter"`
	`#DREFIELD title1="Mr."`
	`#DREFIELD surname2="Miller"`
	`#DREFIELD forename2="Susan"`
	`#DREFIELD title2="Dr."`
	
	If the document contains only one instance of the `DREFIELD` that you are defining, you do not need to add a qualifier to the name of the field. For example:
	
	`#DREFIELD company="Hewlett Packard Enterprise"`
	
	You can define the same `DREFIELD` with different values. For example:
	
	`#DREFIELD MyField="value1"`
	`#DREFIELD MyField="value2"`
	`#DREFIELD MyField="value3"`
	
	The field values can span multiple lines. For example:

IDOL fields in IDX files , continued

	```#DREFIELD MyField="line1```
	```line2```
	```line3"```
	
	#DREFIELD is optional. However, if you do not enter a DREFIELD *NameN* field, you must specify #DRECONTENT. Otherwise, the document does not contain any content.
	
	NOTE: DREFIELD names must not contain spaces, accents, or multibyte characters. If you use these text elements, Content removes them when it indexes the fields. You must also change any queries that reference field names that contain these elements to use the modified field name.

#DREDATE	The creation date of the document in the format that you specified for the DateFormatCSVs parameter in the IDOL Content component configuration file. By default, this is yyyy/mm/dd.
#DREDBNAME	The name of the database into which you want to index the document.
#DRESECTION *N*	The section number. If you are indexing a large document and want to divide it into smaller sections, you can give each section a DRESECTION number to index the defined sections as individual documents into the IDOL Content component. If you divide a document into sections: • the first section must be #DRESECTION 0. • the section numbers must be in numerical order. • apart from the #DRESECTION number and the #DRECONTENT, each section must contain the same Content field values. (See Section a Document, on the next page).
#DREENDDOC	Indicates the end of the document. You must enter this delimiter.

NOTE: The text file must start with #DREREFERENCE and end with #DREENDDOC.

Example

This is an example of a text file that the IDOL Content component can index:

```
#DREREFERENCE 392348A0
#DREFIELD authorname1="Brown"
#DREFIELD authorname2="Edgar"
#DREFIELD title="Dr."
#DREDATE 1998/08/06
#DRETITLE
Jurassic Molecules
```

```
#DRECONTENT
Scientists announced last week the successful reproduction of a possible precursor
to all life on Earth. The molecules consist of a part of DNA and the molecular
"scissors" responsible for destroying messenger RNA in humans.
Using a technique called test tube evolution, scientists created a nucleic acid
enzyme, the first known enzyme that uses an amino acid to start chemical activity.
Scientists hope that the creation of this molecule will lead to the elusive
precursor. The precursor, by definition, will have to contain both the genetic code
for replication and an enzyme to trigger self replication.
#DRETYPE text
#DREDBNAME Science
#DRESTORECONTENT y
#DREENDDOC
```

> **TIP:** You can submit data for Content to index by using the wizard on the **Index** tab in the Console section of IDOL Admin. For more information, refer to the *IDOL Admin User Guide.*

Section a Document

If a document that you want to index contains more than 500 words, divide it into sections to make it more manageable for the IDOL Content component. If you want to index XML rather than IDX, you do not need to section your data because Content automatically applies sectioning to it.

Declare a separate document for each section that you split the original document into. Give each section a #DRESECTION number.

If you divide a document into sections:

- You must name the first section #DRESECTION 0.

- You must put the section numbers in numerical order.

- You must put the content of each section into the #DRECONTENT field.

- You must make each #DRECONTENT no more than 500 words long.

- You must give each section the same DRE field values, except for the #DRESECTION number and the #DRECONTENT.

Example Text File

The following IDX file example shows a document that has been divided into sections:

```
#DREREFERENCE 392348A0
#DREFIELD authorname1="Brown"
#DREFIELD authorname2="Edgar"
#DREFIELD title="Dr."
#DREDATE 1998/08/06
#DRETITLE
```

```
Jurassic Molecules
#DRESECTION 0
#DRECONTENT
Scientists announced last week the successful reproduction of a possible precursor
to all life on Earth. The molecules consist of a part of DNA and the molecular
"scissors" responsible for destroying messenger RNA in humans.
Using a technique called test tube evolution, scientists created a nucleic acid
enzyme, the first known enzyme that uses an amino acid to start chemical activity.
Scientists hope that the creation of this molecule will lead to the elusive
precursor. The precursor, by definition, will have to contain both the genetic code
for replication and an enzyme to trigger self replication.
At this point, no naturally occurring hybrid enzymes have been found. Scientists
speculate that such enzymes may exist in nature and most certainly existed in
Earth's early history.
#DRETYPE text
#DREDBNAME Science
#DRESTORECONTENT y
#DREENDDOC
#DREREFERENCE 392348A0
#DREFIELD authorname1="Brown"
#DREFIELD authorname2="Edgar"
#DREFIELD title="Dr."
#DREDATE 1998/08/06
#DRETITLE
Jurassic Molecules
#DRESECTION 1
#DRECONTENT
Scientists have known for some time that the key ingredients for life are DNA, RNA,
and proteins. An interesting chicken-egg dilemma has developed: which came first,
RNA, DNA, or proteins? Many believe that a replicating RNA molecule is the likely
precursor to all life on Earth.
RNA serves as both a genetic molecule and an enzyme in the body, which scientists
believe strongly suggests the likelihood of an RNA precursor to all life. They
speculate that RNA was first, followed by DNA, the much more stable of the two. It
would serve as an efficient storehouse for the genetic code. Proteins, better
catalysts than RNA, likely evolved later as well. At some point, the current three-
based system developed from the initial one-based system of RNA.
Scientists hope that these scissors molecules may also have practical uses in
medicine, since the molecules can efficiently shred specific DNA. Theoretically, it
may be possible to tailor such a molecule to attack and shred harmful DNA from
pathogenic organisms. These molecules could be made to be activated only in
specific circumstances.
#DRETYPE text
#DREDBNAME Science
#DRESTORECONTENT y
#DREENDDOC
```

Appendix F: Category XML Format

This appendix describes the required structure of the XML file that you can use with the IDOL Category component to create categories.

Introduction

To use the IDOL Category component to use an XML file to create a category structure in IDOL:

- Import a hierarchy from an XML file, by running the `CategoryImportFromXML` action, as in this example:

  `action=CategoryImportFromXML&ImportFilename=MyCategory.xml&BuildNow=True`

  For information on importing category information by using the Category component, see Categorization, on page 157.

XML Format

This section lists and describes the tags that are allowed in the XML file from which you want to import a category structure.

You must include the `<autn:categories>` tag in the XML file; however, there are no required tags within `<autn:categories>`. If you include only the `<autn:categories>` tag, IDOL imports an empty category structure. To expand the category structure, use the `<autn:category>` tag and its children.

<autn:categories> (required)

The `<autn:categories>` tag marks the beginning of the XML categories that the IDOL Category component reads. When you use the `CategoryImportFromXML` action, Category reads the XML between the opening and closing `<autn:categories>` tags.

All categories (`<autn:category>`) are children of (`<autn:categories>`) when you export a category from the IDOL Category component to XML. You can then import the XML to the IDOL Category component. They should have the same format.

You must include an XML namespace in the tag. For example:

`<autn:categories xmlns:autn="http://schemas.autonomy.com/aci">`

Tag name	Number allowed	Required
<autn:category>	one or more	No

<autn:category>

The `<autn:category>` tag marks the limits of each category that you want to import in your XML file. You can include one or more `<autn:category>` tags inside the `<autn:categories>` tag, and `<autn:category>` tags can also contain child `<autn:category>` tags.

The following table lists the tags that are allowed in `<autn:category>`.

Tag name	Number allowed	Required
<autn:name> (required)	one	Yes
<autn:id>	one	No
<autn:parent>	one	No
<autn:refersto>	one	No
<autn:trainingelement>	one or more	No
<autn:simplecat>	one	No
<autn:relevancecat>	one	No
<autn:details>	one	No

<autn:name> (required)

The `<autn:name>` tag sets the name of the category. You must include one `<autn:name>` within each set of `<autn:category>` tags. For example:

`<autn:name>UKpolitics</autn:name>`

Tags allowed within `<autn:name>`: none

<autn:id>

The `<autn:id>` tag sets the IDOL category ID. If a category with this ID already exists in the server, IDOL uses the `OnConflict` ACI parameter to determine the action to take.

For further information, see the `CategoryImportFromXML OnConflict` parameter in the *IDOL Server Reference*.

\<autn:parent\>

The `<autn:parent>` tag identifies the ID of the parent category. This option is effective only if you set `Flat` to **True** in the `CategoryImportFromXML` action.

For further information, refer to the *IDOL Server Reference*.

\<autn:refersto\>

The `<autn:refersto>` tag identifies the ID of the category to which the new category refers. This information is used to create a category that refers to another, and which inherits its fields, training, and special documents. This option is effective only if you set `Flat` to **True** in the `CategoryImportFromXML` action.

For further information, refer to the *IDOL Server Reference*.

\<autn:trainingelement\>

The `<autn:trainingelement>` tag identifies the training element for a category. IDOL Server identifies concepts that belong to the category from this training set. You can include one or more `<autn:trainingelement>` tags in each set of `<autn:category>` tags.

The following table lists the tags that are allowed in `<autn:trainingelement>`.

Tag name	Number allowed	Required
One or more of these:		
[\<autn:type\>](#)	one	Yes
[\<autn:content\>](#)	one	See below
[\<autn:language\>](#)	one	No
[\<autn:reference\>](#)	one	See below
[\<autn:docid\>](#)	one	See below
[\<autn:database\>](#)	one	No

\<autn:type\>

The `<autn:type>` tag sets the type of training to be used by `<autn:trainingelement>`. Each `<autn:trainingelement>` should contain only one `<autn:type>` tag.

The following table describes the values that are valid for `<autn:type>`.

Options for \<autn:type\>	Description
TRAININGTEXT	Identifies the training type as text only.

Options for <autn:type>	Description

BOOLEAN	Identifies the training type as Boolean. You must define the Boolean operator in the `<autn:content>` tag. For example:
	
	`<autn:type>BOOLEAN</autn:type>`
	`<autn:content>(phone AND mobile)</autn:content>`

URLDOWNLOAD	Identifies text from a URL to use for training.
DREDOCUMENT	Identifies the training type as a document indexed in IDOL. The document is specified by either `autn:reference` or `autn:docid`.

<autn:content>

The `<autn:content>` tag specifies the actual training text or Boolean expression. This tag is required if <autn:type> is TRAININGTEXT, BOOLEAN, or URLDOWNLOAD.

For more information, see <autn:type>, on the previous page.

<autn:language>

The `<autn:language>` tag specifies the language of the training text. This tag is optional.

<autn:reference>

The `<autn:reference>` tag specifies the IDOL DREREFERENCE to use for training. This tag is required if `<autn:type>` is DREDOCUMENT, and you do not set `<autn:docid>`.

> **NOTE:** If the document is of the type DREDOCUMENT, you must set either `<autn:reference>` or `<autn:docid>`, but not both.

For more information, see <autn:type>, on the previous page and <autn:docid>, below.

<autn:docid>

The `<autn:docid>` tag specifies the IDOL DocID to use for training. This tag is required if `<autn:type>` is DREDOCUMENT, and you do not set `<autn:reference>`.

> **NOTE:** If the document is of the type DREDOCUMENT, you must set either `<autn:reference>` or `<autn:docid>`, but not both.

For more information, see <autn:type>, on the previous page and <autn:reference>, above.

<autn:database>

The `<autn:database>` tag specifies the IDOL database in which the training document is located. You can set only one database for each `<autn:trainingelement>` tag. This tag is optional.

<autn:simplecat>

The `<autn:simplecat>` tag specifies whether the category is a simple category. For example:

`<autn:simplecat>True</autn:simplecat>`

> **NOTE:** If you use this tag, do not set `<autn:relevancecat>`.

<autn:relevancecat>

The `<autn:relevancecat>` tag specifies whether the category is a relevance category. For example:

`<autn:relevancecat>False</autn:relevancecat>`

> **NOTE:** If you use this tag, do not set `<autn:simplecat>`.

<autn:details>

The `<autn:details>` tag sets training details for the category. You can include one set of `<autn:details>` within each set of `<autn:category>` tags.

The following table lists the tags that are allowed in `<autn:details>`.

Tag name	Number allowed	Required
<autn:generatedterms> and <autn:generatedweights>	one	See below
<autn:queryagenttnw>	one	No
<autn:modifiedterms> and <autn:modifiedweights>	one	No
<autn:exclusions>	one	No
<autn:inclusions>	one	No
<autn:fakeweights>	one	No
<autn:numresults>	one	No
<autn:threshold>	one	No
<autn:databases>	one	No
<autn:fieldtext>	one	No
<autn:taxonomyroot>	one	No
<autn:active>	one	No
<autn:role>	one	No
<autn:memberpermissions>	one	No
<autn:nonmemberpermissions>	one	No
<autn:simplecatdefaultcat>	one	No
<autn:relevantcat>	one	No
<autn:simplecatparam>	one	No

Tag name	Number allowed	Required
<autn:userfields>	one	No

<autn:generatedterms>

The `<autn:generatedterms>` tag sets terms generated for a category from training. Do this only if you are editing an existing category from which you can take the terms. You can include one `<autn:generatedterms>` in each set of `<autn:details>` tags. For example:

`<autn:generatedterms>LYMPH,MISDIAGNOS,PATHOLOGI</autn:generatedterms>`

> **NOTE:** If you specify terms for a category in the `<autn:generatedterms>` tag, you must enter a corresponding list of weights by using the `<autn:generatedweights>` tag.

The `<autn:generatedterms>` tag is required if you set `<autn:queryagenttnw>`. For more information, see <autn:queryagenttnw>, below.

<autn:generatedweights>

The `<autn:generatedweights>` tag sets weights generated for the terms for the category from training. Do this only if you are editing an existing category from which you can take the weights. You can include one `<autn:generatedweights>` in each set of `<autn:details>` tags. For example:

`<autn:generatedweights>5960,4035,4001</autn:generatedweights>`

> **NOTE:** If you specify weights for a category in the `<autn:generatedweights>` tag, you must enter a corresponding list of terms by using the `<autn:generatedterms>` tag.

The `<autn:generatedweights>` tag is required if you set `<autn:queryagenttnw>`. For more information, see <autn:queryagenttnw>, below.

<autn:queryagenttnw>

The `<autn:queryagenttnw>` tag specifies the query string generated by terms and weights used to build the category. For example:

`<autn:queryagenttnw>LYMPH~[5960] MISDIAGNOS~[4305] PATHOLOGI~[4001]</autn:queryagenttnw>`

> **NOTE:** You can use `<autn:queryagenttnw>` with either `<autn:generatedterms>` and `<autn:generatedweights>`, or `<autn:modifiedterms>` and `<autn:modifiedweights>`. If both sets exist, modified terms take precedence over generated ones.

The `<autn:queryagenttnw>` tag performs the same function as the IDOL `CategoryBuild` action. See Build Categories, on page 165.

For more information, see <autn:generatedterms>, above, <autn:generatedweights>, above, <autn:modifiedterms>, below, and <autn:modifiedweights>, on the next page.

<autn:modifiedterms>

The `<autn:modifiedterms>` tag sets terms defined by the user. You can include one `<autn:modifiedterms>` within each set of `<autn:details>` tags. For example:

`<autn:modifiedterms>LYMPH,MISDIAGNOS,PATHOLOGI</autn:modifiedterms>`

For more information on changing the weights of terms in a category, see Change Category Term Weights, on page 164.

> **NOTE:** If you specify terms for a category in the `<autn:modifiedterms>` tag, you must enter a corresponding list of weights by using the `<autn:modifiedweights>` tag.

<autn:modifiedweights>

The `<autn:modifiedweights>` tag sets weights for terms defined by the user. You can include one `<autn:modifiedweights>` within each set of `<autn:details>` tags. For example:

`<autn:modifiedweights>5960,4035,4001</autn:modifiedweights>`

For more information on changing the weights of terms in a category, see Change Category Term Weights, on page 164.

> **NOTE:** If you specify weights for a category in the `<autn:modifiedweights>` tag, you must enter a corresponding list of terms by using the `<autn:modifiedterms>` tag.

<autn:exclusions>

A comma-separated list of documents to be excluded from category queries. For example:

`<autn:exclusions>C:\temp\doc1.txt,C:\temp\doc2.txt</autn:exclusions>`

For more information, see the `CategorySetSpecialDocs Exclusions` parameter in the *IDOL Server Reference*.

<autn:inclusions>

The `<autn:inclusions>` tag contains a comma-separated list of documents to be included in category queries. For example:

`<autn:inclusions>C:\temp\docA.txt,C:\temp\docB.txt</autn:inclusions>`

For more information, see the `CategorySetSpecialDocs Inclusions` parameter in the *IDOL Server Reference*.

> **NOTE:** If you use this tag, you must set corresponding weights for the included terms with the `<autn:fakeweights>` tag.

<autn:fakeweights>

The `<autn:fakeweights>` tag contains a comma-separated list of weights for documents specified in `<autn:inclusions>`. Weights must correspond to the documents in number and order. For example:

`<autn:fakeweights>800,2200</autn:fakeweights>`

For more information, see the `CategorySetSpecialDocs Fakeweights` parameter in the *IDOL Server Reference*.

<autn:numresults>

The `<autn:numresults>` tag sets the maximum number of results that category queries can return. You can include one `<autn:numresults>` tag for each category within `<autn:details>` tags. For example:

`<autn:numresults>10</autn:numresults>`

<autn:threshold>

The `<autn:threshold>` tag sets the minimum relevance score that documents must possess to appear in category query results. You can include one `<autn:threshold>` tag for each category within `<autn:details>` tags. For example:

`<autn:threshold>400</autn:threshold>`

<autn:databases>

The `<autn:databases>` tag sets the databases in which documents must exist to appear in category query results. You must separate multiple databases with plus symbols, commas, or spaces. For example:

`<autn:databases>Archive,Minicar</autn:databases>`

<autn:fieldtext>

The `<autn:fieldtext>` tag specifies the fields that result documents must contain, and the conditions that these fields have to meet for the documents to be returned as results.

For more information, see the `CategorySetFields Fieldtext` parameter in the *IDOL Server Reference*.

<autn:taxonomyroot>

The `<autn:taxonomyroot>` tag specifies whether the category is a taxonomy root. For example:

`<autn:taxonomyroot>True</autn:taxonomyroot>`

<autn:active>

The `<autn:active>` tag specifies whether the category is active. For example:

`<autn:active>True</autn:active>`

<autn:role>

The `<autn:role>` tag specifies the role or roles that you want to give access to the category. For example:

`<autn:role>Usertype1,Usertype2</autn:role>`

Use <autn:memberpermissions> and <autn:nonmemberpermissions> to set which category access permissions members and non-members of this role should have.

<autn:memberpermissions>

The `<autn:memberpermissions>` tag sets the category access permissions that you want role members to have. If you want to list multiple permissions, you must separate them with commas. For example:

`<autn:memberpermissions>read,edit</autn:memberpermissions>`

For more information, see the `CategorySetPermissions MemberPermissions` parameter in the *IDOL Server Reference*.

<autn:nonmemberpermissions>

The `<autn:nonmemberpermissions>` tag sets the category access permissions that you want role non-members to have. If you want to list multiple permissions, you must separate them with commas. For example:

```
<autn:nonmemberpermissions>read</autn:nonmemberpermissions>
```

For more information, refer to the `NonMemberPermissions` parameter for the `CategorySetPermissions` action in the *IDOL Server Reference*.

<autn:simplecatdefaultcat>

The `<autn:simplecatdefaultcat>` tag specifies which of the category's children is to be the default category for `CategorySimpleCategorize`. You must use the category ID value. For example:

```
<autn:simplecatdefaultcat>1230982349874568</autn:simplecatdefaultcat>
```

For more information, see the `CategorySetDetails SimpleCatDefaultCat` parameter in the *IDOL Server Reference*.

<autn:relevantcat>

The `<autn:relevantcat>` tag specifies which of the category's children is to be used as the relevant category. You must use the category ID value. For example:

```
<autn:relevantcat>324987602</autn:relevantcat>
```

For more information, see the `CategorySetDetails RelevantCat` parameter in the *IDOL Server Reference*.

<autn:simplecatparam>

The `<autn:simplecatparam>` tag sets a numeric factor that increases or decreases the probability of this category being chosen by the `CategorySimpleCategorize` action. For example:

```
<autn:simplecatparam>1.4</autn:simplecatparam>
```

For more information, see the `CategorySetDetails SimpleCatParam` parameter in the *IDOL Server Reference*.

<autn:userfields>

The `<autn:userfields>` tag sets fields and values defined by the user, as in this example:

```
<autn:userfields>
<autn:acc-inbox-threshold>30</autn:acc-inbox-threshold>
</autn:userfields>
```

Example Category XML Files

The following is an example XML file that includes only required elements:

```
<?xml version="1.0" encoding="UTF-8" ?>
<autn:categories xmlns:autn="http://schemas.autonomy.com/aci/">
   <autn:category>
      <autn:name>MyCategory</autn:name>
   </autn:category>
</autn:categories>
```

The following is an example of a Category XML file that includes one category:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<autn:categories xmlns:autn="http://schemas.autonomy.com/aci/">
    <autn:category>
    <autn:name>test</autn:name>
        <autn:trainingelement>
            <autn:type>TRAININGTEXT</autn:type>
                <autn:content>hotels paris food stay</autn:content>
                <autn:language>ENGLISH</autn:language>
        </autn:trainingelement>
        <autn:details>
            <autn:generatedterms>STAI,PARI,FOOD,HOTEL</autn:generatedterms>
            <autn:generatedweights>2352,2272,1884,417</autn:generatedweights>
            <autn:queryagenttnw>FOOD~[1884] HOTEL~[417] PARI~[2272]STAI~[2352]
            </autn:queryagenttnw>
            <autn:active>True</autn:active>
        </autn:details>
    </autn:category>
</autn:categories>
```

Appendix G: GetStatus Action Response

This appendix describes the `GetStatus` action and the response that IDOL Server returns.

GetStatus Action

The `GetStatus` action returns information about the status of IDOL Server and its components, as well as the configuration and content of the server.

`http://IDOLhost:port/action=GetStatus`

where:

`IDOLhost`	is the IP address or name of the machine on which IDOL Server is installed.
`port`	is the ACI port by which you send actions to IDOL Server (set by the `Port` parameter in the `[Server]` section of the IDOL Server configuration file).

The `GetStatus` action is a diagnostic tool that you can use to check information about your server.

You can use the output from `GetStatus` to:

- view some of the IDOL Server configuration options without opening the configuration file.
- check that all IDOL Server components are running.
- monitor the number of documents in IDOL Server and see how close the server is to the configured document limit.
- monitor how many index actions IDOL Server has processed, and check the size of the index queue, for example to:
  - determine whether the server processes incoming index actions as fast as it receives them.
  - determine how many items the server has left to process, before performing maintenance or backups.
- check the results of the latest index validation.
- check language configuration and determine how many documents in the server use each configured language type.

- check document security configuration and determine how many documents in the server belong to each configured security type.

- check user security configuration.

> **TIP:** You can also view status information by using the IDOL Admin interface. For more information, refer to the *IDOL Admin User Guide*.

> **NOTE:** Micro Focus recommends that you contact Micro Focus Big Data Support before taking actions based on the information in the `GetStatus` action.

Related Topics

IDOL Server GetStatus Response

This section describes the XML tags that return in the response to a `GetStatus` action sent to the IDOL Proxy component in a unified IDOL Server configuration that does not use a DAH or DIH.

For details about unified IDOL Server configurations, refer to the *IDOL Getting Started Guide*.

The `GetStatus` response from IDOL Proxy contains information from all its child components. Most tags result from the `GetStatus` response of the IDOL Server child components. However, the unified IDOL Server does not display all tags from child servers, and IDOL Proxy returns additional tags that none of the components return (such as, component status).

Tag	Description	Related configuration parameters
`product`	The component that returns the `GetStatus` data.	
`version`	The version of the component. **NOTE:** In the IDOL Server `GetStatus` response, this version is the version of the IDOL Proxy component.	
`indexport`	The port used to send index actions to IDOL Server.	`IndexPort`
`aciport`	The port used to send ACI actions to IDOL Server.	`Port`
`serviceport`	The port used to send service actions to IDOL Server.	`ServicePort`
`directory`	The directory that contains IDOL Server.	

Tag	Description	Related configuration parameters
acithreads	The number of allowed ACI threads.	Threads
component	The status of each child component. This section contains a subsection for each component.	
status	The status of the component.	
aciport	The port used to send ACI actions to the component.	
indexport	The port used to send index actions to the component. NOTE: This port applies only to indexing components, such as Content.	
serviceport	The port used to send service actions to the component.	
licensed_languages	A comma-separated list of the languages that your license allows this IDOL Server to use.	
termsperdoc	The configured number of best terms to generate for each document.	TermsPerDoc
suggestterms	The configured number of best terms to use in a Suggest action.	SuggestTerms
documents	The number of documents in IDOL Server that are available for search.	
document_sections	The number of document sections in IDOL Server.	
committed_documents	The number of documents that have been indexed into IDOL Server, including any deleted documents that have not been removed in a compaction.	
deleted_sections	The number of document sections for deleted sections in IDOL Server. This number represents the number of deleted sections since the last compaction operation. You can use it to check whether you need to compact the index to permanently delete these documents.	

Tag	Description	Related configuration parameters
indexed_data_kb	The amount of indexed data in the Content component. **NOTE:** This number does not specify the amount of disk space that IDOL Server is using. This value specifies the index size for licensing purposes.	
full	Whether the IDOL Server data index has reached the maximum size specified in the MaxDocumentCount or MaxIndexSizeKb configuration parameter. The DIH uses this tag when distributing index actions to IDOL Servers. You can configure DIH to stop indexing to full servers by using the RespectChidlFullness DIH configuration parameter.	MaxDocumentCount MaxIndexSizeKb
full_ratio	How close the IDOL Server data index is to reaching the configured maximum number of documents, as determined by the MaxDocumentCount configuration parameter. If you have not set MaxDocumentCount, IDOL Server uses the MaxIndexSizeKb limit instead.	MaxDocumentCount MaxIndexSizeKb
terms	The number of terms in IDOL Server for which you can search.	
total_terms	The total number of terms in IDOL Server, including internal terms.	
term_hashes	The nearest power of 2 above the value of the DiskHash parameter.	DiskHash
record_size	The maximum size of terms in IDOL Server.	TermSize
max_occurrences	The highest number of documents in which any single term occurs.	
mindate	The earliest date of any document in IDOL Server (in AUTNDATE format).	
maxdate	The latest date of any document in	

Tag	Description	Related configuration parameters
	IDOL Server (in AUTNDATE format).	
ref_fields	The number of reference fields that the documents in IDOL Server use.	
ref_hashes	The configured value of the RefHashes parameter.	RefHashes
indexqueue	Details about the IDOL Server index queue. See Check Index Status, on page 73.	
indexqueuereceived	The number of index actions that IDOL Server has received.	
indexqueuecompleted	The number of index actions that IDOL Server has completed.	
indexqueuequeued	The number of index actions remaining in the index queue.	
initialid	The initial ID of the last DREINITIAL index action that the server processed. You set this ID when you send a DREINITIAL index action, by using the InitialID parameter. See Initialize the Data Index, on page 442.	
termcache	Details about the IDOL Server term cache. See [TermCache] Section, on page 503.	
used_kb	The current size of the IDOL Server term cache (in KB).	
num_terms	The number of terms currently stored in the IDOL Server term cache.	
limit_kb	The maximum configured size of the IDOL Server term cache (in KB).	
requests	The number of terms that have been requested from the cache.	
hits	The number of matches for terms in the cache that IDOL Server has received.	
hitrate	The rate at which IDOL Server has matched terms in the cache.	

Tag	Description	Related configuration parameters
`indexcache`	Details about the IDOL Server index cache. See [IndexCache] Section, on page 491.	
`used_kb`	The current size of the IDOL Server index cache (in KB).	
`num_terms`	The number of terms currently stored in the IDOL Server index cache.	
`limit_kb`	The maximum configured size of the IDOL Server index cache (in KB).	`IndexCacheMaxSize`
`num_blocks`	The number of memory blocks allocated for IDOL Server indexing.	
`fieldcodes`	Details of fields in IDOL Server documents.	
`base`	The number of distinct field names in IDOL Server.	
`total`	The total number of field codes. If you have set the `XMLFullStructure` configuration parameter to **True**, this value is the total number of distinct occurrences of fields.	`XMLFullStructure`
`databases`	Details of the databases in IDOL Server. See Create and Delete Databases, on page 422.	
`max_databases`	The maximum allowed number of databases.	
`num_databases`	The total number of databases in IDOL Server.	`NumDBs`
`active_databases`	The number of active (not deleted) databases.	
`database`	Details of individual IDOL Server databases.	
`name`	The name of the database.	`Name`
`documents`	The number of documents in the database.	
`sections`	The number of document sections in the	

Tag	Description	Related configuration parameters
	database.	
internal	Whether the database is configured as internal.	`Internal`
readonly	Whether the database is configured as read-only.	`ReadOnly`
expiry_hours	The expiration time (in hours) for documents in this database.	`ExpiryTime`
expiry_action	The expiration action to perform when documents expire from this database.	`ExpireIntoDatabase`
security_settings	Details of the configured security types in IDOL Server. See Set up Security, on page 387.	
no_of_security_types	The total number of configured security types in IDOL Server.	
security_type	Details of individual configured document security types. See [Security] Section, on page 499.	`[Security]` section
name	The name of the security type. This value is the name of the configuration section where you define the security settings.	
documents	The number of documents that this security type applies to.	
sections	The number of document sections that this security type applies to.	
language_type_settings	Details of the configured language types in IDOL Server. See Language Support, on page 109.	
no_of_language_types	The number of configured language types in IDOL Server.	
language_type	Details of an individual language type. See [LanguageTypes] Section, on page 492.	`[LanguageTypes]` section
name	The name of the language type. This value is the name given to one encoding	`Encodings`

Tag	Description	Related configuration parameters
	for a language, set in the `Encodings` configuration parameter.	
language	The language that applies to this language type. This value is the name of the language configuration section for this language type.	
encoding	The encoding that applies to this language type.	`Encodings`
documents	The number of documents with this language type.	
sections	The number of document sections with this language type.	
Validation	The results of index validation.	

result	The result of validation on a particular index. There is a result tag for each type of index validation. This tag contains one of the following values:
	
	• **N/A**. No index validation has run for the index type.
	
	• **Success**. Index validation has passed.
	
	• An error message. Indicates a validation failure. In this case you might need to regenerate a data index.
		`ValidateDiskIndex`
		`ValidateNodeTable`
		`ValidateRefIndex`
		`ValidateUnstemmed`
		`ValidateSecIndex`
		`ValidateNumeric`

type	The type of index for the result. This can be one of the following values:
	
	• `diskindex`
	
	• `nodetable`
	
	• `refindex`
	
	• `unstemmed`
	
	• `secindex`
	
	• `numeric`
		`ValidateDiskIndex`
		`ValidateNodeTable`
		`ValidateRefIndex`
		`ValidateUnstemmed`
		`ValidateSecIndex`
		`ValidateNumeric`

timestamp	The time of the last validation, in epoch seconds.	

Tag	Description	Related configuration parameters
duration	The duration (in seconds) of the last validation operation.	
autn:numusers	The number of users in IDOL Server.	
autn:maxusers	The maximum number of users allowed in IDOL Server. This value depends on your product license.	
securitytypes	User security information from the Community component.	[UserSecurity] section
defaultsecuritytype	The default user security type.	DefaultSecurityType
name	The name of the user security type.	
fields	The user security fields associated with the security type.	SecurityFieldCSVs
documentsecurity	Whether the security type uses the document security module.	DocumentSecurity
documentsecuritytype	The document security module that applies to the user security type.	DocumentSecurityType
autn:scheduledthreads	The number of threads available for scheduled tasks.	
autn:categories	The number of categories in IDOL Server.	

Example IDOL Server GetStatus Response

The following XML is an example response to the GetStatus action.

```
<autnresponse>
    <action>GETSTATUS</action>
    <response>SUCCESS</response>
    <responsedata>
        <product>IDOL Server</product>
        <version>10.4.0</version>
        <build>1076091</build>
        <indexport>9001</indexport>
        <queryport>-1</queryport>
        <aciport>9000</aciport>
        <serviceport>9002</serviceport>
        <directory>C:\Program Files\IDOL\IDOLServer\IDOL</directory>
        <acithreads>4</acithreads>
```

```
<component>
    <content>
        <status>RUNNING</status>
        <aciport>9010</aciport>
        <indexport>9011</indexport>
        <serviceport>9012</serviceport>
    </content>
    <community>
        <status>RUNNING</status>
        <aciport>9030</aciport>
        <serviceport>9031</serviceport>
    </community>
    <category>
        <status>RUNNING</status>
        <aciport>9020</aciport>
        <serviceport>9021</serviceport>
    </category>
    <agentstore>
        <status>RUNNING</status>
        <aciport>9050</aciport>
        <indexport>9051</indexport>
        <queryport>9052</queryport>
        <serviceport>9053</serviceport>
    </agentstore>
    <view>
        <status>RUNNING</status>
        <aciport>9080</aciport>
        <serviceport>9081</serviceport>
    </view>
</component>
<licensed_languages>UNLIMITED</licensed_languages>
<querythreads>0</querythreads>
<termsperdoc>50</termsperdoc>
<suggestterms>40</suggestterms>
<documents>1086404</documents>
<document_sections>1368042</document_sections>
<committed_documents>1370830</committed_documents>
<deleted_sections>0</deleted_sections>
<indexed_data_kb>82643321</indexed_data_kb>
<full>false</full>
<full_ratio>0.69</full_ratio>
<terms>3675990</terms>
<total_terms>3872552</total_terms>
<term_hashes>16777216</term_hashes>
<record_size>53</record_size>
<max_occurrences>611200</max_occurrences>
<mindate>1059260400</mindate>
<maxdate>1437484356</maxdate>
<mindatestring>23:00:00 26/07/2003</mindatestring>
```

```
<maxdatestring>13:12:36 21/07/2015</maxdatestring>
<ref_fields>1</ref_fields>
<ref_hashes>10000000</ref_hashes>
<indexqueue>
    <indexqueuereceived>78</indexqueuereceived>
    <indexqueuecompleted>73</indexqueuecompleted>
    <indexqueuequeued>5</indexqueuequeued>
    <initialid>0</initialid>
</indexqueue>
<termcache>
    <used_kb>3867</used_kb>
    <num_terms>29</num_terms>
    <limit_kb>102400</limit_kb>
    <requests>41</requests>
    <hits>2</hits>
    <hitrate>4</hitrate>
</termcache>
<indexcache>
    <used_kb>6303</used_kb>
    <num_terms>9873</num_terms>
    <limit_kb>102400</limit_kb>
    <num_blocks>1</num_blocks>
</indexcache>
<fieldcodes>
    <base>0</base>
    <total>77</total>
</fieldcodes>
<databases>
    <max_databases>65534</max_databases>
    <num_databases>3</num_databases>
    <active_databases>3</active_databases>
    <database>
        <name>News</name>
        <documents>1085127</documents>
        <sections>1366685</sections>
        <internal>false</internal>
        <readonly>false</readonly>
        <expiry_hours> 4320 </expiry_hours>
        <expiry_action> move to Archive </expiry_action>
    </database>
    <database>
        <name>Archive</name>
        <documents>1006</documents>
        <sections>1086</sections>
        <internal>false</internal>
        <readonly>true</readonly>
        <expiry_hours>No Information Available</expiry_hours>
        <expiry_action>No Information Available</expiry_action>
    </database>
```

```
        <database>
            <name>World</name>
            <documents>262</documents>
            <sections>262</sections>
            <internal>false</internal>
            <readonly>false</readonly>
            <expiry_hours>No Information Available</expiry_hours>
            <expiry_action>No Information Available</expiry_action>
        </database>
    </databases>
    <security_settings>
        <no_of_security_types>5</no_of_security_types>
        <security_type>
            <name>NT_V4</name>
            <documents>0</documents>
            <sections>0</sections>
        </security_type>
        <security_type>
            <name>Netware_V4</name>
            <documents>0</documents>
            <sections>0</sections>
        </security_type>
        <security_type>
            <name>Notes_V4</name>
            <documents>0</documents>
            <sections>0</sections>
        </security_type>
        <security_type>
            <name>Exchange_V4</name>
            <documents>0</documents>
            <sections>0</sections>
        </security_type>
        <security_type>
            <name>Documentum_V4</name>
            <documents>0</documents>
            <sections>0</sections>
        </security_type>
    </security_settings>
    <language_type_settings>
        <no_of_language_types>1</no_of_language_types>
        <language_type>
            <name>englishUTF8</name>
            <language>ENGLISH</language>
            <encoding>UTF8</encoding>
            <documents>1086404</documents>
            <sections>1368042</sections>
        </language_type>
    </language_type_settings>
    <validation>
```

```
        <result type="diskindex">N/A</result>
        <result type="nodetable">N/A</result>
        <result type="refindex">N/A</result>
        <result type="unstemmed">N/A</result>
        <result type="secindex">N/A</result>
        <result type="numeric">N/A</result>
    </validation>
    <autn:numusers>25</autn:numusers>
    <autn:cachedusers>0</autn:cachedusers>
    <autn:usersize>0</autn:usersize>
    <autn:maxusers>5000</autn:maxusers>
    <autn:numterms>0</autn:numterms>
    <autn:termsize>0</autn:termsize>
    <autn:maxterms>0</autn:maxterms>
    <autn:termoffset>0</autn:termoffset>
    <autn:storeversion>0</autn:storeversion>
    <securitytypes>
        <defaultsecuritytype>autonomy</defaultsecuritytype>
        <securitytype>
            <name>autonomy</name>
            <documentsecurity>false</documentsecurity>
        </securitytype>
        <securitytype>
            <name>notes</name>
            <fields>username</fields>
            <documentsecurity>true</documentsecurity>
            <documentsecuritytype>Notes_V4</documentsecuritytype>
        </securitytype>
        <securitytype>
            <name>ldap</name>
            <documentsecurity>false</documentsecurity>
        </securitytype>
        <securitytype>
            <name>documentum</name>
            <fields>username</fields>
            <documentsecurity>true</documentsecurity>
            <documentsecuritytype>Documentum_V4</documentsecuritytype>
        </securitytype>
        <securitytype>
            <name>exchange</name>
            <fields>username,domain</fields>
            <documentsecurity>true</documentsecurity>
            <documentsecuritytype>Exchange_V4</documentsecuritytype>
        </securitytype>
        <securitytype>
            <name>netware</name>
            <fields>username</fields>
            <documentsecurity>true</documentsecurity>
            <documentsecuritytype>Netware_V4</documentsecuritytype>
```

```
            </securitytype>
        </securitytypes>
        <autn:schedulethreads>2</autn:schedulethreads>
        <autn:categories>5953</autn:categories>
    </responsedata>
</autnresponse>
```

Appendix H: Error Codes

This appendix lists IDOL error codes.

Community Error Codes

The following table describes common error codes for the IDOL Community component.

Error code	Description
–2147483391	Error: Bad Parameter
–2147483390	Error: Out Of Memory
–2147483388	Error: File Error
–2147438079	Error: User Exists
–2147438053	Error: User Not Found
–2147438078	Error: Too Many Users
–2147438077	Error: Agent Exists
–2147438076	Error: Agent Not Found
–2147438075	Error: Data Index Error, Data Index not found
–2147438074	Error: Agent Index Error
–2147438073	Error: Too Many Agents
–2147438072	Error: Field Not Found
–2147438071	Error: Role Exists
–2147438070	Error: Role Not Found
–2147438069	Error: Privilege Exists
–2147438068	Error: Privilege Not Found
–2147438067	Error: Profile Not Found
–2147438065	Error: Numeric terms must be specified alone
–2147438064	Error: Data Index Error, no terms found

Glossary

A

Access Control List.
See ACL.

ACI (Autonomy Content Infrastructure)
A technology layer that automates operations on unstructured information for cross-enterprise applications. ACI enables an automated and compatible business-to-business, peer-to-peer infrastructure. The ACI allows enterprise applications to understand and process content that exists in unstructured formats, such as email, Web pages, Microsoft Office documents, and IBM Notes.

ACI Server
A server component that runs on the Autonomy Content Infrastructure (ACI).

ACL (access control list)
An ACL is metadata associated with a document that defines which users and groups are permitted to access the document.

action
A request sent to an ACI server.

active directory
A domain controller for the Microsoft Windows operating system, which uses LDAP to authenticate users and computers on a network.

Adaptive Probabilistic Concept Modelling
See APCM.

agent
A process that searches for information about a specific topic. An administrator can create agents for users or allow users to create their own agents. See Also: explicit agent, implicit agent

agent index
An index in IDOL Server that stores agents and profiles.

AgentBoolean field
An IDOL Server field that stores Boolean agents (Boolean or proximity expressions that legacy technologies use to categorize documents). You can then query IDOL Server with text and an AgentBoolean field to return categories whose Boolean agent matches this text.

ALD
Automatic Language Detection. The process of automatically detecting the language of a particular document, and indexing it into IDOL Server according to the rules for the detected language.

alerting
An automatic process for alerting users, by e-mail, text, or message, when new content is added to IDOL Server that matches their agents or profiles. See Also: mailing.

APCM
Adaptive Probabilistic Concept Modelling. A technique whereby terms are given a weight according to their statistical importance in IDOL Server. Terms can have a weight between 0 and 255.

AQG
Automatic Query Guidance. A set of operations that use the results from query summaries. AQG includes dynamic thesaurus generation, automatic query disambiguation, query refinement, and rapid clustering of a results set. See Also: query summary.

authentication

The process of checking user credentials (user names, passwords, and PIN codes) against an IDOL Server or external security repository. The authentication process identifies a user, and allows IDOL Server to confirm their access permissions for different documents.

autnrank

An internal IDOL Server document rank, which determines the order in which two or more documents return in a results list when the relevance or other sort option is equal.

Automatic Language Detection

See ALD.

Automatic Query Guidance

See AQG.

Autonomy Content Infrastructure

See ACI.

C

category

A set of criteria that define a particular topic, which you can use to categorize documents that contain content relevant to the topic.

Category component

The IDOL Server component that manages categorization and clustering.

Category index

An IDOL Server index that stores categories.

category training

Text or documents that define a topic or subject for a particular category. When IDOL Server categorizes documents, it matches document content to similar category training.

cluster

A hierarchically agglomerated collection of data that has been extracted from snapshots. Each cluster represents a concept area that contains a set of items, which share common properties. Clustering data allows you to make trends and developments in data visible.

combine

A query operation that combines two or more query results into a specified smaller number of results. The most usual case is to combine two or more sections of the same document as a single query result. It can also combine results by a reference or metadata field value.

community

All the people in a user network neighborhood. It allows users to find other people in the community who have been looking at similar documents, or have agents that are similar to their agents.

Community component

The IDOL Server component that manages users and communities.

concept summary

A brief summary of each result document that returns for a query. The concept summary displays a few sentences that are typical of the result content (these sentences can be from different parts of the result document).

conceptual search

A type of query that allows you to search for documents that match the concept that your query text defines, rather than matching the particular keywords in your text. See Also: query.

connector

An IDOL component (for example File System Connector) that retrieves

information from a local or remote repository (for example, a file system, database, or Web site).

Connector Framework Server (CFS)

Connector Framework Server processes the information that is retrieved by connectors. Connector Framework Server uses KeyView to extract document content and metadata from over 1,000 different file types. When the information has been processed, it is sent to an IDOL Server or Distributed Index Handler (DIH).

Content component

The IDOL Server component that manages the data index and performs most of the search and retrieval operations from the index.

context summary

A conceptual summary of the result document that is biased by the terms of the query. A context summary includes sentences that are particularly relevant to the terms in the query (these sentences can be from different parts of the result document).

crawling

The process that Connectors and Web crawlers use to retrieve content from Web resources, by recursively following hyperlinks from an initial page. See Also: spidering

D

DAH (Distributed Action Handler)

DAH distributes actions to multiple copies of IDOL Server or a component. It allows you to use failover, load balancing, or distributed content.

data index

An IDOL Server index that stores content data. You can customize how to store data in the data index by configuring appropriate settings in the IDOL Server configuration file.

database

An IDOL Server data pool that stores indexed information. The administrator can set up one or more databases, and specifies how data is fed to the databases. By default, IDOL Server contains the databases Profile, Agent, Activated, Deactivated, News, and Archive.

default user

The default user role in IDOL Server. A default user has only the privileges that have been allocated to this default role.

DIH (Distributed Index Handler)

DIH allows you to efficiently split and index extremely large quantities of data into multiple copies of IDOL Server or the Content component. DIH allows you to create a scalable solution that delivers high performance and high availability. It provides a flexible way to batch, route, and categorize the indexing of internal and external content into IDOL Server.

Distributed Action Handler

See DAH.

Distributed Index Handler

See DIH.

E

Eduction

The process of extracting entities (patterns of text) from documents.

entity

In Eduction, an entity is a word, phrase, or block of information that the Eduction component can match and extract from documents. An entity can be a specific text string, such as a name, or it can be a pattern of text such as an address or phone number. You define the pattern in a grammar, which Eduction uses to find the entities in documents.

expertise location

An IDOL Server operation to find groups of users with a particular set of expertise or interests.

explicit agent

An IDOL agent that users explicitly create from themselves. See Also: agent, implicit agent

Extensible Markup Language

See XML.

extraction

The process of extracting text, metadata, and subfiles from documents. IDOL KeyView performs this extraction process in IDOL.

F

faceted search

See: parametric search

fetch

The process of downloading documents from the repository in which they are stored (such as a local folder, Web site, Lotus Domino server, and so on), importing them to IDX format, and indexing them into IDOL Server.

fetch task

A group of settings that instruct a Connector how to retrieve data from a repository.

Connectors can run fetch tasks automatically, or in response to an action.

field

Fields define different parts of content in IDOL documents, such as the title, content, and metadata information.

field operator

A syntax string that defines a matching criteria in FieldText.

FieldText

A type of query that searches for particular content in a particular document field. See Also: query, field

G

grammar

In Eduction, a grammar is a pattern that defines an entity. See Also: Eduction, entity

H

hyperlink

The ability for IDOL Server to connect related documents to results, by using suggestions. See Also: suggest

I

identifier

An encoded value that identifies the source of a document in IDOL Server. Connectors and CFS add identifiers to every document that they create for indexing into IDOL Server. They store this value in the AUTN_IDENTIFIER field.

IDOL

The Intelligent Data Operating Layer (IDOL) Server, which integrates unstructured, semi-structured and structured information

from multiple repositories through an understanding of the content. It delivers a real-time environment in which operations across applications and content are automated.

IDOL Proxy component

An IDOL Server component that accepts incoming actions and distributes them to the appropriate subcomponent. IDOL Proxy also performs some maintenance operations to make sure that the subcomponents are running, and to start and stop them when necessary.

IDOL Search Optimizer

An IDOL application that allows you to manage IDOL Server content.

IDX

A structured file format that can be indexed into IDOL Server. You can use a connector to import files into this format, or you can manually create IDX files.

implicit agent

An IDOL agent that is created as part of a user profile. When you profile a user, IDOL Server creates these agents for a user, according to the documents and search results that the user views. See Also: agent, explicit agent

importing

After a document has been downloaded from the repository in which it is stored, it is imported to an IDX or XML file format. This process is called importing.

index

The IDOL Server data index contains document content and field information for analysis and retrieval.

index action

An IDOL Server command to index data, or to maintain and manipulate the data index.

index fields

Fields that IDOL Server processes linguistically when it stores them. Store fields that contain text that you want to query frequently as Index fields. IDOL Server applies stemming and stop word lists to text in Index fields before it stores them, which allows IDOL Server to process queries for these fields more quickly. Typically DRETITLE and DRECONTENT are fields that are set up as Index fields.

indexing

The process of storing data in IDOL Server. IDOL Server stores data in different field types (such as index, numeric, and ordinary fields). It is important to store data in appropriate field types to ensure optimized performance.

Intellectual Asset Protection System (IAS)

An integrated security solution to protect your data. At the front end, authentication checks that users are allowed to access the system that contains the result data. At the back end, entitlement checking and authentication combine to ensure that query results contain only documents that the user is allowed to see, from repositories that the user has permission to access. For more information, refer to the IDOL Document Security Administration Guide.

Intelligent Query Logic.

See IQL.

IQL

Intelligent Query Logic. Functionality that allows you to set up rules to return a particular set of documents, or to run a secondary query in response to an initial keyword or conceptual query.

K

KeyView

The IDOL component that extracts data, including text, metadata, and subfiles from over 1,000 different file types. KeyView can also convert documents to HTML format for viewing in a Web browser.

L

LDAP

Lightweight Directory Access Protocol. Applications can use LDAP to retrieve information from a server. LDAP is used for directory services (such as corporate email and telephone directories) and user authentication. See also: active directory, primary domain controller.

License Server

License Server enables you to license and run multiple IDOL solutions. You must have a License Server on a machine with a known, static IP address.

link term

Also referred to as "links". Terms in query text that are also contained in the result documents that IDOL Server returns for this query.

Lua

An embedded scripting language that you can use to write custom scripts to expand certain IDOL functionality.

M

mailing

An automatic process for sending an email to users when new content is added to IDOL Server that matches their agents or profiles. See Also: alerting

mapped security

A security setup where IDOL Connectors index documents into IDOL Server with an encrypted access control list, which IDOL Server uses to match user permissions for the document. With this method, IDOL Server does not need to check the original data repository to check the security information every time a user attempts to access the document. Compare With: unmapped security. See Also: access control list.

meaning based computing

The ability for computers to act on the meaning of content. This includes conceptual searching, and also workflows that automatically process documents according to their content.

O

OmniGroupServer (OGS)

A server that manages access permissions for your users. It communicates with your repositories and IDOL Server to apply access permissions to documents.

P

parametric search

A type of query that returns a list of all possible values of a specified field for documents that match a particular standard query. You can use the values to find matching documents with a particular property. This process is also known as faceted search. Compare With: FieldText

PIN code

Personal Identification Number security feature used in addition to a user ID and password.

primary domain controller

A server computer in a Microsoft Windows domain that controls various computer resources. See also: active directory, LDAP.

privilege

Role-based capabilities that determine, for example, whether a user is allowed to access specific data.

profile

Information about a user that is based on the concepts in documents that the user reads. Every time a user opens a document, IDOL Server updates their profile. This process allows the administrator to bring new documents that match the interests in a user profile to the attention of the users.

promotions

Targeted content that you want to display to users but that is not included in the search results, such as advertisements.

Q

query

A string that you submit to IDOL Server, which analyzes the concept of the query and returns documents that are conceptually similar to it. You can submit queries to IDOL Server to perform several kinds of search, such as natural language, Boolean, bracketed Boolean, and keyword.

query summary

A query operation that determines the important topics and phrases in a set of documents. Query summaries are used in Automatic Query Guidance (AQG). See Also: AQG.

quick summary

A brief summary of each result document that returns for a query. The quick summary

displays the first few sentences of the result document.

R

reference

A string that is used to identify a document. This might be a title or a URL, and allows IDOL to identify documents for retrieval, indexing, and deduplication.

ReferenceType field

Fields used to identify documents. At index time IDOL Server can use ReferenceType fields to eliminate duplicate copies of documents. It uses them at query time to filter results.

relevance

The similarity that a particular query result has to the initial query. IDOL Server assigns results a percentage relevance score according to how closely it matches the query criteria.

retrain

The process used to increase the accuracy of agents by indicating which of the results that return to you are most relevant to your query. The retrained agent then returns more relevant results.

role

A set of privileges that an administrator can allocate to an IDOL Server user.

S

section breaking

The process of separating a document into sections for indexing. The number of sections that a document is split into increases proportionally with the size of the document. This process ensures that when you, for example, query for text that is

relevant to a specific part of a book, IDOL Server can find the appropriate section and return it (if the book was not indexed in sections, IDOL Server might not find the text you search for, because it might not be conceptually relevant to the entire book).

security

Security includes anything that makes sure that only authorized users can access or perform actions on data. It includes making sure that only permitted users can view and retrieve documents, user authentication, and secure communications.

seed

In clustering with snapshots, a seed is a potential cluster. It contains a document, and suggested conceptually similar documents from the IDOL Server index.

sentiment analysis

A form of Eduction that identifies positive and negative sentiment in text.

snapshot

Internal raw data from which you can extract clusters. You can thus generate cluster information and spectrographs.

soundex

An IDOL Server query type that allows you to search for a term by using a phonetic spelling.

spectrograph

A graphical representation of the results of clustering. The spectrograph displays clusters of documents, and the similarities between different clusters.

spidering

The process that Connectors use to retrieve content from Web resources, by recursively following hyperlinks from an initial page.

stemming

The process of extracting the morphological root of a word. In languages, some words have a common morphological root. IDOL includes stemming algorithms that reduce words to this form. This process allows IDOL Server to match concepts regardless of the grammatical use of words. In English, for example, the words 'help', 'helpful', 'helping', and 'helped' all reduce to their stem 'help' without significant loss of meaning. IDOL includes as standard a set of stemming algorithms for the most commonly used languages. IDOL Server applies stemming after stop words have been discarded, both at index time (when content is stored in IDOL Server), and at query time (query text is stopped and stemmed before it is matched).

stop word

A very common word that occurs too frequently to be useful for searching. Stop words include articles (for example, the) and prepositions (for example, to or from). Stop words are language-specific. You can use a stop word list in IDOL Server to allow it to discard these words at index and query time to save index space and improve retrieval performance.

stop word list

Also called stop list. A list (located in the IDOL Server langfiles directory) that contains common words (stop words) that IDOL Server does not store. Words such as the, and, or a occur too frequently to carry any significance, and IDOL Server does not require them to understand the concept of the text.

stopping

The process of removing the words listed in the stop word list from documents before they are stored in IDOL Server, and from query text before it is matched against IDOL Server content.

stored state

A set of query results, which is stored in IDOL Server for you to re-use later in other operations. When you store a state, IDOL Server provides a state token, which you can use to retrieve the stored state.

subfile

A file that has been extracted from a container (such as a .ZIP archive)

suggest

A type of query that returns documents that contain similar concepts to a particular document, rather than matching a particular query string. See Also: query

summary

A few sentences or paragraphs that describe what a document is about. IDOL can automatically create summaries from document content.

synonym file

A file that allows IDOL Server to handle synonym queries. A synonym query returns results which are conceptually similar to the query terms, and conceptually similar to the synonyms that are available for the query terms. A synonym file contains comma-separated lists of synonym strings for words. You can specify lists for each language type that you have set up in IDOL Server in this file.

synonym search

A type of query that returns documents that contain synonyms for a particular search term, as well as documents that contain the term. See Also: query, synonym file

T

tagging

The process of adding extra information to documents. The tag might be a category, or entities returned from Eduction. Tagging usually adds a field to a document, which you can use to search by the name of a tag.

taxonomy

An automatically created hierarchical structure of clusters or other information. A taxonomy provides you with an overview of the information landscape, and an insight into specific areas of the information.

term

The basic entity that IDOL Server indexes (for example, a word in a document after it has been stemmed).

Terms and Weights.

See TNW.

TNW

Terms and Weights. These values are used in categorization to define the most important terms that define a category topic.

training

Text, documents, and query syntax used to define the topic that an agent or category must match.

U

Universal Query Language.

See UQL.

unmapped security

A security setup where IDOL Server checks the security entitlement of a user against the original data repositories in real time when the user attempts to access a document. With this method, IDOL Server always has the current security information, but the response can be slow because of the additional connection to the repository. Compare With: mapped security

UQL

Universal Query Language. A name for the IDOL Server query syntax, which you can use for keyword, conceptual, Boolean, and Wildcard searches.

V

View

An IDOL component that converts files in a repository to HTML formats for viewing in a Web browser.

W

Wildcard

A character that stands in for any character or group of characters in a query.

X

XML

Extensible Markup Language. XML is a language that defines the different attributes of document content in a format that can be read by humans and machines. In IDOL Server, you can index documents in XML format. IDOL Server also returns action responses in XML format.

Send documentation feedback

If you have comments about this document, you can contact the documentation team by email. If an email client is configured on this system, click the link above and an email window opens with the following information in the subject line:

Feedback on Micro Focus IDOL Server 12.8 Administration Guide

Add your feedback to the email and click **Send**.

If no email client is available, copy the information above to a new message in a web mail client, and send your feedback to swpdl.idoldocsfeedback@microfocus.com.

We appreciate your feedback!