MICRO FOCUS®

SharePoint OData Connector

Software Version: 11.6

Administration Guide

Legal notices

Warranty

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

Restricted rights legend

Confidential computer software. Except as specifically indicated, valid license from Seattle required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright notice

© Copyright 2018 EntIT Software LLC, a Micro Focus company

Trademark notices

Adobe™ is a trademark of Adobe Systems Incorporated.

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

UNIX® is a registered trademark of The Open Group.

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 verify you are using the most recent edition of a document, go to https://softwaresupport.softwaregrp.com/group/softwaresupport/search-result?doctype=online help.

This site requires you to sign in with a Software Passport. You can register for a Passport through a link on the site.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your Micro Focus sales representative for details.

Support

Visit the Micro Focus Software Support Online website at https://softwaresupport.softwaregrp.com.

This website provides contact information and details about the products, services, and support that Micro Focus offers.

Micro Focus online support provides customer self-solve capabilities. It provides 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 support website to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Access the Software Licenses and Downloads portal
- Download software patches
- Access product documentation
- Manage support contracts

- Look up Micro Focus support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require you to register as a Passport user and sign in. Many also require a support contract.

You can register for a Software Passport through a link on the Software Support Online site.

To find more information about access levels, go to
https://softwaresupport.softwaregrp.com/web/softwaresupport/access-levels.

About this PDF version of online Help

This document is a PDF version of the online Help.

This PDF file is provided so you can easily print multiple topics or read the online Help.

Because this content was originally created to be viewed as online help in a web browser, some topics may not be formatted properly. Some interactive topics may not be present in this PDF version. Those topics can be successfully printed from within the online Help.

Contents

Chapter 1: Introduction

This section provides an overview of the Micro Focus SharePoint OData Connector.

SharePoint OData Connector

SharePoint OData Connector is an IDOL connector that retrieves information from a Microsoft SharePoint repository, through the SharePoint OData REST API.

The connector can also retrieve information from an instance of SharePoint Online.

After the connector has retrieved information from SharePoint, the files are sent to Connector Framework Server (CFS), which processes the information and indexes it into an IDOL Server or Haven OnDemand.

After the documents are indexed, IDOL server automatically processes them, performing a number of intelligent operations in real time, such as:

- Agents
- Alerting
- Automatic Query Guidance
- Categorization
- Channels
- Clustering
- Collaboration
- Dynamic Clustering
- Dynamic Thesaurus

- Eduction
- Expertise
- Hyperlinking
- Mailing
- Profiling
- Retrieval
- Spelling Correction
- Summarization
- Taxonomy Generation

Features and Capabilities

The SharePoint OData Connector retrieves data from a SharePoint or SharePoint Online repository.

Repository	SharePoint Online.
	Microsoft SharePoint 2016.
	Microsoft SharePoint 2013.

	Microsoft SharePoint 2013 Foundation Edition.
Data the connector can retrieve	The connector creates documents for:

- Sites.
- Lists.
- List Items, including:
  - Announcements.
  - Appointments
  - Blog Entries
  - Posts in discussion forums.
  - Publishing pages.
  - Survey responses.
  - Tasks.
  - Wiki pages.
- Attachments to list items.
- Folders in lists and document libraries.
- Files in document libraries.
- User Profiles.
- Social feeds associated with SharePoint users and SharePoint sites.

Data the connector cannot retrieve	The connector uses an API to retrieve the information stored in SharePoint, rather than crawling the web pages presented by the front end. This means that information displayed on a single page can be indexed in separate documents.

You can use the `IndexHidden` parameter to index hidden items. However, if you set `ChangesFromContentDatabase=true` in the connector's configuration file, the connector cannot process updates to hidden lists and list items. This is due to a limitation with the Microsoft SharePoint APIs.

Supported Actions

The SharePoint OData Connector supports the following actions:

Action	Supported	Further Information
Synchronize	✔	
Synchronize (identifiers)	✔	The connector's `Synchronize` action supports the optional `identifiers` parameter, which accepts a comma-separated list of document identifiers. If you set this parameter the action synchronizes the specified documents, regardless of whether they have changed.

Synchronize Groups	✔	The connector's `SynchronizeGroups` action is used by OmniGroupServer to retrieve security groups from the SharePoint repository. Micro Focus recommends scheduling `SynchronizeGroups` tasks in the OmniGroupServer configuration file.
Collect	✔	You can collect the following items from SharePoint: • A site • A list • A list item, file, or file version • An attachment
Identifiers	✔	
Insert	✔	Insert Information into the SharePoint Repository, on page 54
Delete/Remove	✔	Delete Items from the SharePoint Repository, on page 63
Hold/ReleaseHold	✘	
Update	✔	Update Metadata of Items in SharePoint, on page 62
Stub	✔	The connector can insert a stub as a file, list item, or attachment. For more information about inserting stubs, see Insert a Stub into the SharePoint Repository, on page 61.
GetURI	✔	You can obtain a URI for the following items in SharePoint: • A site • A list • A list item, file, or file version • An attachment
View	✔	You can view a file, file version, or attachment.

Mapped Security

You can use *Mapped Security* to protect documents that you retrieve from SharePoint using the SharePoint OData Connector.

The connector can retrieve security permissions (Access Control Lists) that are attached to items in the repository. The connector adds an Access Control List to each document that is indexed into IDOL. The connector uses LDAP to resolve SIDs in ACLs into user and group names.

The connector can also retrieve SharePoint security group information from the repository and send it to OmniGroupServer. The connector retrieves security group information when OmniGroupServer initiates the connector's `SynchronizeGroups` fetch action.

For detailed information about how to configure Mapped Security for documents retrieved from SharePoint, see SharePoint Mapped Security, on page 66.

OEM Certification

SharePoint OData Connector works in OEM licensed environments.

Connector Framework Server

Connector Framework Server (CFS) processes the information that is retrieved by connectors, and then indexes the information into IDOL.

A single CFS can process information from any number of connectors. For example, a CFS might process files retrieved by a File System Connector, web pages retrieved by a Web Connector, and e-mail messages retrieved by an Exchange Connector.

To use the SharePoint OData Connector to index documents into IDOL Server, you must have a CFS. When you install the SharePoint OData Connector, you can choose to install a CFS or point the connector to an existing CFS.

For information about how to configure and use Connector Framework Server, refer to the *Connector Framework Server Administration Guide*.

Filter Documents and Extract Subfiles

The documents that are sent by connectors to CFS contain only metadata extracted from the repository, such as the location of a file or record that the connector has retrieved. CFS uses KeyView to extract the file content and file specific metadata from over 1000 different file types, and adds this information to the documents. This allows IDOL to extract meaning from the information contained in the repository, without needing to process the information in its native format.

CFS also uses KeyView to extract and process sub-files. Sub-files are files that are contained within other files. For example, an e-mail message might contain attachments that you want to index, or a Microsoft Word document might contain embedded objects.

Manipulate and Enrich Documents

CFS provides features to manipulate and enrich documents before they are indexed into IDOL. For example, you can:

- add additional fields to a document.
- divide long documents into multiple sections.
- run tasks including Eduction, Optical Character Recognition, or Face Recognition, and add the information that is obtained to the document.
- run a custom Lua script to modify a document.

Index Documents

After CFS finishes processing documents, it automatically indexes them into one or more indexes. CFS can index documents into:

- **IDOL Server** (or send them to a *Distributed Index Handler*, so that they can be distributed across multiple IDOL servers).
- **Haven OnDemand**.
- **Vertica**.

Import Process

This section describes the import process for new files that are added to IDOL through CFS.
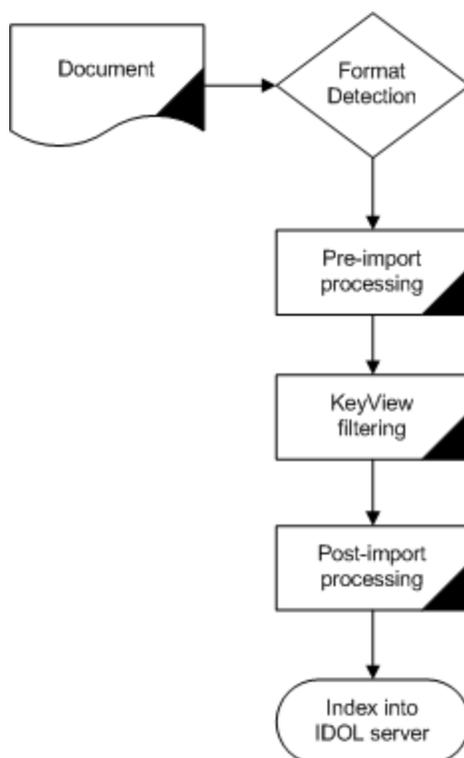
1. Connectors aggregate documents from repositories and send the files to CFS. A single CFS can process documents from multiple connectors. For example, CFS might receive HTML files from HTTP Connectors, e-mail messages from Exchange Connector, and database records from ODBC Connector.

2. CFS runs pre-import tasks. Pre-Import tasks occur before document content and file-specific metadata is extracted by KeyView.

3. KeyView filters the document content, and extracts sub-files.

4. CFS runs post-import tasks. Post-Import tasks occur after KeyView has extracted document content and file-specific metadata.

5. The data is indexed into IDOL.

The IDOL Platform

At the core of SharePoint OData Connector is the *Intelligent Data Operating Layer* (IDOL).

IDOL gathers and processes unstructured, semi-structured, and structured information in any format from multiple repositories using IDOL connectors and a global relational index. It can automatically form a contextual understanding of the information in real time, linking disparate data sources together based on the concepts contained within them. For example, IDOL can automatically link concepts contained in an email message to a recorded phone conversation, that can be associated with a stock trade. This information is then imported into a format that is easily searchable, adding advanced retrieval, collaboration, and personalization to an application that integrates the technology.

For more information on IDOL, see the *IDOL Getting Started Guide*.

System Architecture

An IDOL infrastructure can include the following components:

- **Connectors**. Connectors aggregate data from repositories and send the data to CFS.
- **Connector Framework Server (CFS)**. Connector Framework Server (CFS) processes and enriches the information that is retrieved by connectors.
- **IDOL Server**. IDOL stores and processes the information that is indexed into it by CFS.
- **Distributed Index Handler (DIH)**. The Distributed Index Handler distributes data across multiple IDOL servers. Using multiple IDOL servers can increase the availability and scalability of the system.
- **License Server**. The License server licenses multiple products.

These components can be installed in many different configurations. The simplest installation consists of a single connector, a single CFS, and a single IDOL Server.

Repository   Connector   CFS   IDOL
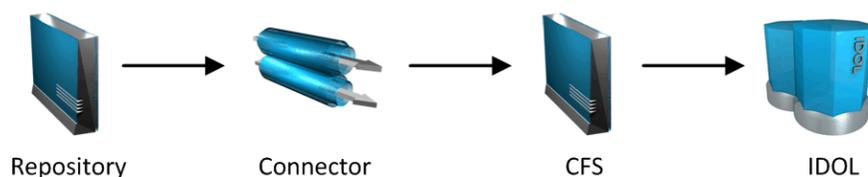
A more complex configuration might include more than one connector, or use a Distributed Index Handler (DIH) to index content across multiple IDOL Servers.
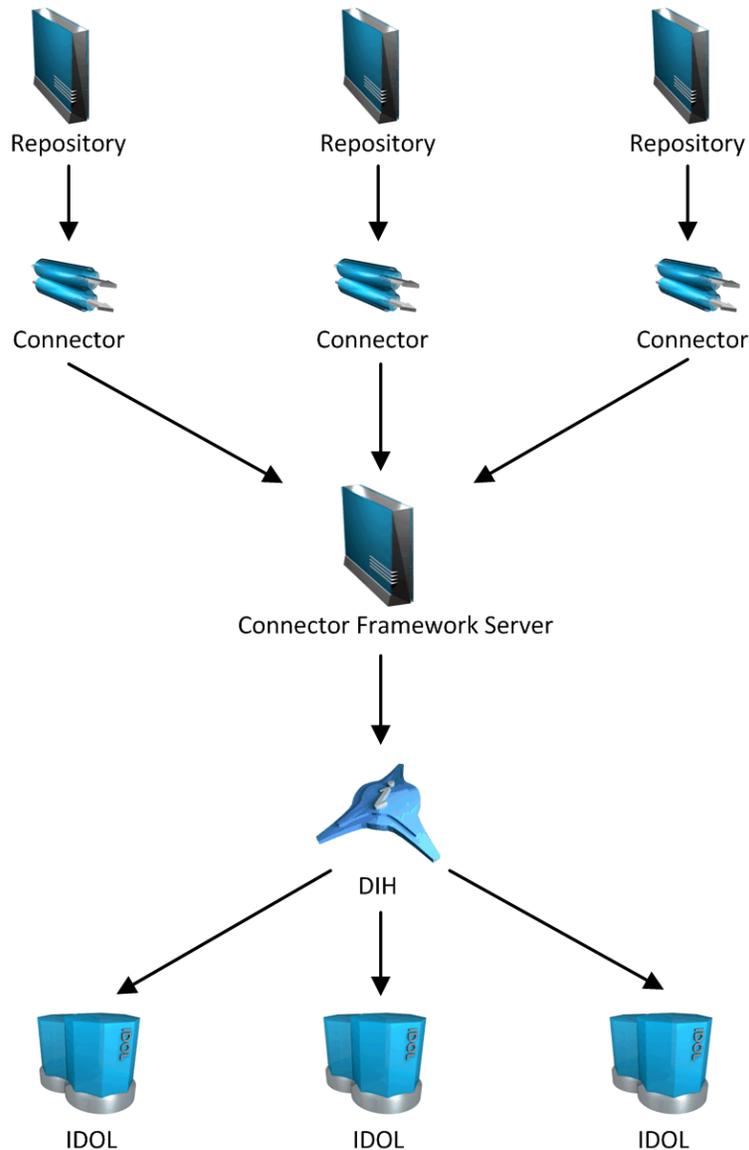
Display Online Help

You can display the SharePoint OData Connector Reference by sending an action from your web browser. The SharePoint OData Connector Reference describes the actions and configuration parameters that you can use with SharePoint OData Connector.

For SharePoint OData Connector to display help, the help data file (`help.dat`) must be available in the installation folder.

To display help for SharePoint OData Connector

1. Start SharePoint OData Connector.

2. Send the following action from your web browser:

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

   where:

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

   For example:

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

Related Documentation

The following documents provide further information related to SharePoint OData Connector.

- *SharePoint OData Connector Reference*

  The *SharePoint OData Connector Reference* describes the configuration parameters and actions that you can use with the SharePoint OData Connector.

- *Connector Framework Server Administration Guide*

  Connector Framework Server (CFS) processes documents that are retrieved by connectors. CFS then indexes the documents into IDOL Server, Haven OnDemand, or Vertica. The *Connector Framework Server Administration Guide* describes how to configure and use CFS.

- *IDOL Getting Started Guide*

  The *IDOL Getting Started Guide* provides an introduction to IDOL. It describes the system architecture, how to install IDOL components, and introduces indexing and security.

- *IDOL Server Administration Guide*

  The *IDOL Server Administration Guide* describes the operations that IDOL server can perform with detailed descriptions of how to set them up.

- *IDOL Document Security Administration Guide*

  The *IDOL Document Security Administration Guide* describes how to protect the information that you index into IDOL Server.

- *License Server Administration Guide*

  This guide describes how to use a License Server to license multiple services.

Chapter 2: Install SharePoint OData Connector

This section describes how to install the SharePoint OData Connector.

System Requirements

SharePoint OData Connector can be installed as part of a larger system that includes an IDOL Server and an interface for the information stored in IDOL Server. To maximize performance, Micro Focus recommends that you install IDOL Server and the connector on different machines.

You can install the connector and CFS on the same machine and hard disk. However, to maximize performance, Micro Focus recommends that you install the connector and CFS on separate machines.

For information about the minimum system requirements required to run IDOL components, including SharePoint OData Connector, refer to the *IDOL Getting Started Guide*.

OmniGroupServer

To use Mapped Security to protect documents that you extract from SharePoint and index into IDOL, you must install OmniGroupServer. OmniGroupServer can be installed anywhere on the system, but it must be able to communicate with the connector, your Active Directory, and your IDOL Server.

Permissions

To use the connector with an on-premise SharePoint server, the following permissions are required:

- To perform the `Synchronize`, `SynchronizeGroups`, `Collect`, and `View` actions, the user specified by the `Username` parameter in the connector's configuration file must have "Full Read" access. Micro Focus recommends that you create a web application policy to grant this permission to the relevant user.
- To index user profiles (when `IndexUserProfiles=true`) the user specified in the configuration file needs to have at least "Retrieve People Data for Search Crawlers" permission on the user profile service application.
- To insert documents using the `Insert` fetch action the user specified in the configuration file needs to have sufficient privileges to create new documents. For example, create a web application policy granting "Full control" to the user.

When you use the connector with SharePoint Online, the connector can retrieve any information that the user (specified by the `Username` parameter in the connector's configuration file) has permission to read. Some operations require additional privileges or additional configuration:

- To enable mapped security or index user profiles, the user must be a site-collection administrator.

- If you set `IndexSiteAssets=False` (which is the default value), the connector has to determine whether lists are site assets libraries. This requires site collection administrator privileges, so if the user is not a site-collection administrator and you want to ignore site asset libraries you must exclude them using the configuration parameters `ListMustHaveRegex` and `ListCantHaveRegex`. If the connector attempts to determine whether a list is a site asset library and is unable to do so because it has insufficient permissions, a warning is written to the synchronize log and the list is ingested.

Install SharePoint OData Connector on Windows

To install the SharePoint OData Connector on Windows, use the following procedure.

To install the SharePoint OData Connector

1. Run the SharePoint OData Connector installation program.

   The installation wizard opens.

2. Read the installation instructions and click **Next**.

   The License Agreement dialog box opens.

3. Read the license agreement. If you agree to its terms, click **I accept the agreement** and click **Next**.

   The Installation Directory dialog box opens.

4. Choose an installation folder for SharePoint OData Connector and click **Next**.

   The Service Name dialog box opens.

5. In the **Service name** box, type a name to use for the connector's Windows service and click **Next**.

   The Service Port and ACI Port dialog box opens.

6. Type the following information, and click **Next**.

Service port	The port used by the connector to listen for service actions.
ACI port	The port used by the connector to listen for actions.

   The License Server Configuration dialog box opens.

7. Type the following information, and click **Next**.

License server host	The host name or IP address of your License server.
License server port	The ACI port of your License server.

   The IDOL database dialog box opens.

8. In the **IDOL database** box, type the name of the IDOL database that you want to index data into. Then, click **Next**.

   The Proxy Server dialog box opens.

9. If the connector must use a proxy server to access SharePoint, type the following information. Then, click **Next**.

Proxy host	The host name or IP address of the proxy server to use to access the repository.
Proxy port	The port of the proxy server to use to access the repository.
Proxy username	The user name to use to authenticate with the proxy server.
Proxy password	The password to use to authenticate with the proxy server.

    The SharePoint Online Server dialog box opens.

10. Choose whether you intend to retrieve data from an instance of SharePoint Online and click **Next**.

11. Type the following information, and click **Next**.

Sharepoint site collection URL	The site collection URL to use as the starting point for the `synchronize` fetch action.
Username	The user name to use to authenticate with SharePoint.
Password	The password to use to authenticate with SharePoint.
Domain	(Not for SharePoint Online). The domain of the specified user.

12. Choose whether to install a new CFS or use an existing CFS.

    - To install a new CFS, select the **Install a new CFS** check box and click **Next**.
      The Installation Directory dialog box opens. Go to the next step.
    - To use an existing CFS, clear the **Install a new CFS** check box and click **Next**.
      Type the **Hostname** and **Port** of your existing CFS. Then, click **Next** and go to step 17.

13. Choose an installation folder for the Connector Framework Server and then click **Next**.
    The Installation name dialog box opens.

14. In the **Service name** box, type a unique name for the Connector Framework service and click **Next**. The name must not contain any spaces.
    The CFS dialog box opens.

15. Type the following information, and click **Next**.

Service port	The port used by CFS to listen for service actions.
ACI port	The port used by CFS to listen for actions.

16. Type the following information and click **Next**.

IDOL Server hostname	The host name or IP address of the IDOL server that you want to index documents into.
ACI port	The ACI port of the IDOL server.

The Pre-Installation Summary dialog box opens.

17. Review the installation settings. If necessary, click **Back** to go back and change any settings. If you are satisfied with the settings, click **Next**.

    The connector is installed and the Advanced Configuration dialog box opens.

18. The installation wizard can run the connector, query the repository, and help you to configure what to synchronize.

    - To configure what to synchronize, select the **Perform advanced connector configuration** check box and click **Next**.

      The connector queries the repository. If you are connecting to SharePoint Online, go to step 20 because SharePoint Online does not have Site Collections. Otherwise, go to the next step.

    - To configure the connector manually, clear the **Perform advanced connector configuration** check box, and click **Next**.

      The installation is complete. Click **Finish**. You can now edit the connector's configuration file and start the connector.

19. In the **SiteCollectionUrlMustHaveRegex** and **SiteCollectionUrlCantHaveRegex** boxes, type regular expressions to choose the site collections that you want to retrieve information from.

SiteCollectionUrlMustHaveRegex	The absolute URL of a site collection must match the regular expression, or the site collection is not indexed.
SiteCollectionUrlCantHaveRegex	If the absolute URL of a site collection matches the regular expression, the site collection is not indexed.

    Then, in the **Next action** list, choose one of the following options and click **Next**:

    - **Test parameters** - filter the list of items, using the parameter values you entered.
    - **Finish configuration** - save the configuration and end the configuration process.
    - **Continue configuration** - continue configuring the connector.

    If you choose to continue configuring the connector, the connector queries the repository for a list of sites contained in your chosen site collections.

20. In the **SiteUrlMustHaveRegex** and **SiteUrlCantHaveRegex** boxes, type regular expressions to choose the sites that you want to retrieve information from.

SiteUrlMustHaveRegex	The absolute URL of a site must match the regular expression, or the site is not indexed.
SiteUrlCantHaveRegex	If the absolute URL of a site matches the regular expression, the site is not indexed.

    Then, in the **Next action** list, choose one of the following options and click **Next**:

    - **Test parameters** - filter the list of items, using the parameter values you entered.
    - **Finish configuration** - save the configuration and end the configuration process.
    - **Continue configuration** - continue configuring the connector.

    If you choose to continue configuring the connector, the connector queries the repository to find the lists contained in your chosen sites.

21. In the **ListUrlMustHaveRegex** and **ListUrlCantHaveRegex** boxes, type regular expressions to

choose the lists that you want to retrieve information from.

ListUrlMustHaveRegex   The absolute URL of a list must match the regular expression, or the list is not indexed.

ListUrlCantHaveRegex   If the absolute URL of a list matches the regular expression, the list is not indexed.

Then, in the **Next action** list, choose one of the following options and click **Next**:

- **Test regexes** - filter the list of items, using the regular expressions you entered.
- **Finish configuration** - save the configuration and complete the configuration process.

When you finish configuring the connector, the Connector Configuration Completed dialog box opens.

22. Click **Next**, and then click **Finish**.

Install SharePoint OData Connector on Linux

To install the SharePoint OData Connector, use the following procedure.

To install SharePoint OData Connector on Linux

1. Open a terminal in the directory in which you have placed the installer, and run the following command:

   `./ConnectorName_VersionNumber_Platform.exe --mode text`

2. Follow the on-screen instructions. For information about the options that are specified during installation, see Install SharePoint OData Connector on Windows. For more information about installing IDOL components, refer to the *IDOL Getting Started Guide*.

Chapter 3: Configure SharePoint OData Connector

This section describes how to configure the SharePoint OData Connector.

SharePoint OData Connector Configuration File

You can configure the SharePoint OData Connector by editing the configuration file. The configuration file is located in the connector's installation folder. You can modify the file with a text editor.

The parameters in the configuration file are divided into sections that represent connector functionality.

Some parameters can be set in more than one section of the configuration file. If a parameter is set in more than one section, the value of the parameter located in the most specific section overrides the value of the parameter defined in the other sections. For example, if a parameter can be set in "*TaskName* or `FetchTasks` or `Default`", the value in the `TaskName` section overrides the value in the `FetchTasks` section, which in turn overrides the value in the `Default` section. This means that you can set a default value for a parameter, and then override that value for specific tasks.

For information about the parameters that you can use to configure the SharePoint OData Connector, refer to the *SharePoint OData Connector Reference*.

Server Section

The `[Server]` section specifies the ACI port of the connector. It can also contain parameters that control the way the connector handles ACI requests.

Service Section

The `[Service]` section specifies the service port of the connector.

Actions Section

The `[Actions]` section contains configuration parameters that specify how the connector processes actions that are sent to the ACI port. For example, you can configure event handlers that run when an action starts, finishes, or encounters an error.

Logging Section

The `[Logging]` section contains configuration parameters that determine how messages are logged. You can use *log streams* to send different types of message to separate log files. The configuration file also contains a section to configure each of the log streams.

Connector Section

The `[Connector]` section contains parameters that control general connector behavior. For example, you can specify a schedule for the fetch tasks that you configure.

Default Section

The `[Default]` section is used to define default settings for configuration parameters. For example, you can specify default settings for the tasks in the `[FetchTasks]` section.

FetchTasks Section

The `[FetchTasks]` section lists the fetch tasks that you want to run. A *fetch task* is a task that retrieves data from a repository. Fetch tasks are usually run automatically by the connector, but you can also run a fetch task by sending an action to the connector's ACI port.

In this section, enter the total number of fetch tasks in the `Number` parameter and then list the tasks in consecutive order starting from 0 (zero). For example:

```
[FetchTasks]
Number=2
0=MyTask0
1=MyTask1
```

[*TaskName*] Section

The `[TaskName]` section contains configuration parameters that apply to a specific task. There must be a `[TaskName]` section for every task listed in the `[FetchTasks]` section.

Ingestion Section

The [Ingestion] section specifies where to send the data that is extracted by the connector.

You can send data to a Connector Framework Server, Haven OnDemand, or another connector. For more information about ingestion, see Ingestion, on page 88.

DistributedConnector Section

The [DistributedConnector] section configures the connector to operate with the Distributed Connector. The Distributed Connector is an ACI server that distributes actions (synchronize, collect and so on) between multiple connectors.

For more information about the Distributed Connector, refer to the *Distributed Connector Administration Guide*.

ViewServer Section

The [ViewServer] section contains parameters that allow the connector's *view* action to use a View Server. If necessary, the View Server converts files to HTML so that they can be viewed in a web browser.

License Section

The [License] section contains details about the License server (the server on which your license file is located).

Document Tracking Section

The [DocumentTracking] section contains parameters that enable the tracking of documents through import and indexing processes.

Related Topics

- Modify Configuration Parameter Values, below
- Customize Logging, on page 103

Modify Configuration Parameter Values

You modify SharePoint OData Connector 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 SharePoint OData Connector 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.

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 SharePoint OData Connector 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 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 to and name of the external configuration file, in quotation marks (""). You can use relative paths and network locations. After the configuration file name, 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 a Parameter from an External Configuration File

This method allows you to import a parameter from an external configuration file at a specified point in your configuration file. You can include a section or a single parameter in this way, but the value in the external file must exactly match what you want to use in your file.

To include a parameter 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 parameter from 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. After the configuration file name, add the name of the configuration section name 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
   ```

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 SharePoint OData Connector 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, SharePoint OData Connector uses the values in the `[SharedSSLOptions]` section of the external configuration file as the values in the `[SSLOptions1]` section of the SharePoint OData Connector 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, SharePoint OData Connector 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.

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.

To create a new key file

1. Open a command-line window and change directory to the SharePoint OData Connector 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. SharePoint OData Connector 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 SharePoint OData Connector 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** 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**
-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 SharePoint OData Connector installation folder.

2. At the command line, type:

```
autpassword -d -tEncryptionType [-oKeyFile] PasswordString
```

where:

Option	Description
-t *EncryptionType*	The type of encryption: • **Basic** • **AES** For example: **-tAES**
-oKeyFile	AES encryption and decryption requires a key file. This option specifies the path and file name of the key file used to decrypt the password. For example: **-oKeyFile=./key.ky**
PasswordString	The password to decrypt.

For example:

```
autpassword -d -tBASIC 9t3M3t7awt/J8A
```

```
autpassword -d -tAES -oKeyFile=./key.ky 9t3M3t7awt/J8A
```

The password is returned in plain text.

Configure Client Authorization

You can configure SharePoint OData Connector 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` 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 *SharePoint OData Connector Reference*.

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`, and `ServiceActions`. For example:

```
[AdminRole]
StandardRoles=Admin,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,ServiceControl,ServiceStatus
Clients=localhost
SSLIdentities=admin.example.com
```

7. Save and close the configuration file.
8. Restart SharePoint OData Connector for your changes to take effect.

Register with a Distributed Connector

To receive actions from a Distributed Connector, a connector must register with the Distributed Connector and join a *connector group*. A connector group is a group of similar connectors. The connectors in a group must be of the same type (for example, all HTTP Connectors), and must be able to access the same repository.

To configure a connector to register with a Distributed Connector, follow these steps. For more information about the Distributed Connector, refer to the *Distributed Connector Administration Guide*.

To register with a Distributed Connector

1. Stop the connector.
2. Open the connector's configuration file in a text editor.
3. In the `[DistributedConnector]` section, set the following parameters:

    `RegisterConnector`   (Required) To register with a Distributed Connector, set this parameter to **true**.

HostN	(Required) The host name or IP address of the Distributed Connector.
PortN	(Required) The ACI port of the Distributed Connector.
DataPortN	(Optional) The data port of the Distributed Connector.
ConnectorGroup	(Required) The name of the connector group to join. The value of this parameter is passed to the Distributed Connector.
ConnectorPriority	(Optional) The Distributed Connector can distribute actions to connectors based on a priority value. The lower the value assigned to ConnectorPriority, the higher the probability that an action is assigned to this connector, rather than other connectors in the same connector group.
SharedPath	(Optional) The location of a shared folder that is accessible to all of the connectors in the ConnectorGroup. This folder is used to store the connectors' datastore files, so that whichever connector in the group receives an action, it can access the information required to complete it. If you set the DataPortN parameter, the datastore file is streamed directly to the Distributed Connector, and this parameter is ignored.

4. Save and close the configuration file.

5. Start the connector.

   The connector registers with the Distributed Connector. When actions are sent to the Distributed Connector for the connector group that you configured, they are forwarded to this connector or to another connector in the group.

Set Up Secure Communication

You can configure Secure Socket Layer (SSL) connections between the connector and other ACI servers.

Configure Outgoing SSL Connections

To configure the connector to send data to other components (for example Connector Framework Server) over SSL, follow these steps.

To configure outgoing SSL connections

1. Open the SharePoint OData Connector configuration file in a text editor.

2. Specify the name of a section in the configuration file where the SSL settings are provided:

   - To send data to an ingestion server over SSL, set the IngestSSLConfig parameter in the [Ingestion] section. To send data from a single fetch task to an ingestion server over SSL, set IngestSSLConfig in a [*TaskName*] section.

   - To send data to a Distributed Connector over SSL, set the SSLConfig parameter in the [DistributedConnector] section.

- To send data to a View Server over SSL, set the `SSLConfig` parameter in the `[ViewServer]` section.

  You can use the same settings for each connection. For example:

  ```
  [Ingestion]
  IngestSSLConfig=SSLOptions

  [DistributedConnector]
  SSLConfig=SSLOptions
  ```

3. Create a new section in the configuration file. The name of the section must match the name you specified in the `IngestSSLConfig` or `SSLConfig` parameter. Then specify the SSL settings to use.

`SSLMethod`	The SSL protocol to use.
`SSLCertificate`	(Optional) The SSL certificate to use (in PEM format).
`SSLPrivateKey`	(Optional) The private key for the SSL certificate (in PEM format).

   For example:

   ```
   [SSLOptions]
   SSLMethod=TLSV1.2
   SSLCertificate=host1.crt
   SSLPrivateKey=host1.key
   ```

4. Save and close the configuration file.
5. Restart the connector.

Related Topics

-

Configure Incoming SSL Connections

To configure a connector to accept data sent to its ACI port over SSL, follow these steps.

To configure an incoming SSL Connection

1. Stop the connector.
2. Open the configuration file in a text editor.
3. In the `[Server]` section set the `SSLConfig` parameter to specify the name of a section in the configuration file for the SSL settings. For example:

   ```
   [Server]
   SSLConfig=SSLOptions
   ```

4. Create a new section in the configuration file (the name must match the name you used in the `SSLConfig` parameter). Then, use the SSL configuration parameters to specify the details for the connection. You must set the following parameters:

`SSLMethod`	The SSL protocol to use.

SSLCertificate   The SSL certificate to use (in PEM format).

SSLPrivateKey   The private key for the SSL certificate (in PEM format).

For example:

[SSLOptions]
SSLMethod=TLSV1.2
SSLCertificate=host1.crt
SSLPrivateKey=host1.key

5. Save and close the configuration file.

6. Restart the connector.

Related Topics

Backup and Restore the Connector's State

After configuring a connector, and while the connector is running, you can create a backup of the connector's state. In the event of a failure, you can restore the connector's state from the backup.

To create a backup, use the `backupServer` action. The `backupServer` action saves a ZIP file to a path that you specify. The backup includes:

- a copy of the `actions` folder, which stores information about actions that have been queued, are running, and have finished.
- a copy of the configuration file.
- a copy of the connector's datastore files, which contain information about the files, records, or other data that the connector has retrieved from a repository.

Backup a Connector's State

To create a backup of the connectors state

- In the address bar of your Web browser, type the following action and press **ENTER**:

  `http://`*host*`:`*port*`/action=backupServer&path=`*path*

  where,

  host   The host name or IP address of the machine where the connector is running.

  port   The connector's ACI port.

  path   The folder where you want to save the backup.

  For example:

  `http://localhost:1234/action=backupServer&path=./backups`

Restore a Connector's State

To restore a connector's state

- In the address bar of your Web browser, type the following action and press **ENTER**:

  `http://`*`host`*`:`*`port`*`/action=restoreServer&filename=`*`filename`*

  where,

host	The host name or IP address of the machine where the connector is running.
port	The connector's ACI port.
filename	The path of the backup that you created.

  For example:

  `http://localhost:1234/action=restoreServer&filename=./backups/filename.zip`

Validate the Configuration File

You can use the `ValidateConfig` service action to check for errors in the configuration file.

> **NOTE:**
> For the `ValidateConfig` action to validate a configuration section, SharePoint OData Connector must have previously read that configuration. In some cases, the configuration might be read when a task is run, rather than when the component starts up. In these cases, `ValidateConfig` reports any unread sections of the configuration file as unused.

To validate the configuration file

- Send the following action to SharePoint OData Connector:

  `http://`*`Host`*`:`*`ServicePort`*`/action=ValidateConfig`

  where:

Host	is the host name or IP address of the machine where SharePoint OData Connector is installed.
ServicePort	is the service port, as specified in the `[Service]` section of the configuration file.

Chapter 4: Start and Stop the Connector

This section describes how to start and stop the SharePoint OData Connector.

> **NOTE:**
> You must start and stop the Connector Framework Server separately from the SharePoint OData Connector.

Start the Connector

After you have installed and configured a connector, you are ready to run it. Start the connector using one of the following methods.

Start the Connector on Windows

To start the connector using Windows Services

1. Open the Windows Services dialog box.
2. Select the connector service, and click **Start**.
3. Close the Windows Services dialog box.

To start the connector by running the executable

- In the connector installation directory, double-click the connector executable file.

Start the Connector on UNIX

To start the connector on a UNIX operating system, follow these steps.

To start the connector using the UNIX start script

1. Change to the installation directory.
2. Enter the following command:

   ```
   ./startconnector.sh
   ```

3. If you want to check the SharePoint OData Connector service is running, enter the following command:

   ```
   ps -aef | grep ConnectorInstallName
   ```

This command returns the SharePoint OData Connector service process ID number if the service is running.

Verify that SharePoint OData Connector is Running

After starting SharePoint OData Connector, you can run the following actions to verify that SharePoint OData Connector is running.

- GetStatus
- GetLicenseInfo

GetStatus

You can use the `GetStatus` service action to verify the SharePoint OData Connector is running. For example:

```
http://Host:ServicePort/action=GetStatus
```

> **NOTE:**
> You can send the `GetStatus` action to the ACI port instead of the service port. The `GetStatus` ACI action returns information about the SharePoint OData Connector setup.

GetLicenseInfo

You can send a `GetLicenseInfo` action to SharePoint OData Connector to return information about your license. This action checks whether your license is valid and returns the operations that your license includes.

Send the `GetLicenseInfo` action to the SharePoint OData Connector ACI port. For example:

```
http://Host:ACIport/action=GetLicenseInfo
```

The following result indicates that your license is valid.

```
<autn:license>
        <autn:validlicense>true</autn:validlicense>
</autn:license>
```

As an alternative to submitting the `GetLicenseInfo` action, you can view information about your license, and about licensed and unlicensed actions, on the **License** tab in the Status section of IDOL Admin.

Stop the Connector

You must stop the connector before making any changes to the configuration file.

To stop the connector using Windows Services

1. Open the Windows Services dialog box.
2. Select the *ConnectorInstallName* service, and click **Stop**.
3. Close the Windows Services dialog box.

To stop the connector by sending an action to the service port

- Type the following command in the address bar of your Web browser, and press ENTER:

  `http://host:ServicePort/action=stop`

host	The IP address or host name of the machine where the connector is running.
ServicePort	The connector's service port (specified in the `[Service]` section of the connector's configuration file).

Chapter 5: Send Actions to SharePoint OData Connector

This section describes how to send actions to SharePoint OData Connector.

Send Actions to SharePoint OData Connector

SharePoint OData Connector 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 SharePoint OData Connector is installed.
port	is the SharePoint OData Connector ACI port. The ACI port is specified by the `Port` parameter in the `[Server]` section of the SharePoint OData Connector configuration file. For more information about the `Port` parameter, see the *SharePoint OData Connector 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 *SharePoint OData Connector Reference*.

Asynchronous Actions

When you send an asynchronous action to SharePoint OData Connector, the connector adds the task to a queue and returns a token. SharePoint OData Connector performs the task when a thread becomes available. You can use the token with the `QueueInfo` action to check the status of the action and retrieve the results of the action.

Most of the features provided by the connector are available through `action=fetch`, so when you use the `QueueInfo` action, query the `fetch` action queue, for example:

```
/action=QueueInfo&QueueName=Fetch&QueueAction=GetStatus
```

Check the Status of an Asynchronous Action

To check the status of an asynchronous action, use the token that was returned by SharePoint OData Connector with the `QueueInfo` action. For more information about the `QueueInfo` action, refer to the *SharePoint OData Connector Reference*.

To check the status of an asynchronous action

- Send the `QueueInfo` action to SharePoint OData Connector with the following parameters.

QueueName	The name of the action queue that you want to check.
QueueAction	The action to perform. Set this parameter to `GetStatus`.
Token	(Optional) The token that the asynchronous action returned. If you do not specify a token, SharePoint OData Connector returns the status of every action in the queue.

For example:

```
/action=QueueInfo&QueueName=fetch&QueueAction=getstatus&Token=...
```

Cancel an Asynchronous Action that is Queued

To cancel an asynchronous action that is waiting in a queue, use the following procedure.

To cancel an asynchronous action that is queued

- Send the `QueueInfo` action to SharePoint OData Connector with the following parameters.

QueueName	The name of the action queue that contains the action to cancel.
QueueAction	The action to perform . Set this parameter to `Cancel`.
Token	The token that the asynchronous action returned.

For example:

```
/action=QueueInfo&QueueName=fetch&QueueAction=Cancel&Token=...
```

Stop an Asynchronous Action that is Running

You can stop an asynchronous action at any point.

To stop an asynchronous action that is running

- Send the `QueueInfo` action to SharePoint OData Connector with the following parameters.

`QueueName`	The name of the action queue that contains the action to stop.
`QueueAction`	The action to perform. Set this parameter to `Stop`.
`Token`	The token that the asynchronous action returned.

For example:

```
/action=QueueInfo&QueueName=fetch&QueueAction=Stop&Token=...
```

Store Action Queues in an External Database

SharePoint OData Connector provides asynchronous actions. Each asynchronous action has a queue to store requests until threads become available to process them. You can configure SharePoint OData Connector to store these queues either in an internal database file, or in an external database hosted on a database server.

The default configuration stores queues in an internal database. Using this type of database does not require any additional configuration.

You might want to store the action queues in an external database so that several servers can share the same queues. In this configuration, sending a request to any of the servers adds the request to the shared queue. Whenever a server is ready to start processing a new request, it takes the next request from the shared queue, runs the action, and adds the results of the action back to the shared database so that they can be retrieved by any of the servers. You can therefore distribute requests between components without configuring a Distributed Action Handler (DAH).

> **NOTE:**
> You cannot use multiple servers to process a single request. Each request is processed by one server.

> **NOTE:**
> Although you can configure several connectors to share the same action queues, the connectors do not share fetch task data. If you share action queues between several connectors and distribute synchronize actions, the connector that processes a synchronize action cannot determine which items the other connectors have retrieved. This might result in some documents being ingested several times.

Prerequisites

- Supported databases:
  - PostgreSQL 9.0 or later.
  - MySQL 5.0 or later.
- If you use PostgreSQL, you must set the PostgreSQL ODBC driver setting `MaxVarChar` to `0` (zero).

If you use a DSN, you can configure this parameter when you create the DSN. Otherwise, you can set the `MaxVarcharSize` parameter in the connection string.

Configure SharePoint OData Connector

To configure SharePoint OData Connector to use a shared action queue, follow these steps.

To store action queues in an external database

1. Stop SharePoint OData Connector, if it is running.

2. Open the SharePoint OData Connector configuration file.

3. Find the relevant section in the configuration file:

   - To store queues for all asynchronous actions in the external database, find the `[Actions]` section.

   - To store the queue for a single asynchronous action in the external database, find the section that configures that action.

4. Set the following configuration parameters.

`AsyncStoreLibraryDirectory`	The path of the directory that contains the library to use to connect to the database. Specify either an absolute path, or a path relative to the server executable file.
`AsyncStoreLibraryName`	The name of the library to use to connect to the database. You can omit the file extension. The following libraries are available: • **postgresAsyncStoreLibrary** - for connecting to a PostgreSQL database. • **mysqlAsyncStoreLibrary** - for connecting to a MySQL database.

`ConnectionString`	The connection string to use to connect to the database. The user that you specify must have permission to create tables in the database. For example:
	
	`ConnectionString=DSN=my_ASYNC_QUEUE`
	
	or
	
	`ConnectionString=Driver={PostgreSQL};`
	`Server=10.0.0.1; Port=9876;`
	`Database=SharedActions; Uid=user; Pwd=password;`
	`MaxVarcharSize=0;`

   For example:

   ```
   [Actions]
   AsyncStoreLibraryDirectory=acidlls
   AsyncStoreLibraryName=postgresAsyncStoreLibrary
   ConnectionString=DSN=ActionStore
   ```

5. If you are using the same database to store action queues for more than one type of component,

set the following parameter in the `[Actions]` section of the configuration file.

`DatastoreSharingGroupName`	The group of components to share actions with. You can set this parameter to any string, but the value must be the same for each server in the group. For example, to configure several SharePoint OData Connectors to share their action queues, set this parameter to the same value in every SharePoint OData Connector configuration. Micro Focus recommends setting this parameter to the name of the component.

> **CAUTION:**
> Do not configure different components (for example, two different types of connector) to share the same action queues. This will result in unexpected behavior.

For example:

```
[Actions]
...
DatastoreSharingGroupName=ComponentType
```

6. Save and close the configuration file.

   When you start SharePoint OData Connector it connects to the shared database.

Store Action Queues in Memory

SharePoint OData Connector provides asynchronous actions. Each asynchronous action has a queue to store requests until threads become available to process them. These queues are usually stored in a datastore file or in a database hosted on a database server, but in some cases you can increase performance by storing these queues in memory.

> **NOTE:**
> Storing action queues in memory improves performance only when the server receives large numbers of actions that complete quickly. Before storing queues in memory, you should also consider the following:
>
> - The queues (including queued actions and the results of finished actions) are lost if SharePoint OData Connector stops unexpectedly, for example due to a power failure or the component being forcibly stopped. This could result in some requests being lost, and if the queues are restored to a previous state some actions could run more than once.
>
> - Storing action queues in memory prevents multiple instances of a component being able to share the same queues.
>
> - Storing action queues in memory increases memory use, so please ensure that the server has sufficient memory to complete actions and store the action queues.

If you stop SharePoint OData Connector cleanly, SharePoint OData Connector writes the action queues from memory to disk so that it can resume processing when it is next started.

To configure SharePoint OData Connector to store asynchronous action queues in memory, follow these steps.

To store action queues in memory

1. Stop SharePoint OData Connector, if it is running.

2. Open the SharePoint OData Connector configuration file and find the `[Actions]` section.

3. If you have set any of the following parameters, remove them:

   - `AsyncStoreLibraryDirectory`
   - `AsyncStoreLibraryName`
   - `ConnectionString`
   - `UseStringentDatastore`

4. Set the following configuration parameters.

`UseInMemoryDatastore`	A Boolean value that specifies whether to keep the queues for asynchronous actions in memory. Set this parameter to **TRUE**.
`InMemoryDatastoreBackupIntervalMins`	(Optional) The time interval (in minutes) at which the action queues are written to disk. Writing the queues to disk can reduce the number of queued actions that would be lost if SharePoint OData Connector stops unexpectedly, but configuring a frequent backup will increase the load on the datastore and might reduce performance.

   For example:

   ```
   [Actions]
   UseInMemoryDatastore=TRUE
   InMemoryDatastoreBackupIntervalMins=30
   ```

5. Save and close the configuration file.

   When you start SharePoint OData Connector, it stores action queues in memory.

Use XSL Templates to Transform Action Responses

You can transform the action responses returned by SharePoint OData Connector using XSL templates. You must write your own XSL templates and save them with either an `.xsl` or `.tmpl` file extension.

After creating the templates, you must configure SharePoint OData Connector to use them, and then apply them to the relevant actions.

To enable XSL transformations

1. Ensure that the `autnxslt` library is located in the same directory as your configuration file. If the library is not included in your installation, you can obtain it from Micro Focus Support.

2. Open the SharePoint OData Connector configuration file in a text editor.

3. In the `[Server]` section, ensure that the `XSLTemplates` parameter is set to **true**.

> **CAUTION:**
> If `XSLTemplates` is set to `true` and the `autnxslt` library is not present in the same directory as the configuration file, the server will not start.

4. (Optional) In the `[Paths]` section, set the `TemplateDirectory` parameter to the path to the directory that contains your XSL templates. The default directory is `acitemplates`.

5. Save and close the configuration file.

6. Restart SharePoint OData Connector for your changes to take effect.

To apply a template to action output

- Add the following parameters to the action:

`Template`	The name of the template to use to transform the action output. Exclude the folder path and file extension.
`ForceTemplateRefresh`	(Optional) If you modified the template after the server started, set this parameter to `true` to force the ACI server to reload the template from disk rather than from the cache.

    For example:

    ```
    action=QueueInfo&QueueName=Fetch
                    &QueueAction=GetStatus
                    &Token=...
                    &Template=myTemplate
    ```

    In this example, SharePoint OData Connector applies the XSL template `myTemplate` to the response from a `QueueInfo` action.

    > **NOTE:**
    > If the action returns an error response, SharePoint OData Connector does not apply the XSL template.

Example XSL Templates

SharePoint OData Connector includes the following sample XSL templates, in the `acitemplates` folder:

XSL Template	Description
`FetchIdentifiers`	Transforms the output from the `Identifiers` fetch action, to show what is in the repository.
`FetchIdentifiersTreeview`	Transforms the output from the `Identifiers` fetch action, to show what is in the repository. This template produces a tree or hierarchical view, instead of the flat view produced by the `FetchIdentifiers` template.
`LuaDebug`	Transforms the output from the `LuaDebug` action, to assist with debugging Lua scripts.

Chapter 6: Use the Connector

This section describes how to use the connector.

Retrieve Data from SharePoint

To automatically retrieve content from SharePoint, create a new fetch task by following these steps. The connector runs each fetch task automatically, based on the schedule that is defined in the configuration file.

To create a new Fetch Task

1. Stop the connector.

2. Open the configuration file in a text editor.

3. In the `[FetchTasks]` section of the configuration file, specify the number of fetch tasks using the `Number` parameter. If you are configuring the first fetch task, type `Number=1`. If one or more fetch tasks have already been configured, increase the value of the `Number` parameter by one (1). Below the `Number` parameter, specify the names of the fetch tasks, starting from zero (0). For example:

   ```
   [FetchTasks]
   Number=1
   0=MyTask
   ```

4. Below the `[FetchTasks]` section, create a new *TaskName* section. The name of the section must match the name of the new fetch task. For example:

   ```
   [FetchTasks]
   Number=1
   0=MyTask

   [MyTask]
   ```

5. In the new section, set the following parameters. These parameters are required to connect to the SharePoint repository.

SharepointUrl	The starting point for the synchronize action. You can specify either a web application URL or a site collection URL.
SharepointUrlType	The type of URL specified by SharepointUrl: • **WebApplication** • **SiteCollection**
Username	The user name for authentication with the Sharepoint OData API. You can encrypt the user name before adding it to the configuration file.
Password	The password for authentication with the Sharepoint OData API. Micro Focus recommends encrypting the password. For information about how to do this, see Encrypt Passwords, on page 28.
Domain	The domain of the user that is specified by the Username parameter. The domain can be encrypted.

6. (Optional) You can set additional configuration parameters to specify the information that you want to retrieve. For more information about the parameters that you can use, refer to the *SharePoint OData Connector Reference*.

7. Save and close the configuration file.

Related Topics

• SharePoint Mapped Security, on page 66

Retrieve Data from SharePoint Online

To automatically retrieve content from SharePoint Online (part of Office 365), create a new fetch task by following these steps. The connector runs each fetch task automatically, based on the schedule that is defined in the configuration file.

To create a new Fetch Task

1. Stop the connector.

2. Open the configuration file in a text editor.

3. In the [FetchTasks] section of the configuration file, specify the number of fetch tasks using the Number parameter. If you are configuring the first fetch task, type **Number=1**. If one or more fetch tasks have already been configured, increase the value of the Number parameter by one (1). Below the Number parameter, specify the names of the fetch tasks, starting from zero (0). For example:

```
[FetchTasks]
Number=1
0=MyTask
```

4. Below the [FetchTasks] section, create a new *TaskName* section. The name of the section must match the name of the new fetch task. For example:

```
[FetchTasks]
Number=1
0=MyTask
```

[MyTask]

5. In the new section, set the following parameters.

SharepointOnline	A Boolean that specifies whether you are retrieving data from a SharePoint Online shared server. To retrieve data from a SharePoint Online shared server, set this parameter to **true**. If you are retrieving data from a SharePoint Online dedicated server, set this parameter to **false**.
SSLMethod	The version of SSL/TLS to use to connect to SharePoint Online. To use the latest protocol supported by both client and server, set this parameter to **NEGOTIATE**.
SharepointUrlType	The type of site collection from which you want to retrieve data: • **SiteCollection** • **PersonalSiteCollection**
SharepointUrl	If you set SharepointUrlType=SiteCollection, specify the URL of the site collection. If you set SharepointUrlType=PersonalSiteCollection, specify the URL of the personal site collection.
SharepointAdminUrl	The URL of the admin site collection. If you want to retrieve user profiles from SharePoint Online, you must set this parameter.
SharepointMySiteUrl	The URL of the "MySites" site collection.
Username	The user name for authentication with SharePoint. You can encrypt the user name before adding it to the configuration file.
Password	The password for authentication with SharePoint. Micro Focus recommends encrypting the password. For information about how to do this, see Encrypt Passwords, on page 28.

6. (Optional) You can set additional configuration parameters to specify the information that you want to retrieve. For more information about the parameters that you can use, refer to the *SharePoint OData Connector Reference*.

7. Save and close the configuration file.

Related Topics

• SharePoint Mapped Security, on page 66

Connect to SharePoint with Federated Authentication

This section explains how to connect to a SharePoint server that uses federated authentication.

Federated Authentication with SharePoint Online

When the connector is configured to retrieve data from SharePoint Online, it makes a request to Microsoft Online (https://login.microsoftonline.com/GetUserRealm.srf) to determine if the SharePoint Online instance uses federated authentication. If the SharePoint Online instance uses federated authentication, the response can contain up to two authentication endpoints that the connector can use to authenticate. These are at the following locations in the XML response:

- `/RealmInfo/AuthURL` - This endpoint is used by default. The connector uses `https://`*`AuthURLHost`*`/adfs/services/trust/2005/usernamemixed/` as the ADFS proxy to authenticate with SharePoint Online.

- `/RealmInfo/STSAuthURL` - To use this endpoint as the ADFS proxy to authenticate with SharePoint Online, set the parameter `SharePointOnlineFederatedAuthUseSTSUrl` to `true`. Use this option when the default endpoint is not available.

A common indicator that you need to set the parameter `SharePointOnlineFederatedAuthUseSTSUrl` is the following error appearing in the connector logs: "Sign in failed using corporate credentials: The remote server returned an error: (404) Not Found".

Federated Authentication with SharePoint On-Premise

To retrieve information from an on-site SharePoint server with federated authentication, you must specify the details of the federation servers to use to authenticate with SharePoint:

- Use the `Username` and `Password` configuration parameters to specify the credentials to use to authenticate with the federation server.

- Set `FederatedAuthStsUrl` to the URL of the security token service (STS) endpoint of the federation server. For example, `FederatedAuthStsUrl=https://login-mydomain.com/idp/sts.wst`

- Set `FederatedAuthSharepointStsUrn` to the URN of the SharePoint server to use in the Request Security Token (RST) message sent to the security token service (STS) endpoint. For example, `FederatedAuthSharepointStsUrn=urn:sharepoint:myinstance`.

In most environments, this is all you need to do. However, in some scenarios you might need to configure the WS-Federation passive protocol URL for the SharePoint server. The usual WS-Federation passive protocol URL can be constructed by appending `/_trust/` to the URL of the Web Application or Site Collection that the connector has been configured to process, and if this is the case then no further configuration is needed. If the WS-Federation passive protocol URL for the SharePoint server is not at this location, then use the configuration parameter `FederatedAuthSharepointTrustUrl` to specify the WS-Federation passive protocol URL, for example `FederatedAuthSharepointTrustUrl=http://sharepoint/_trust/`.

Choose the Content to Index

This section explains how to configure the connector so that it retrieves the content that you want to index, and nothing else.

Restrict the Content to Process

The content in SharePoint is organized in the following structure.

```
Web Application (on-premise only)
  |- Site Collection
     |- Site
        |- Site
        |    |- ...
        |- Document Library
        |    |- File
        |    |    |- File version(s)
        |    |- Folder
        |        |- File
        |            |- File version(s)
        |- List
            |- List Item
                |- Attachment(s)
            |- Folder
                |- List Item
                    |- Attachment(s)
```

There can be multiple site collections, and multiple sites within a site or site collection. There can be multiple lists and document libraries within a site, multiple folders and files within a document library, and so on.

> **NOTE:**
> Instances of SharePoint Online have a single site collection at the root level, and no concept of Web Applications.

You can restrict the content to retrieve by setting the following configuration parameters:

- `SiteCollectionUrlCantHaveRegex` and `SiteCollectionUrlMustHaveRegex` restrict the site collections to process, when you are retrieving information from a Web Application (`SharePointUrlType=WebApplication`).

- `SiteUrlCantHaveRegex` and `SiteUrlMustHaveRegex` restrict the sites to process. If a site is excluded the connector does not index a document for the site and ignores all lists, list items, files, file versions, and attachments contained by the site.

  > **NOTE:**
  > If a site is excluded, its child sites are still processed, unless they are also excluded by one of the regular expressions.

- `ListUrlCantHaveRegex` and `ListUrlMustHaveRegex` restrict the lists and document libraries to process. If a list or document library is excluded, the connector does not index a document for the list or document library and ignores all list items, files, file versions, and attachments contained in the list or document library.

- `ListItemUrlCantHaveRegex` and `ListItemUrlMustHaveRegex` restrict the list items or files to process. If a list item or file is excluded, the connector does not index a document for the list item or file, and ignores all file versions and attachments for that list item or file.

- `IndexAttachments` specifies whether to index attachments for list items that are processed.

- `IndexFileVersions` and `VersionIndexingMode` specify which file versions to index for files that have versioning enabled.

- `FileExtnCantHaveCSVs` and `FileExtnMustHaveCSVs` restrict the files and attachments to process, based on their file extension.

- `IndexSites`, `IndexLists`, and `IndexFolders` specify whether to index a metadata-only document for each container object (sites, lists, and folders respectively).

The connector performs best if you choose the objects to process at the highest possible level. Take for example the following structure:

```
http://sharepoint/                         Site Collection
http://sharepoint/site1/                    Site
http://sharepoint/site1/List1               List
http://sharepoint/site1/List1/Item1         List Item
http://sharepoint/site1/List1/Item2         List Item
http://sharepoint/site1/List2               List
http://sharepoint/site1/List2/Item1         List Item
http://sharepoint/site1/List2/Item2         List Item
http://sharepoint/site2/                    Site
http://sharepoint/site2/List1               List
http://sharepoint/site2/List1/Item1         List Item
http://sharepoint/site2/SubSite/            Site
http://sharepoint/site2/SubSite/List1       List
http://sharepoint/site2/SubSite/List1/Item1  List Item
```

You could ignore all content from `site1` by configuring `ListUrlCantHaveRegex=http://sharepoint/site1/.*`, but the connector would have to process `site1`, and all of the lists on that site, just to determine that the lists should be ignored. A more efficient configuration is `SiteUrlCantHaveRegex=http://sharepoint/site1/`, because the connector can immediately determine that nothing from that site has to be processed.

Similarly, you could ignore content on `site2`, but still index content on `site2/subsite`, by configuring `ListUrlCantHaveRegex=http://sharepoint/site2/List.*`. However, the connector would have to process `site2` and all of the lists on that site, just to determine that the lists should be ignored. A more efficient configuration would contain `SiteUrlCantHaveRegex=http://sharepoint/site2/$`, so that the connector can immediately determine that nothing from `site2` has to be processed. The URL for `site2/subsite` does not match the regular expression `http://sharepoint/site2/$`, so content from `site2/subsite` is still processed.

Index Content that does not appear in Search Results

In SharePoint, a user can choose whether to allow items from a list or document library to appear in search results. Users can also choose whether to allow publishing pages (a type of list item) to appear in search engine results, using Search Engine Optimization (SEO) settings. In both cases, by default, the connector ignores items that do not appear. You can choose to modify this behavior:

- To index a list or document library regardless of whether it appears in search results, set the configuration parameter `IgnoreNoCrawl` to `True`.
- To index publishing pages regardless of SEO settings, set the configuration parameter `IgnoreRobotsNoIndex` to `True`.

Schedule Fetch Tasks

The connector automatically runs the fetch tasks that you have configured, based on the schedule in the configuration file. To modify the schedule, follow these steps.

To schedule fetch tasks

1. Stop the connector.
2. Open the connector's configuration file in a text editor.
3. Find the `[Connector]` section.
4. The `EnableScheduleTasks` parameter specifies whether the connector should automatically run the fetch tasks that have been configured in the `[FetchTasks]` section. To run the tasks, set this parameter to `true`. For example:

   ```
   [Connector]
   EnableScheduledTasks=True
   ```

5. In the `[Connector]` section, set the following parameters:

`ScheduleStartTime`	The start time for the fetch task, the first time it runs after you start the connector. The connector runs subsequent synchronize cycles after the interval specified by `ScheduleRepeatSecs`.
	Specify the start time in the format H[H][:MM][:SS]. To start running tasks as soon as the connector starts, do not set this parameter or use the value **now**.
`ScheduleRepeatSecs`	The interval (in seconds) from the start of one scheduled `synchronize` cycle to the start of the next. If a previous synchronize cycle is still running when the interval elapses, the connector queues a maximum of one action.
`ScheduleCycles`	The number of times that each fetch task is run. To run the tasks continuously until the connector is stopped, set this parameter to **-1**. To run each task only one time, set this parameter to **1**.

   For example:

```
[Connector]
EnableScheduledTasks=True
ScheduleStartTime=15:00:00
ScheduleRepeatSecs=3600
ScheduleCycles=-1
```

6. (Optional) To run a specific fetch task on a different schedule, you can override these parameters in a *TaskName* section of the configuration file. For example:

```
[Connector]
EnableScheduledTasks=TRUE
ScheduleStartTime=15:00:00
ScheduleRepeatSecs=3600
ScheduleCycles=-1


...

[FetchTasks]
Number=2
0=MyTask0
1=MyTask1

...

[MyTask1]
ScheduleStartTime=16:00:00
ScheduleRepeatSecs=60
ScheduleCycles=-1
```

In this example, `MyTask0` follows the schedule defined in the `[Connector]` section, and `MyTask1` follows the scheduled defined in the `[MyTask1]` *TaskName* section.

7. Save and close the configuration file. You can now start the connector.

Related Topics

Synchronize from Identifiers

The connector's `synchronize` action searches a repository for document updates and sends these updates for ingestion (for example, to CFS, for indexing into IDOL Server).

You can use the `identifiers` parameter to synchronize a specific set of documents, whether they have been updated or not, and ignore other files. For example:

`/action=fetch&fetchaction=synchronize&identifiers=<identifiers>`

(where `<identifiers>` is a comma-separated list of identifiers that specifies the documents to synchronize).

For example, if some documents fail the ingestion process, and are indexed into an IDOL Error Server, you can use the `identifiers` parameter with the `synchronize` action to retrieve those documents again. You can retrieve a list of identifiers for the failed documents by sending a query to the IDOL Error

Server. For more information about IDOL Error Server, refer to the *IDOL Error Server Technical Note*. For more information about the `synchronize` action, refer to the *SharePoint OData Connector Reference*.

Insert Information into the SharePoint Repository

The connector's `insert` fetch action inserts information into a SharePoint repository. To use the `insert` action, you must construct some XML that specifies how and where to add each item, and the information to insert. You must add the XML to the action as the value of the `insertXML` action parameter. The exact structure of the XML depends on the type of item that you want to insert.

The XML contained in the `insertXml` parameter must be URL encoded before being used in the action command. For example:

```
http://host:port/action=Fetch&FetchAction=Insert
                            &ConfigSection=MyTask
                            &InsertXML=[URL encoded XML]
```

The connector can insert items into the repository by ID or by path. Inserting items by ID is faster. To find the GUID of a site or list, or the ID of a list item, look in the metadata of documents that have been ingested from those items.

For more information about using the `insert` fetch action, refer to the *SharePoint OData Connector Reference*.

Insert a Site

To insert a site, your `insertXml` must include the following properties:

`TYPE`	The type of item to insert. To insert a site, specify `SITE`.
`TITLE`	The title to use for the site.
`URL`	The parent relative URL of the site.

You must then specify either (`SITEID` and `SITECOLLECTIONURL`) or (`INSERTFROMPATH` and `SITEURL`):

`INSERTFROMPATH`	To insert a site by path, set this to `TRUE`.
`SITECOLLECTIONURL`	The absolute URL of the site collection into which you want to insert the site.
`SITEID`	The GUID of the parent site.
`SITEURL`	The absolute URL of the parent site.

The following properties are optional:

`DESCRIPTION`	A description of the site.

For example, the value of your `insertXML` action parameter might look like this:

```
<insertXML>
  <insert>
```

```
    <reference>TEST_SITE</reference>
    <property name="TYPE" value="SITE"/>
    <property name="TITLE" value="My New Site"/>
    <property name="URL" value="MyNewSite"/>
    <property name="SITECOLLECTIONURL" value="http://sharepoint/SiteCollection/"/>
    <property name="SITEID" value="AF1F1E11526FFA45"/>
    <property name="DESCRIPTION" value="My Inserted Site"/>
  </insert>
</insertXML>
```

To insert a site by path, your XML might look like this:

```
<insertXML>
  <insert>
    <reference>TEST_SITE</reference>
    <property name="TYPE" value="SITE"/>
    <property name="TITLE" value="My New Site"/>
    <property name="URL" value="MyNewSite"/>
    <property name="SITEURL" value="http://sharepoint/SiteCollection/"/>
    <property name="INSERTFROMPATH" value="TRUE"/>
    <property name="DESCRIPTION" value="My Inserted Site"/>
  </insert>
</insertXML>
```

Insert a List

To insert a list, the `insertXml` must include the following properties.

TYPE   The type of item to insert. To insert a list, specify `LIST`.

TITLE   The title of the list.

You must then specify either (`SITEID` and `SITECOLLECTIONURL`) or (`INSERTFROMPATH` and `SITEURL`).

INSERTFROMPATH   To insert a list by path, set this property to `TRUE`.

SITECOLLECTIONURL   The absolute URL of the site collection to which you want to add the
                            list.

SITEID   The GUID of the site to which you want to add the list.

SITEURL   The absolute URL of the site to which you want to add the list.

The following properties are optional:

BASETEMPLATE   The list template type, for example "GenericList" or "DocumentLibrary". For a
                        list of acceptable types, refer to http://msdn.microsoft.com/en-
                        us/library/microsoft.sharepoint.client.listtemplatetype

DESCRIPTION   A description for the list.

ENABLEVERSIONING   To enable versioning for the list, set this parameter to `TRUE`.

ENABLEATTACHMENTS   To allow attachments in the list, set this parameter to TRUE.

You can optionally add metadata to the list. In the following field names, replace `[Prefix]` with the value of the `ItemFieldNamePrefix` parameter, as set in the configuration file.

`[Prefix]FIELDNAME`   The name of a column that you want to create.

`[Prefix]`
`[FieldName]_Title`   The title of the column.

`[Prefix]`
`[FieldName]_`
`FieldTypeKind`   The type of the column. For a list of acceptable types, refer to http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.client.fieldtype.ASPX

`[Prefix]`
`[FieldName]_`
`Description`   A description of the column.

`[Prefix]`
`[FieldName]_`
`Required`   A Boolean that specifies whether the column requires a value.

For example, the value of your `insertXML` action parameter might look like this:

```
<insertXML>
  <insert>
    <reference>TEST_LIST</reference>
    <property name="TYPE" value="LIST"/>
    <property name="TITLE" value="My New List"/>
    <property name="SITEID" value="E11526FFA45AF1F1"/>
    <property name="SITECOLLECTIONURL" value="http://sharepoint/SiteCollection/"/>
    <metadata name="MyPrefixFIELDNAME" value="Field1"/>
    <metadata name="MyPrefixField1_Title" value="MyField"/>
    <metadata name="MyPrefixField1_FieldTypeKind" value="Integer"/>
    <metadata name="MyPrefixField1_Description" value="My Field"/>
    <metadata name="MyPrefixField1_Required" value="True"/>
  </insert>
</insertXML>
```

To insert a list by path, your XML might look like this:

```
<insertXML>
  <insert>
    <reference>TEST_LIST</reference>
    <property name="TYPE" value="LIST"/>
    <property name="TITLE" value="My New List"/>
    <property name="SITEURL" value="http://sharepoint/SiteCollection/Site/"/>
    <property name="INSERTFROMPATH" value="TRUE"/>
  </insert>
</insertXML>
```

Insert a List Item

> **NOTE:**
> The connector can only insert list items into the root of a list.

To insert a list item, the `insertXml` must include the following properties.

TYPE   The type of item to insert. To insert a list item, specify **LISTITEM**.

TITLE   A name for the list item.

You must then specify either (`LISTID`, `SITEID`, and `SITECOLLECTIONURL`) or (`INSERTFROMPATH`, `SITEURL`, and `LISTNAME`).

INSERTFROMPATH   To insert a list item by path, set this to **TRUE**.

LISTID   The GUID of the list to which you want to add the list item.

LISTNAME   The name of the list to which you want to add the list item.

SITECOLLECTIONURL   The absolute URL of the site collection to which you want to add the list item.

SITEID   The GUID of the site to which you want to add the list item.

SITEURL   The absolute URL of the site to which you want to add the list item.

You can optionally add metadata to the list item.

[Prefix][FieldName]   Replace [`Prefix`] with the value of the `ItemFieldNamePrefix` parameter, as set in the configuration file. Replace [`FieldName`] with the name of the field that you want to add, and specify the value.

For example, the value of your `insertXML` action parameter might look like this:

```
<insertXML>
  <insert>
    <reference>TEST_LIST_ITEM</reference>
    <property name="TYPE" value="LISTITEM"/>
    <property name="TITLE" value="Title" />
    <property name="LISTID" value="26FFE115A45AF1F1"/>
    <property name="SITEID" value="E11526FFA45AF1F1"/>
    <property name="SITECOLLECTIONURL" value="http://sharepoint/SiteCollection/"/>
    <metadata name="MyPrefixMyIntegerField" value="123"/>
  </insert>
</insertXML>
```

To insert a list item by path, your XML might look like this:

```
<insertXML>
  <insert>
```

```
    <reference>TEST_LIST_ITEM </reference>
    <property name="TYPE" value="LISTITEM"/>
    <property name="TITLE" value="Title" />
    <property name="LISTNAME" value="MyList"/>
    <property name="SITEURL" value="http://sharepoint/SiteCollection/Site/"/>
    <property name="INSERTFROMPATH" value="TRUE"/>
    <metadata name="MyPrefixMyIntegerField" value="123"/>
  </insert>
</insertXML>
```

Insert a File

To insert a file, the `insertXml` must include the following properties.

TYPE   The type of item to insert. To insert a file, specify **FILE**.

PATH   The path of the file, including the filename, relative to the root of the list. The connector will
        create any folders that do not exist.

You must then specify either (`LISTID`, `SITEID`, and `SITECOLLECTIONURL`) or (`INSERTFROMPATH`,
`SITEURL`, and `LISTNAME`).

INSERTFROMPATH	To insert a file by path, set this to **TRUE**.
LISTID	The GUID of the list to which you want to add the file.
LISTNAME	The name of the list to which you want to add the file.
SITECOLLECTIONURL	The absolute URL of the site collection to which you want to add the file.
SITEID	The GUID of the site to which you want to add the file.
SITEURL	The absolute URL of the site to which you want to add the file.

For example, the value of your `insertXML` action parameter might look like this:

```
<insertXML>
  <insert>
    <reference>TEST_FILE</reference>
    <property name="TYPE" value="FILE"/>
    <property name="PATH" value="folder/subfolder/somefile.file"/>
    <property name="LISTID" value="26FFE115A45AF1F1"/>
    <property name="SITEID" value="E11526FFA45AF1F1"/>
    <property name="SITECOLLECTIONURL" value="http://sharepoint/SiteCollection/"/>
    <file>
      <type>file</type>
      <content>e:\my_files\myfile.doc</content>
    </file>
  </insert>
</insertXML>
```

To insert a file by path, your XML might look like this:

```
<insertXML>
  <insert>
    <reference>TEST_FILE</reference>
    <property name="TYPE" value="FILE"/>
    <property name="PATH" value="folder/subfolder/somefile.file"/>
    <property name="LISTNAME" value="MyList"/>
    <property name="SITEURL" value="http://sharepoint/SiteCollection/Site/"/>
    <property name="INSERTFROMPATH" value="TRUE"/>
    <file>
      <type>file</type>
      <content>e:\my_files\myfile.doc</content>
    </file>
  </insert>
</insertXML>
```

Insert an Attachment

To insert an attachment to a list item, the `insertXml` must include the following properties.

TYPE	The type of item to insert. To insert an attachment, specify **ATTACHMENT**.
FILENAME	The file name of the attachment.

You must then specify either (`LISTITEMID`, `LISTID`, `SITEID`, and `SITECOLLECTIONURL`) or (`INSERTFROMPATH`, `SITEURL`, `LISTNAME`, and `PATH`).

INSERTFROMPATH	To insert an attachment by path, set this to **TRUE**.
LISTID	The GUID of the list to which you want to add the attachment.
LISTITEMID	The integer ID of the list item to which you want to add the attachment.
LISTNAME	The name of the list to which you want to add the attachment.
PATH	The full path to the list item, including the list item name, relative to the root of the list.
SITECOLLECTIONURL	The absolute URL of the site collection to which you want to add the attachment.
SITEID	The GUID of the site to which you want to add the attachment.
SITEURL	The absolute URL of the site to which you want to add the attachment.

For example, the value of your `insertXML` action parameter might look like this:

```
<insertXML>
  <insert>
    <reference>TEST_ATTACHMENT</reference>
    <property name="TYPE" value="ATTACHMENT"/>
    <property name="FILENAME" value="myattachment.file"/>
```

```
    <property name="LISTITEMID" value="125"/>
    <property name="LISTID" value="26FFE115A45AF1F1"/>
    <property name="SITEID" value="E11526FFA45AF1F1"/>
    <property name="SITECOLLECTIONURL" value="http://sharepoint/SiteCollection/"/>
    <file>
      <type>file</type>
      <content>e:\my_files\myfile.doc</content>
    </file>
  </insert>
</insertXML>
```

To insert an attachment by path, your XML might look like this:

```
<insertXML>
  <insert>
    <reference>TEST_ATTACHMENT</reference>
    <property name="TYPE" value="ATTACHMENT"/>
    <property name="FILENAME" value="myattachment.file"/>
    <property name="PATH" value=" folder/subfolder/item1"/>
    <property name="LISTNAME" value="MyList"/>
    <property name="SITEURL" value="http://sharepoint/SiteCollection/Site/"/>
    <property name="INSERTFROMPATH" value="TRUE"/>
    <file>
      <type>content</type>
      <content>[The entire file base64 encoded]</content>
    </file>
  </insert>
</insertXML>
```

Insert a Social Post

To insert a post in a social feed, the `insertXml` must include the following properties:

TYPE   The type of item to insert. To insert a post, specify **SOCIALPOST**.

The following properties specify where to insert the post:

SOCIALTHREADID   The ID of the thread to add the post to. If you set SOCIALTHREADID and not
                 SITEFEEDURL, the post is added as a reply to the specified thread.

SITEFEEDURL   The URL of the feed to add the post to. If you set SITEFEEDURL and not
                 SOCIALTHREADID, the post is created in a new thread in the specified site feed.

If you set neither SOCIALTHREADID or SITEFEEDURL, the post is created in a new thread in the
authenticated user's feed (the user specified in the connector's configuration file).

The following metadata is required:

TEXT   The content to include in the new post.

The following metadata is optional:

`LINK`	The text (that you added to the `TEXT` field) to make into a link, followed by a comma, followed by the link URL. You can include any number of `LINK` metadata elements.
`ATTACHMENTURI`	The location (URI) of an attachment to include in the new post.

For example, the value of your `insertXML` action parameter might look like this:

```
<insertXML>
  <insert>
    <reference>[REF]</reference>
    <property name="TYPE" value="SOCIALPOST" />
    <property name="SITEFEEDURL"
value="http://SiteCollection.com/Site/newsfeed.aspx"/>
    <metadata name="TEXT" value="SomeContent"/>
    <metadata name="LINK" value="[TEXT],[URL]"/>
    <metadata name="ATTACHMENTURI" value="[URI]"/>
  </insert>
</insertXML>
```

Insert a Stub into the SharePoint Repository

Some applications move or archive documents. In place of the original document they can leave a stub, which points to the new location. If the original document is requested, an application instead finds the stub. The stub contains the name of a connector group that can access the document in the new location, and the document's new identifier. The application can then send this information to a Distributed Connector, which queries a suitable connector to retrieve the document from its new location.

The stub fetch action works in a similar way to the insert action. To insert a stub, you must construct some XML that specifies how and where to add the stub. You must add the XML to the action as the value of the `stubXML` action parameter. The exact structure of the XML depends on the information required to insert items into the repository.

You can insert a stub into SharePoint as one of the following types:

- `FILE`. When inserting the stub into a document library, the document library must support the "Link to a Document" content type.

- `LISTITEM`. The list must support the "Link" content type.

- `ATTACHMENT`.

The properties that you must set for inserting the stub are the same as for inserting a document with the insert action. For example, to insert a stub as a list item, you could use the following XML:

```
<stubXML>
  <stub>
    <property name="TYPE" value="LISTITEM"/>
    <property name="TITLE" value="Title" />
    <property name="INSERTFROMPATH" value="TRUE"/>
```

```
      <property name="SITEURL" value="http://sharepointhost/site/"/>
      <property name="LISTNAME" value="listname"/>
      <target>
        <uri>http://host/target</uri>
      </target>
    </stub>
  </stubXML>
```

For information the properties you must set, see Insert a File, on page 58, Insert a List Item, on page 57, or Insert an Attachment, on page 59. The `target` element contains information about the new location of the document. The URI could be the URI to a Distributed Connector.

The XML contained in the `stubXml` parameter must be URL encoded. For example:

```
http://host:port/action=Fetch&FetchAction=Stub
                            &ConfigSection=MyTask
                            &StubXML=[URL encoded XML]
```

For more information about using the `stub` fetch action, refer to the *SharePoint OData Connector Reference*.

Update Metadata of Items in SharePoint

The connector's `update` fetch action updates the metadata of items in the SharePoint repository.

You can update the following metadata:

- The values of metadata fields for list items (including files in document libraries).

To use the `update` action, you must construct some XML that specifies the identifiers of the items to update, and provides the new values for any metadata fields that you want to change.

You must add the XML to the action as the value of the `identifiersXML` action parameter. The XML must be URL-encoded before being used in the action command. For example:

```
http://host:port/action=Fetch&FetchAction=update
                                &IdentifiersXML=URL-encoded XML
```

For more information about how to construct the XML, see Construct XML to Update Metadata, below.

Construct XML to Update Metadata

To use the `update` fetch action, you must construct some XML that specifies the identifiers of the items to update, and provides the new values for any metadata fields that you want to change.

The field names specified in the XML should be the names or titles of the fields you want to set in SharePoint. Depending on your connector configuration (see the `ItemFieldNamePrefix` parameter), these might have a prefix in documents retrieved by the synchronize action. Field names passed to the update fetch action must not include this prefix.

The following XML would update the title, modified date, and creation date for an item with the specified identifier. In the identifier `value` attribute, replace "..." with the document identifier of the item you want to update. A document identifier can be found in the `AUTN_IDENTIFIER` field of an indexed document.

```
<identifiersXML>
  <identifier value="...">
    <metadata name="Created" value="2015-02-17T16:01Z"/>
    <metadata name="Modified" value="2015-02-17T16:04Z"/>
    <metadata name="Title" value="New Title"/>
  </identifier>
</identifiersXML>
```

You can update the metadata of several items by including more than one `identifier` element:

```
<identifiersXML>
  <identifier value="...">
    <metadata name="..." value="..."/>
  </identifier>
 <identifier value="...">
    <metadata name="..." value="..."/>
  </identifier>
</identifiersXML>
```

> **TIP:**
> You can update the metadata of any field, but you must specify the value in a format that is accepted by SharePoint.

Unless you explicitly set the modified date, SharePoint automatically updates the date to the current time.

> **NOTE:**
> Updating the created or modified date of an item in a versioned list or document library might result in a new version of the item being created.

Delete Items from the SharePoint Repository

The connector's `delete` fetch action deletes items from the SharePoint repository.

You can use this action to delete:

- A site

  > **CAUTION:**
  > If you delete a site, all of the sub-sites are also deleted.

- A list

- A list item, file, or file version

- An attachment

For more information about using the `delete` fetch action, refer to the *SharePoint OData Connector Reference*.

Reset the Connector

When the connector runs the `synchronize` action, it updates a datastore file that stores information about the data retrieved from the repository. The next time the connector runs the `synchronize` action, it retrieves only data that is new or has been modified. The connector can also determine whether files or records have been deleted, so that related documents can be removed from the IDOL index.

When you are configuring the connector and you make a change to the configuration, you might want to purge all information from the datastore so that the connector retrieves all of your data again.

To purge the datastore for a fetch task

- To purge the datastore for a fetch task, use the `PurgeDatastore` action. Specify the name of the task as the value of the `section` parameter, for example:

  `/action=PurgeDatastore&section=MyTask`

In some cases you might want to delete all queued actions.

To delete the actions queue

1. Stop the connector.
2. Delete the `actions` folder. This ensures that information about incomplete and queued actions is deleted.
3. Restart the connector.

Troubleshoot the Connector

This section describes how to troubleshoot common problems that might occur when you set up the SharePoint OData Connector.

Error: The length of the URL for this request exceeds the configured maxUrlLength value

This error can occur when the Internet Information Services (IIS) web server refuses to accept long URLs. To resolve this issue, update the `web.config` configuration file for the SharePoint site in IIS. For example:

```
<configuration>
    ...
    <system.web>
        ...
        <httpRuntime ... maxUrlLength="1024" ... />
        ...
    </system.web>
```

```
        ...
    </configuration>
```

Chapter 7: SharePoint Mapped Security

This section describes how to set up mapped security for information that is extracted from a SharePoint repository.

Introduction

The SharePoint mapped security architecture includes the following components:
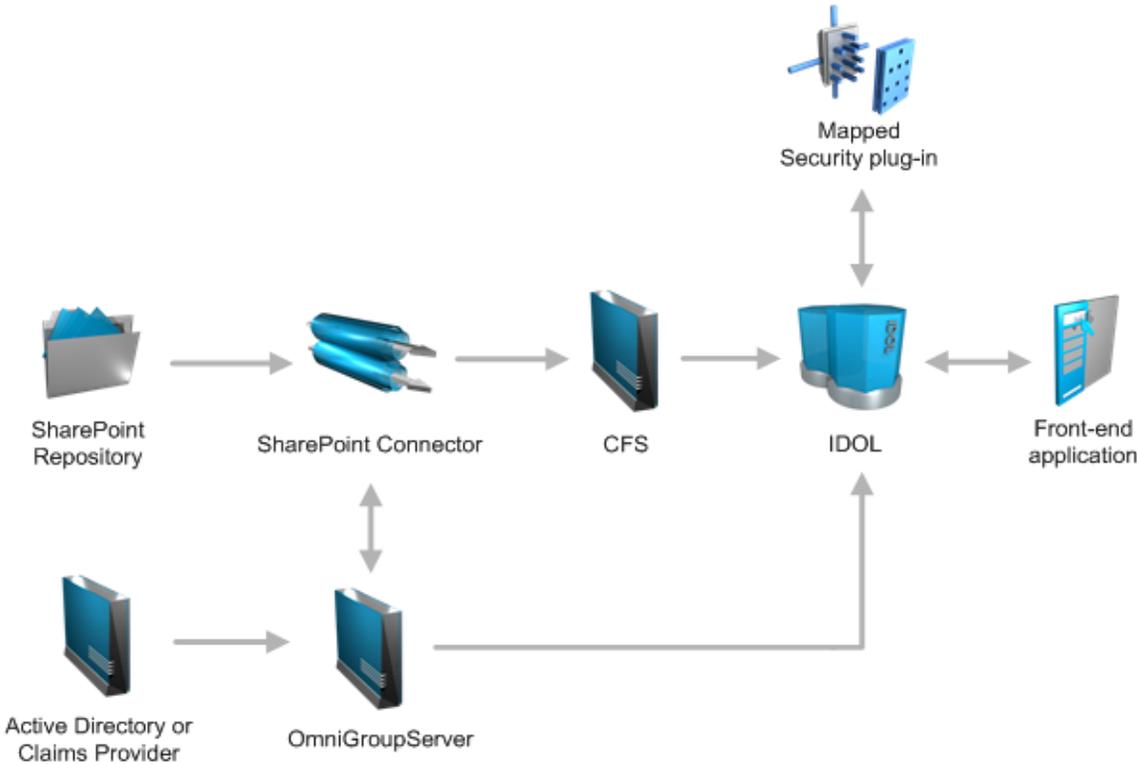
Items in the SharePoint repository have an Access Control List (ACL) that lists the users and groups who are permitted, and are not permitted, to view the item. IDOL needs the ACL to determine whether a user can view a document that is returned as a result to a query. When the SharePoint Connector retrieves information from the repository, it extracts the Access Control List (ACL) for the item and writes it to a document field. Each time the connector synchronizes with the repository, it extracts updated ACLs.

On-premise SharePoint instances are typically linked with Active Directory, and administrators can assign permissions to Active Directory users, Active Directory groups, or groups that have been defined within

SharePoint site collections (SharePoint groups). For example, a user might have permission to view a document in SharePoint because they are a member of an NT domain group.

In SharePoint Online permissions can be assigned to users and groups. These users and groups could be managed in SharePoint Online, or in an on-premise Active Directory.

In either SharePoint on-premise or SharePoint Online, claims-based authentication can be used, meaning that users and groups could be managed from a claims provider other than Active Directory.

Therefore IDOL must also consider the groups that a user belongs to. A user might not be permitted to view a document, but they could be a member of a group that has permission. This means that IDOL also requires user and group information.

OmniGroupServer can retrieve users and groups from Active Directory or a claims provider, but it must request SharePoint groups through the SharePoint Connector. OmniGroupServer must be configured to extract, combine, and store the group information from both sources.

When a user logs on to a front-end application, the application requests the user's security information and group memberships from IDOL server. IDOL returns a token containing the information. The front-end application includes this token in all queries it sends to IDOL.

After a user submits a query, IDOL sends the result documents and the user's security token to the Mapped Security plug-in. The Mapped Security plug-in compares the user's security information and group memberships to each document's ACL. The plug-in determines which documents the user is permitted to view and returns the results. IDOL server then sends only the documents that the user is permitted to view to the front-end application.

Set up SharePoint Mapped Security

To use mapped security to protect information that was extracted from a SharePoint repository, set up the following components:

- IDOL Server. You must set up the IDOL Content component to identify the security type associated with each document. You must also configure the IDOL Community component, so that IDOL sends user and group information to the front-end application when a user logs on. For information about how to set up IDOL Server, refer to the *IDOL Document Security Administration Guide*.

- SharePoint OData Connector. You must set up the SharePoint OData Connector to include security information (Access Control Lists) in the documents that are indexed into IDOL server. You must also add a field to each document that identifies the security type. For information about how to do this, see Retrieve and Index Access Control Lists, on the next page.

- OmniGroupServer. You must set up OmniGroupServer to retrieve, and then combine, group information from SharePoint and either Active Directory or a claims provider. This results in three repositories in OmniGroupServer, but only the repository that contains the combined information should be queried by IDOL Community to populate user security info strings. OmniGroupServer retrieves SharePoint groups by sending the `SynchronizeGroups` action to the SharePoint connector. OmniGroupServer extracts NT security information directly from Active Directory. For information about how to configure OmniGroupServer, see Retrieve Security Group Information using OmniGroupServer, on page 69.

- A front-end application for querying IDOL Server.

Retrieve and Index Access Control Lists

To configure the connector to retrieve and index Access Control Lists (ACLs), follow these steps.

To retrieve and index Access Control Lists

1. If the connector is running, stop the connector.
2. Open the connector's configuration file.
3. Find the [*TaskName*] section in which you configured a fetch task to retrieve information from SharePoint.
4. Set the following parameters:

   MappedSecurity

   Set `MappedSecurity` to `true` so that the connector extracts Access Control Lists (ACLs) and adds these to documents.

   IngestActions

   Set this parameter to add a document field to each document that identifies the security type.

   > **NOTE:**
   > The field name and value that you specify must match the name and value you used to identify SharePoint security in your IDOL Server configuration file.

   IncludeProviderNameInACLs

   (Optional) This parameter specifies whether to include the name of the membership or role provider, from which a user or group originates, in the ACL. If your SharePoint repository is configured to use claims-based authentication with multiple claims providers, set this parameter to `true`.

   UseEmailAsGroupName

   (Optional) This parameter specifies whether to add the e-mail address of a group to a document ACL, instead of the group name. If you are retrieving documents from SharePoint Online, set this parameter to `true`.

   ```
   [SharePointOnPremise]
   ...
   MappedSecurity=TRUE
   UseEmailAsGroupName=FALSE
   IngestActions=META:SecurityType=SharePoint

   [SharePointOnline]
   ...
   MappedSecurity=TRUE
   UseEmailAsGroupName=TRUE
   IngestActions=META:SecurityType=SharePointOnline
   ```

5. Save and close the configuration file.

Retrieve Security Group Information using OmniGroupServer

To ensure that users are allowed to view documents that you have retrieved from SharePoint and indexed into IDOL Server, you must:

- retrieve the security group information from the SharePoint repository.
- retrieve the security group information from Active Directory or a claims provider (depending on how your SharePoint installation is configured).
- combine the security groups into a single repository in OmniGroupServer.

Micro Focus recommends that you schedule all of these tasks using OmniGroupServer. When the tasks are scheduled through OmniGroupServer, no changes need to be made to the connector's configuration file. You can also schedule the tasks to occur in the correct order (the combine operation must occur after the group information has been retrieved).

Configure the Connector

The connector can synchronize security groups from SharePoint using the same credentials that are required for a standard fetch task. If you have already configured a fetch task, or specified the necessary parameters in the `[FetchTasks]` or `[Default]` section of the configuration file, no additional configuration is required.

For example, set the following parameters in your connector configuration file:

```
[SharePointOnPremise]
SharepointUrlType=SiteCollection
SharepointUrl=http://sharepoint.domain.com/sites/site/
Username=user
Password=*****
Domain=domain
MappedSecurity=TRUE
UseEmailAsGroupName=FALSE

[SharePointOnline]
SharepointOnline=TRUE
SharepointUrlType=SiteCollection
SharepointUrl=https://mysharepoint-sites.sharepoint.com
SharepointAdminUrl=https://mysharepoint-admin.sharepoint.com
SharepointMySiteUrl=https://mysharepoint-my.sharepoint.com
Username=user
Password=*****
MappedSecurity=TRUE
UseEmailAsGroupName=TRUE
```

Configure OmniGroupServer (Active Directory Authentication)

To retrieve and combine the security groups from SharePoint and Active Directory, create the following tasks in the OmniGroupServer configuration file.

- A task to retrieve the security groups from the SharePoint repository. This task sends the `SynchronizeGroups` fetch action to the SharePoint connector, according to the schedule that you configure.

- A task to retrieve the security groups from Active Directory.

- A task to combine the information into a single repository in OmniGroupServer. The combine operation must run after the security information has been retrieved.

To retrieve and combine security groups

1. Open the OmniGroupServer configuration file.

2. In the `[Repositories]` section, create three repositories, one for the SharePoint groups, one for the Active Directory groups retrieved through LDAP, and another to combine the information. For example:

```
[Repositories]
GroupServerDefaultRepositories=Combine
Number=3
0=SharePointOnline
1=LDAP
2=Combine
```

3. In the section that you created to retrieve the SharePoint groups, create a task to extract the information from SharePoint. You can use the following configuration parameters (for a complete list of configuration parameters, refer to the *OmniGroupServer Reference*):

GroupServerJobType The type of task that OmniGroupServer must run. To retrieve SharePoint groups, set this parameter to **Connector**. This instructs OmniGroupServer to send the `SynchronizeGroups` fetch action to the connector.

ConnectorHost The host name or IP address of the machine that hosts the SharePoint connector.

ConnectorPort The ACI port of the connector.

ConnectorTask The name of a fetch task in the connector's configuration file that contains the information and credentials required to connect to the SharePoint repository.

For example:

```
[SharepointOnline]
GroupServerJobType=Connector
ConnectorHost=localhost
```

```
ConnectorPort=7024
ConnectorTask=SharePointOnline
```

4. In the section that you created to retrieve the Active Directory groups, configure a task to extract the information from the directory using LDAP. You can use the following configuration parameters (for a complete list of configuration parameters, refer to the *OmniGroupServer Reference*).

`GroupServerLibrary`	The full path (including the file name) to the library file that allows the group server to access the repository. Use the LDAP group server library.
`LDAPServer`	The host name or IP address of the machine that hosts the LDAP directory.
`LDAPPort`	The port to use to access the LDAP directory.
`LDAPBase`	The distinguished name of the search base.
`LDAPType`	The type of LDAP server (for example, Microsoft Active Directory).
`LDAPSecurityType`	The type of security to use when communicating with the LDAP server (for example, `SSL` or `TLS`).
`LDAPBindMethod`	The type of authentication to use to access the LDAP directory. To log on as the same user that is running OmniGroupServer, set this parameter to `NEGOTIATE`.
`KeyUserName`	(Optional) The name of the attribute from which to extract the user name. If you are retrieving documents from SharePoint Online and users from a local Active Directory, set this parameter so that OmniGroupServer extracts an e-mail address rather than a user name.
`KeyGroupName`	(Optional) The name of the attribute from which to extract the group name. If you are retrieving documents from SharePoint Online and groups from a local Active Directory, set this parameter so that OmniGroupServer extracts an e-mail address rather than a group name.

For example:

```
[LDAP]
GroupServerLibrary=ogs_ldap.dll
LDAPServer=myLDAPserver
LDAPPort=636
LDAPBase=DC=DOMAIN,DC=COM
LDAPType=MAD
LDAPSecurityType=SSL
LDAPBindMethod=NEGOTIATE
KeyUserName=mail
KeyGroupName=mail
```

5. In the section that you created for combining the security groups, configure a task to combine the group information. You can use the following configuration parameters (for a complete list of configuration parameters, refer to the *OmniGroupServer Reference*):

GroupServerJobType	The type of task that OmniGroupServer must run. Set this parameter to `Combine`.
GroupServerSections	The names of the repositories in the configuration file that you want to merge.
GroupServerStartDelaySecs	The number of seconds to wait before starting the task.
	It is important to set this parameter so that the combine operation does not start until the security groups have been retrieved from the SharePoint repository and Active directory. This ensures that the combine operation uses the latest information.
	The delay that you specify only has to ensure that the other jobs start first.

For example:

```
[Combine]
GroupServerJobType=Combine
GroupServerSections=SharepointOnline,LDAP
GroupServerStartDelaySecs=10
```

6. (Optional) You can set further parameters to define the schedule for the tasks. To run all of the tasks on the same schedule, set these parameters in the `[Default]` section. To run a task on a different schedule, set these parameters in the task section.

GroupServerStartTime	The time when a task starts.
GroupServerRepeatSecs	The number of seconds that should elapse before the Group Server repeats a task.

For example:

```
[Default]
GroupServerStartTime=12:00
GroupServerRepeatSecs=3600
```

7. Save and close the OmniGroupServer configuration file.

Configure OmniGroupServer (Claims-Based Authentication)

To retrieve and combine the security groups from SharePoint and a claims provider other than Active Directory, create the following tasks in the OmniGroupServer configuration file.

- A task to retrieve the security groups from the SharePoint repository. This task sends the `SynchronizeGroups` fetch action to the SharePoint connector, according to the schedule that you configure.

- A task to retrieve the security groups from the claims provider.

- A task to combine the information into a single repository in OmniGroupServer. The combine operation must run after the security information has been retrieved.

To retrieve and combine security groups

1. Open the OmniGroupServer configuration file.

2. In the `[Repositories]` section, create three repositories - one for the SharePoint groups, one for groups retrieved from the claims provider, and another to combine the information. For example:

```
[Repositories]
GroupServerDefaultRepositories=Combine
Number=3
0=SharePointOnPremise
1=Claims
2=Combine
```

3. In the section that you created to retrieve the SharePoint groups, create a task to extract the information from SharePoint. You can use the following configuration parameters (for a complete list of configuration parameters, refer to the *OmniGroupServer Reference*):

GroupServerJobType	The type of task that OmniGroupServer must run. To retrieve SharePoint groups, set this parameter to **Connector**. This instructs OmniGroupServer to send the SynchronizeGroups fetch action to the connector.
ConnectorHost	The host name or IP address of the machine that hosts the SharePoint connector.
ConnectorPort	The ACI port of the connector.
ConnectorTask	The name of a fetch task in the connector's configuration file that contains the information and credentials required to connect to the SharePoint repository.

For example:

```
[SharepointOnPremise]
GroupServerJobType=Connector
ConnectorHost=localhost
ConnectorPort=7024
ConnectorTask=SharePointOnPremise
```

4. In the section that you created to retrieve the users and groups from the claims provider, configure a task to extract the information. The following example shows how you might extract information from an ASP.NET claims provider using ODBC. For a complete list of configuration parameters that you can use, refer to the *OmniGroupServer Reference*.

```
[Claims]
GroupServerLibrary=ogs_dbodbc
ConnectionString=DSN=SP2013FBA;UID=user;PWD=password;Database=DatabaseName
ExecuteSection0=Database_GetUsers
ExecuteSection1=Database_GetGroups
ExecuteSection2=Database_GetGroupMembers
```

```
GroupServerOpApplyTo0=GROUP
GroupServerOp0=PREPEND
GroupServerOpParam0=SQL_ROLE_PROVIDER:

GroupServerOpApplyTo1=USER
GroupServerOp1=PREPEND
GroupServerOpParam1=SQL_MEMBERSHIP_PROVIDER:

[Database_GetUsers]
SQLStatement=SELECT DISTINCT UserName FROM dbo.aspnet_Users
AddType=USER
UserColumn=UserName

[Database_GetGroups]
SQLStatement=SELECT DISTINCT RoleName FROM dbo.aspnet_Roles
AddType=GROUP
GroupColumn=RoleName

[Database_GetGroupMembers]
SQLStatement=SELECT U.UserName,R.RoleName FROM FBA_USER_DB.dbo.aspnet_Users U
    JOIN FBA_USER_DB.dbo.aspnet_UsersInRoles UIR ON U.UserId = UIR.UserId
    JOIN FBA_USER_DB.dbo.aspnet_Roles R ON R.RoleId = UIR.RoleId
AddType=USERGROUP
UserColumn=UserName
GroupColumn=RoleName
```

> **NOTE:**
> The value of the `SQLStatement` parameter must be on one line.

5. In the section that you created for combining the security groups, configure a task to combine the group information. You can use the following configuration parameters (for a complete list of configuration parameters, refer to the *OmniGroupServer Reference*):

`GroupServerJobType`	The type of task that OmniGroupServer must run. Set this parameter to **Combine**.
`GroupServerSections`	The names of the repositories in the configuration file that you want to merge.
`GroupServerStartDelaySecs`	The number of seconds to wait before starting the task.
	It is important to set this parameter so that the combine operation does not start until the security groups have been retrieved. This ensures that the combine operation uses the latest information.
	The delay that you specify only has to ensure that the other jobs start first.

For example:

```
[Combine]
GroupServerJobType=Combine
GroupServerSections=SharepointOnPremise,Claims
GroupServerStartDelaySecs=10
```

6. (Optional) You can set further parameters to define the schedule for the tasks. To run all of the tasks on the same schedule, set these parameters in the `[Default]` section. To run a task on a different schedule, set these parameters in the task section.

`GroupServerStartTime`	The time when a task starts.
`GroupServerRepeatSecs`	The number of seconds that should elapse before the Group Server repeats a task.

For example:

```
[Default]
GroupServerStartTime=12:00
GroupServerRepeatSecs=3600
```

7. Save and close the OmniGroupServer configuration file.

Chapter 8: Manipulate Documents

This section describes how to manipulate documents that are created by the connector and sent for ingestion.

Introduction

IDOL Connectors retrieve data from repositories and create documents that are sent to Connector Framework Server, another connector, or Haven OnDemand. You might want to manipulate the documents that are created. For example, you can:

- Add or modify document fields, to change the information that is indexed into IDOL Server or Haven OnDemand.
- Add fields to a document to customize the way the document is processed by CFS.
- Convert information into another format so that it can be inserted into another repository by a connector that supports the `Insert` action.

When a connector sends documents to CFS, the documents only contain metadata extracted from the repository by the connector (for example, the location of the original files). To modify data extracted by KeyView, you must modify the documents using CFS. For information about how to manipulate documents with CFS, refer to the *Connector Framework Server Administration Guide*.

Add a Field to Documents using an Ingest Action

To add a field to all documents retrieved by a fetch task, or all documents sent for ingestion, you can use an Ingest Action.

> **NOTE:**
> To add a field only to selected documents, use a Lua script (see Run Lua Scripts, on page 83). For an example Lua script that demonstrates how to add a field to a document, see Add a Field to a Document, on page 86.

To add a field to documents using an Ingest Action

1. Open the connector's configuration file.
2. Find one of the following sections in the configuration file:

- To add the field to all documents retrieved by a specific fetch task, find the [*TaskName*] section.

- To add a field to all documents that are sent for ingestion, find the [Ingestion] section.

> **NOTE:**
> If you set the IngestActions parameter in a [TaskName] section, the connector does not run any IngestActions set in the [Ingestion] section for documents retrieved by that task.

3. Use the IngestActions parameter to specify the name of the field to add, and the field value. For example, to add a field named AUTN_NO_EXTRACT, with the value SET, type:

   IngestActions0=META:AUTN_NO_EXTRACT=SET

4. Save and close the configuration file.

Customize Document Processing

You can add the following fields to a document to control how the document is processed by CFS. Unless stated otherwise, you can add the fields with any value.

AUTN_FILTER_META_ONLY

Prevents KeyView extracting file content from a file. KeyView only extracts metadata and adds this information to the document.

AUTN_NO_FILTER

Prevents KeyView extracting file content and metadata from a file. You can use this field if you do not want to extract text from certain file types.

AUTN_NO_EXTRACT

Prevents KeyView extracting subfiles. You can use this field to prevent KeyView extracting the contents of ZIP archives and other container files.

AUTN_NEEDS_MEDIA_SERVER_ANALYSIS

Identifies media files (images, video, and documents such as PDF files that contain embedded images) that you want to send to Media Server for analysis, using a MediaServerAnalysis import task. You do not need to add this field if you are using a Lua script to run media analysis. For more information about running analysis on media, refer to the *Connector Framework Server Administration Guide*.

AUTN_NEEDS_TRANSCRIPTION

Identifies audio and video assets that you want to send to an IDOL Speech Server for speech-to-text processing, using an IdolSpeech import task. You do not need to add this field if you are using a Lua script to run speech-to-text. For more information about running speech-to-text on documents, refer to the *Connector Framework Server Administration Guide*.

AUTN_FORMAT_CORRECT_FOR_TRANSCRIPTION

To bypass the transcoding step of an IdolSpeech import task, add the field AUTN_FORMAT_CORRECT_ FOR_TRANSCRIPTION. Documents that have this field are not sent to a Transcode Server. For more

information about the `IdolSpeech` task, refer to the *Connector Framework Server Administration Guide*.

AUTN_AUDIO_LANGUAGE

To bypass the language identification step of an `IdolSpeech` import task add the field `AUTN_AUDIO_LANGUAGE`. The value of the field must be the name of the IDOL Speech Server language pack to use for extracting speech. Documents that have this field are not sent to the IDOL Speech Server for language identification. For more information about the `IdolSpeech` task, refer to the *Connector Framework Server Administration Guide*.

Standardize Field Names

Field standardization modifies documents so that they have a consistent structure and consistent field names. You can use field standardization so that documents indexed into IDOL through different connectors use the same fields to store the same type of information.

For example, documents created by the File System Connector can have a field named `FILEOWNER`. Documents created by the Documentum Connector can have a field named `owner_name`. Both of these fields store the name of the person who owns a file. Field standardization renames the fields so that they have the same name.

Field standardization only modifies fields that are specified in a dictionary, which is defined in XML format. A standard dictionary, named `dictionary.xml`, is supplied in the installation folder of every connector. If a connector does not have any entries in the dictionary, field standardization has no effect.

Configure Field Standardization

IDOL Connectors have several configuration parameters that control field standardization. All of these are set in the `[Connector]` section of the configuration file:

- `EnableFieldNameStandardization` specifies whether to run field standardization.
- `FieldNameDictionaryPath` specifies the path of the dictionary file to use.
- `FieldNameDictionaryNode` specifies the rules to use. The default value for this parameter matches the name of the connector, and Micro Focus recommends that you do not change it. This prevents one connector running field standardization rules that are intended for another.

To configure field standardization, use the following procedure.

> **NOTE:**
> You can also configure CFS to run field standardization. To standardize all field names, you must run field standardization from both the connector and CFS.

To enable field standardization

1. Stop the connector.
2. Open the connector's configuration file.
3. In the `[Connector]` section, set the following parameters:

EnableFieldNameStandardization	A Boolean value that specifies whether to enable field standardization. Set this parameter to `true`.
FieldNameDictionaryPath	The path to the dictionary file that contains the rules to use to standardize documents. A standard dictionary is included with the connector and is named `dictionary.xml`.

For example:

```
[Connector]
EnableFieldNameStandardization=true
FieldNameDictionaryPath=dictionary.xml
```

4. Save the configuration file and restart the connector.

Customize Field Standardization

Field standardization modifies documents so that they have a consistent structure and consistent field names. You can use field standardization so that documents indexed into IDOL through different connectors use the same fields to store the same type of information. Field standardization only modifies fields that are specified in a dictionary, which is defined in XML format. A standard dictionary, named `dictionary.xml`, is supplied in the installation folder of every connector.

In most cases you should not need to modify the standard dictionary, but you can modify it to suit your requirements or create dictionaries for different purposes. By modifying the dictionary, you can configure the connector to apply rules that modify documents before they are ingested. For example, you can move fields, delete fields, or change the format of field values.

The following examples demonstrate how to perform some operations with field standardization.

The following rule renames the field `Author` to `DOCUMENT_METADATA_AUTHOR_STRING`. This rule applies to all components that run field standardization and applies to all documents.

```
<FieldStandardization>
    <Field name="Author">
        <Move name="DOCUMENT_METADATA_AUTHOR_STRING"/>
    </Field>
</FieldStandardization>
```

The following rule demonstrates how to use the `Delete` operation. This rule instructs CFS to remove the field `KeyviewVersion` from all documents (the `Product` element with the attribute `key="ConnectorFrameWork"` ensures that this rule is run only by CFS).

```
<FieldStandardization>
    <Product key="ConnectorFrameWork">
        <Field name="KeyviewVersion">
            <Delete/>
        </Field>
    </Product>
</FieldStandardization>
```

There are several ways to select fields to process using the `Field` element.

Field element attribute	Description	Example

name	Select a field where the field name matches a fixed value.	Select the field `MyField`:
		
		`<Field name="MyField">`
		   `...`
		`</Field>`
		
		Select the field `Subfield`, which is a subfield of `MyField`:
		
		`<Field name="MyField">`
		   `<Field name="Subfield">`
		     `...`
		   `</Field>`
		`</Field>`

path	Select a field where its path matches a fixed value.	Select the field `Subfield`, which is a subfield of `MyField`.
		
		`<Field path="MyField/Subfield">`
		   `...`
		`</Field>`

nameRegex	Select all fields at the current depth where the field name matches a regular expression.	In this case the field name must begin with the word `File`:
		
		`<Field nameRegex="File.*">`
		   `...`
		`</Field>`

pathRegex	Select all fields where the path of the field matches a regular expression.
	
	This operation can be inefficient because every metadata field must be checked. If possible, select the fields to process another way.
		This example selects all subfields of `MyField`.
		
		`<Field pathRegex="MyField/[^/]*">`
		   `...`
		`</Field>`
		
		This approach would be more efficient:
		
		`<Field name="MyField">`
		   `<Field nameRegex=".*">`
		     `...`
		   `</Field>`
		`</Field>`

You can also limit the fields that are processed based on their value, by using one of the following:

Field element attribute	Description	Example
matches	Process a field if its value matches a fixed value.	Process a field named `MyField`, if its value matches `abc`.

		`<Field name="MyField" matches="abc">`
		`   ...`
		`</Field>`

`matchesRegex`	Process a field if its entire value matches a regular expression.	Process a field named `MyField`, if its value matches one or more digits.
		
		`<Field name="MyField" matchesRegex="\d+">`
		`   ...`
		`</Field>`

`containsRegex`	Process a field if its value contains a match to a regular expression.	Process a field named `MyField` if its value contains three consecutive digits.
		
		`<Field name="MyField" containsRegex="\d{3}">`
		`   ...`
		`</Field>`

The following rule deletes every field or subfield where the name of the field or subfield begins with `temp`.

```
<FieldStandardization>
    <Field pathRegex="(.*/)?temp[^/]*">
        <Delete/>
    </Field>
</FieldStandardization>
```

The following rule instructs CFS to rename the field `Author` to `DOCUMENT_METADATA_AUTHOR_STRING`, but only when the document contains a field named `DocumentType` with the value `230` (the KeyView format code for a PDF file).

```
<FieldStandardization>
    <Product key="ConnectorFrameWork">
        <IfField name="DocumentType" matches="230"> <!-- PDF -->
            <Field name="Author">
                <Move name="DOCUMENT_METADATA_AUTHOR_STRING"/>
            </Field>
        </IfField>
    </Product>
</FieldStandardization>
```

> **TIP:**
> In this example, the `IfField` element is used to check the value of the `DocumentType` field. The `IfField` element does not change the current position in the document. If you used the `Field` element, field standardization would attempt to find an `Author` field that is a subfield of `DocumentType`, instead of finding the `Author` field at the root of the document.

The following rules demonstrate how to use the `ValueFormat` operation to change the format of dates. The only format that you can convert date values into is the IDOL AUTNDATE format. The first rule transforms the value of a field named `CreatedDate`. The second rule transforms the value of an attribute named `Created`, on a field named `Date`.

```
<FieldStandardization>
    <Field name="CreatedDate">
        <ValueFormat type="autndate" format="YYYY-SHORTMONTH-DD HH:NN:SS"/>
    </Field>
    <Field name="Date">
        <Attribute name="Created">
            <ValueFormat type="autndate" format="YYYY-SHORTMONTH-DD HH:NN:SS"/>
        </Attribute>
    </Field>
</FieldStandardization>
```

As demonstrated by this example, you can select field attributes to process in a similar way to selecting fields.

You must select attributes using either a fixed name or a regular expression:

Select a field attribute by name	`<Attribute name="MyAttribute">`
Select attributes that match a regular expression	`<Attribute nameRegex=".*">`

You can then add a restriction to limit the attributes that are processed:

Process an attribute only if its value matches a fixed value	`<Attribute name="MyAttribute" matches="abc">`
Process an attribute only if its value matches a regular expression	`<Attribute name="MyAttribute" matchesRegex=".*">`
Process an attribute only if its value contains a match to a regular expression	`<Attribute name="MyAttribute" containsRegex="\w+">`

The following rule moves all of the attributes of a field to sub fields, if the parent field has no value. The `id` attribute on the first `Field` element provides a name to a matching field so that it can be referred to by later operations. The `GetName` and `GetValue` operations save the name and value of a selected field or attribute (in this case an attribute) into variables (in this case `$'name'` and `$'value'`) which can be used by later operations. The `AddField` operation uses the variables to add a new field at the selected location (the field identified by `id="parent"`).

```
<FieldStandardization>
    <Field pathRegex=".*" matches="" id="parent">
        <Attribute nameRegex=".*">
            <GetName var="name"/>
            <GetValue var="value"/>
            <Field fieldId="parent">
                <AddField name="$'name'" value="$'value'"/>
            </Field>
            <Delete/>
```

```
            </Attribute>
        </Field>
</FieldStandardization>
```

The following rule demonstrates how to move all of the subfields of `UnwantedParentField` to the root of the document, and then delete the field `UnwantedParentField`.

```
<FieldStandardization id="root">
    <Product key="MyConnector">
        <Field name="UnwantedParentField">
            <Field nameRegex=".*">
                <Move destId="root"/>
            </Field>
            <Delete/>
        </Field>
    </Product>
</FieldStandardization>
```

Run Lua Scripts

IDOL Connectors can run custom scripts written in Lua, an embedded scripting language. You can use Lua scripts to process documents that are created by a connector, before they are sent to CFS and indexed into IDOL Server. For example, you can:

- Add or modify document fields.
- Manipulate the information that is indexed into IDOL.
- Call out to an external service, for example to alert a user.

There might be occasions when you do not want to send documents to a CFS. For example, you might use the `Collect` action to retrieve documents from one repository and then insert them into another. You can use a Lua script to transform the documents from the source repository so that they can be accepted by the destination repository.

To run a Lua script from a connector, use one of the following methods:

- Set the `IngestActions` configuration parameter in the connector's configuration file. For information about how to do this, see Run a Lua Script using an Ingest Action, on page 85. The connector runs ingest actions on documents before they are sent for ingestion.
- Set the `IngestActions` action parameter when using the `Synchronize` action.
- Set the `InsertActions` configuration parameter in the connector's configuration file. The connector runs insert actions on documents before they are inserted into a repository.
- Set the `CollectActions` action parameter when using the `Collect` action.

Write a Lua Script

A Lua script that is run from a connector must have the following structure:

```
function handler(config, document, params)
        ...
end
```

The `handler` function is called for each document and is passed the following arguments:

Argument	Description
config	A `LuaConfig` object that you can use to retrieve the values of configuration parameters from the connector's configuration file.
document	A `LuaDocument` object. The document object is an internal representation of the document being processed. Modifying this object changes the document.
params	The `params` argument is a table that contains additional information provided by the connector: • **TYPE**. The type of task being performed. The possible values are `ADD`, `UPDATE`, `DELETE`, or `COLLECT`. • **SECTION**. The name of the section in the configuration file that contains configuration parameters for the task. • **FILENAME**. The document filename. The Lua script can modify this file, but must not delete it. • **OWNFILE**. Indicates whether the connector (and CFS) has ownership of the file. A value of `true` means that CFS deletes the file after it has been processed.

The following script demonstrates how you can use the `config` and `params` arguments:

```
function handler(config, document, params)
        -- Write all of the additional information to a log file
        for k,v in pairs(params) do
                log("logfile.txt", k..": "..tostring(v))
        end

        -- The following lines set variables from the params argument
        type = params["TYPE"]
        section = params["SECTION"]
        filename = params["FILENAME"]

        -- Read a configuration parameter from the configuration file
        -- If the parameter is not set, "DefaultValue" is returned
        val = config:getValue(section, "Parameter", "DefaultValue")

        -- If the document is not being deleted, set the field FieldName
        -- to the value of the configuration parameter
        if type ~= "DELETE" then
                document:setFieldValue("FieldName", val)
        end

        -- If the document has a file (that is, not just metadata),
        -- copy the file to a new location and write a stub idx file
```

```
        -- containing the metadata.
        if filename ~= "" then
                copytofilename = "./out/"..create_uuid(filename)
                copy_file(filename, copytofilename)
                document:writeStubIdx(copytofilename..".idx")
        end


        return true
end
```

For the connector to continue processing the document, the `handler` function must return **true**. If the function returns **false**, the document is discarded.

> **TIP:**
> You can write a library of useful functions to share between multiple scripts. To include a library of functions in a script, add the code **dofile("library.lua")** to the top of the lua script, outside of the `handler` function.

Run a Lua Script using an Ingest Action

To run a Lua script on documents that are sent for ingestion, use an Ingest Action.

To run a Lua script using an Ingest Action

1. Open the connector's configuration file.

2. Find one of the following sections in the configuration file:

   - To run a Lua script on all documents retrieved by a specific task, find the [*TaskName*] section.

   - To run a Lua script on all documents that are sent for ingestion, find the [Ingestion] section.

     > **NOTE:**
     > If you set the IngestActions parameter in a [TaskName] section, the connector does not run any IngestActions set in the [Ingestion] section for that task.

3. Use the IngestActions parameter to specify the path to your Lua script. For example:

   IngestActions=LUA:C:\Autonomy\myScript.lua

4. Save and close the configuration file.

Related Topics

-

Example Lua Scripts

This section contains example Lua scripts.

-
-

Add a Field to a Document

The following script demonstrates how to add a field named "MyField" to a document, with a value of "MyValue".

```
function handler(config, document, params)
      document:addField("MyField", "MyValue");
      return true;
end
```

The following script demonstrates how to add the field AUTN_NEEDS_MEDIA_SERVER_ANALYSIS to all JPEG, TIFF and BMP documents. This field indicates to CFS that the file should be sent to a Media Server for analysis (you must also define the MediaServerAnalysis task in the CFS configuration file).

The script finds the file type using the DREREFERENCE document field, so this field must contain the file extension for the script to work correctly.

```
function handler(config, document, params)
      local extensions_for_ocr = { jpg = 1 , tif = 1, bmp = 1 };
      local filename = document:getFieldValue("DREREFERENCE");
      local extension, extension_found = filename:gsub("^.*%.(%w+)$", "%1", 1);

      if extension_found > 0 then
            if extensions_for_ocr[extension:lower()] ~= nil then
                document:addField("AUTN_NEEDS_MEDIA_SERVER_ANALYSIS", "");
            end
      end

      return true;
end
```

Merge Document Fields

This script demonstrates how to merge the values of document fields.

When you extract data from a repository, the connector can produce documents that have multiple values for a single field, for example:

```
#DREFIELD ATTACHMENT="attachment.txt"
#DREFIELD ATTACHMENT="image.jpg"
#DREFIELD ATTACHMENT="document.pdf"
```

This script shows how to merge the values of these fields, so that the values are contained in a single field, for example:

```
#DREFIELD ATTACHMENTS="attachment.txt, image.jpg, document.pdf"
```

Example Script

```
function handler(config, document, params)
      onefield(document,"ATTACHMENT","ATTACHMENTS")
      return true;
end

function onefield(document,existingfield,newfield)
      if document:hasField(existingfield) then
            local values = { document:getFieldValues(existingfield) }

            local newfieldvalue=""
            for i,v in ipairs(values) do
                if i>1 then
                    newfieldvalue = newfieldvalue ..", "
                end

                newfieldvalue = newfieldvalue..v
            end

            document:addField(newfield,newfieldvalue)
      end

      return true;
end
```

Chapter 9: Ingestion

After a connector finds new documents in a repository, or documents that have been updated or deleted, it sends this information to another component called the *ingestion target*. This section describes where you can send the information retrieved by the SharePoint OData Connector, and how to configure the ingestion target.

Introduction

A connector can send information to a single ingestion target, which could be:

- **Connector Framework Server**. To process information and then index it into IDOL, Haven OnDemand, or Vertica, send the information to a Connector Framework Server (CFS). Any files retrieved by the connector are *imported* using KeyView, which means the information contained in the files is converted into a form that can be indexed. If the files are containers that contain *subfiles*, these are extracted. You can manipulate and enrich documents using Lua scripts and automated tasks such as field standardization, image analysis, and speech-to-text processing. CFS can index your documents into one or more indexes. For more information about CFS, refer to the *Connector Framework Server Administration Guide*.
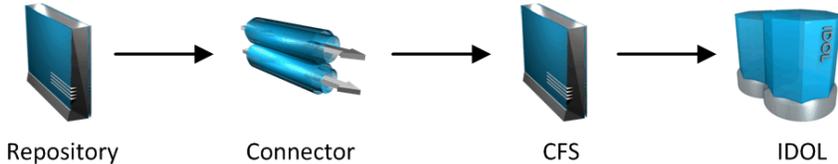
- **Haven OnDemand**. You can index documents directly into a Haven OnDemand text index. Haven OnDemand can extract text, metadata, and subfiles from over 1000 different file formats, so you might not need to send documents to CFS.

- **Another Connector.** Use another connector to keep another repository up-to-date. When a connector receives documents, it inserts, updates, or deletes the information in the repository. For example, you could use an Exchange Connector to extract information from Microsoft Exchange, and send the documents to a Notes Connector so that the information is inserted, updated, or deleted in the Notes repository.

  > **NOTE:**
  > The destination connector can only insert, update, and delete documents if it supports the `insert`,

> `update`, and `delete` fetch actions.

In most cases Micro Focus recommends ingesting documents through CFS, so that KeyView can extract content from any files retrieved by the connector and add this information to your documents. You can also use CFS to manipulate and enrich documents before they are indexed. However, if required you can configure the connector to index documents directly into:

- **IDOL Server**. You might index documents directly into IDOL Server when your connector produces metadata-only documents (documents that do not have associated files). In this case there is no need for the documents to be imported. Connectors that can produce metadata-only documents include ODBC Connector and Oracle Connector.
- **Vertica.** The metadata extracted by connectors is structured information held in structured fields, so you might use Vertica to analyze this information.
- **MetaStore**. You can index document metadata into a MetaStore for records management.

Send Data to Connector Framework Server

This section describes how to configure ingestion into Connector Framework Server (CFS).

To send data to a CFS

1. Stop the connector.
2. Open the connector's configuration file in a text editor.
3. In the `[Ingestion]` section, set the following parameters:

`EnableIngestion`	To enable ingestion, set this parameter to **true**.
`IngesterType`	To send data to CFS, set this parameter to **CFS**.
`IngestHost`	The host name or IP address of the CFS.
`IngestPort`	The ACI port of the CFS.

    For example:

    ```
    [Ingestion]
    EnableIngestion=True
    IngesterType=CFS
    IngestHost=localhost
    IngestPort=7000
    ```

4. (Optional) If you are sending documents to CFS for indexing into IDOL Server, set the `IndexDatabase` parameter. When documents are indexed, IDOL adds each document to the database specified in the document's `DREDBNAME` field. The connector sets this field for each document, using the value of `IndexDatabase`.

`IndexDatabase`	The name of the IDOL database into which documents are indexed. Ensure that this database exists in the IDOL Server configuration file.

    - To index all documents retrieved by the connector into the same IDOL database, set this parameter in the `[Ingestion]` section.

- To use a different database for documents retrieved by each task, set this parameter in the
  TaskName section.

5. Save and close the configuration file.

Send Data to Haven OnDemand

This section describes how to configure ingestion into Haven OnDemand. SharePoint OData Connector can index documents into a Haven OnDemand text index, or send the documents to a Haven OnDemand combination which can perform additional processing and then index the documents into a text index.

> **NOTE:**
> Haven OnDemand combinations do not accept binary files, so any documents that have associated binary files are indexed directly into a text index and cannot be sent to a combination.

Prepare Haven OnDemand

Before you can send documents to Haven OnDemand, you must create a text index. For information about how to create text indexes, refer to the Haven OnDemand documentation.

Before you can send documents to a Haven OnDemand combination endpoint, the combination must exist. SharePoint OData Connector requires your combination to accept the following input parameters, and produce the following output.

Input Parameters

Name	Type	Description
json	any	A JSON object that contains a single attribute 'documents' that is an array of document objects.
index	string	The name of the text index that you want the combination to add documents to. SharePoint OData Connector uses the value of the parameter HavenOnDemandIndexName to set this value.
duplicate_mode	string	Specifies how to handle duplicates when adding documents to the text index. SharePoint OData Connector uses the value of the parameter HavenOnDemandDuplicateMode to set this value.

Output

Name	Type	Description
result	any	The result of the call to AddToTextIndex made by the combination.

Send Data to Haven OnDemand

This section describes how to send documents to Haven OnDemand.

To send data to Haven OnDemand

1. Stop the connector.
2. Open the connector's configuration file in a text editor.
3. In the `[Ingestion]` section, set the following parameters:

`EnableIngestion`	To enable ingestion, set this parameter to **true**.
`IngesterType`	To send data to Haven OnDemand, set this parameter to **HavenOnDemand**.
`HavenOnDemandApiKey`	Your Haven OnDemand API key. You can obtain the key from your Haven OnDemand account.
`HavenOnDemandIndexName`	The name of the Haven OnDemand text index to index documents into.
`IngestSSLConfig`	The name of a section in the connector's configuration file that contains SSL settings. The connection to Haven OnDemand must be made over TLS. For more information about sending documents to the ingestion server over TLS, see Configure Outgoing SSL Connections, on page 32.
`HavenOnDemandCombinationName`	(Optional) The name of the Haven OnDemand combination to send documents to. If you set this parameter, SharePoint OData Connector sends documents to the combination endpoint instead of indexing them directly into the text index.

> **NOTE:**
> Haven OnDemand combinations do not accept binary files. Therefore any document that has an associated binary file is indexed directly into the text index.

If you don't set this parameter, SharePoint OData Connector indexes all documents directly into the text index specified by `HavenOnDemandIndexName`.

For example:

```
[Ingestion]
EnableIngestion=True
IngesterType=HavenOnDemand
HavenOnDemandApiKey=[Your API Key]
HavenOnDemandIndexName=MyTextIndex
IngestSSLConfig=SSLOptions
HavenOnDemandCombinationName=MyCombination
```

```
[SSLOptions]
SSLMethod=TLSV1
```

4. Save and close the configuration file.

Send Data to Another Repository

You can configure a connector to send the information it retrieves to another connector. When the destination connector receives the documents, it inserts them into another repository. When documents are updated or deleted in the source repository, the source connector sends this information to the destination connector so that the documents can be updated or deleted in the other repository.

> **NOTE:**
> The destination connector can only insert, update, and delete documents if it supports the `insert`, `update`, and `delete` fetch actions.

To send data to another connector for ingestion into another repository

1. Stop the connector.
2. Open the connector's configuration file in a text editor.
3. In the `[Ingestion]` section, set the following parameters:

    EnableIngestion   To enable ingestion, set this parameter to **true**.

    IngesterType   To send data to another repository, set this parameter to **Connector**.

    IngestHost   The host name or IP address of the machine hosting the destination connector.

    IngestPort   The ACI port of the destination connector.

    IngestActions   Set this parameter so that the source connector runs a Lua script to convert documents into form that can be used with the destination connector's insert action. For information about the required format, refer to the Administration Guide for the destination connector.

    For example:

    ```
    [Ingestion]
    EnableIngestion=True
    IngesterType=Connector
    IngestHost=AnotherConnector
    IngestPort=7010
    IngestActions=Lua:transformation.lua
    ```

4. Save and close the configuration file.

Index Documents Directly into IDOL Server

This section describes how to index documents from a connector directly into IDOL Server.

> **TIP:**
> In most cases, Micro Focus recommends sending documents to a Connector Framework Server (CFS). CFS extracts metadata and content from any files that the connector has retrieved, and can manipulate and enrich documents before they are indexed. CFS also has the capability to insert documents into more than one index, for example IDOL Server and a Vertica database. For information about sending documents to CFS, see Send Data to Connector Framework Server, on page 89

To index documents directly into IDOL Server

1. Stop the connector.
2. Open the connector's configuration file in a text editor.
3. In the `[Ingestion]` section, set the following parameters:

`EnableIngestion`	To enable ingestion, set this parameter to `true`.
`IngesterType`	To send data to IDOL Server, set this parameter to `Indexer`.
`IndexDatabase`	The name of the IDOL database to index documents into.

   For example:

   ```
   [Ingestion]
   EnableIngestion=True
   IngesterType=Indexer
   IndexDatabase=News
   ```

4. In the `[Indexing]` section of the configuration file, set the following parameters:

`IndexerType`	To send data to IDOL Server, set this parameter to `IDOL`.
`Host`	The host name or IP address of the IDOL Server.
`Port`	The IDOL Server ACI port.
`SSLConfig`	(Optional) The name of a section in the connector's configuration file that contains SSL settings for connecting to IDOL.

   For example:

   ```
   [Indexing]
   IndexerType=IDOL
   Host=10.1.20.3
   Port=9000
   SSLConfig=SSLOptions
   ```

```
[SSLOptions]
SSLMethod=SSLV23
```

5. Save and close the configuration file.

Index Documents into Vertica

SharePoint OData Connector can index documents into Vertica, so that you can run queries on structured fields (document metadata).

Depending on the metadata contained in your documents, you could investigate the average age of documents in a repository. You might want to answer questions such as: How much time has passed since the documents were last updated? How many files are regularly updated? Does this represent a small proportion of the total number of documents? Who are the most active users?

> **TIP:**
> In most cases, Micro Focus recommends sending documents to a Connector Framework Server (CFS). CFS extracts metadata and content from any files that the connector has retrieved, and can manipulate and enrich documents before they are indexed. CFS also has the capability to insert documents into more than one index, for example IDOL Server and a Vertica database. For information about sending documents to CFS, see Send Data to Connector Framework Server, on page 89

Prerequisites

- SharePoint OData Connector supports indexing into Vertica 7.1 and later.
- You must install the appropriate Vertica ODBC drivers (version 7.1 or later) on the machine that hosts SharePoint OData Connector. If you want to use an ODBC Data Source Name (DSN) in your connection string, you will also need to create the DSN. For more information about installing Vertica ODBC drivers and creating the DSN, refer to the Vertica documentation.

New, Updated and Deleted Documents

When documents are indexed into Vertica, SharePoint OData Connector adds a timestamp that contains the time when the document was indexed. The field is named `VERTICA_INDEXER_TIMESTAMP` and the timestamp is in the format `YYYY-MM-DD HH:NN:SS`.

When a document in a data repository is modified, SharePoint OData Connector adds a new record to the database with a new timestamp. All of the fields are populated with the latest data. The record describing the older version of the document is not deleted. You can create a projection to make sure your queries only return the latest record for a document.

When SharePoint OData Connector detects that a document has been deleted from a repository, the connector inserts a new record into the database. The record contains only the `DREREFERENCE` and the field `VERTICA_INDEXER_DELETED` set to `TRUE`.

Fields, Sub-Fields, and Field Attributes

Documents that are created by connectors can have multiple levels of fields, and field attributes. A database table has a flat structure, so this information is indexed into Vertica as follows:

- Document fields become columns in the flex table. An IDOL document field and the corresponding database column have the same name.

- Sub-fields become columns in the flex table. A document field named `my_field` with a sub-field named `subfield` results in two columns, `my_field` and `my_field.subfield`.

- Field attributes become columns in the flex table. A document field named `my_field`, with an attribute named `my_attribute` results in two columns, `my_field` holding the field value and `my_field.my_attribute` holding the attribute value.

Prepare the Vertica Database

Indexing documents into a standard database is problematic, because documents do not have a fixed schema. A document that represents an image has different metadata fields to a document that represents an e-mail message. Vertica databases solve this problem with *flex tables*. You can create a flex table without any column definitions, and you can insert a record regardless of whether a referenced column exists.

You must create a flex table before you index data into Vertica.

When creating the table, consider the following:

- Flex tables store entire records in a single column named `__raw__`. The default maximum size of the `__raw__` column is 128K. You might need to increase the maximum size if you are indexing documents with large amounts of metadata.

- Documents are identified by their `DREREFERENCE`. Micro Focus recommends that you do not restrict the size of any column that holds this value, because this could result in values being truncated. As a result, rows that represent different documents might appear to represent the same document. If you do restrict the size of the `DREREFERENCE` column, ensure that the length is sufficient to hold the longest `DREREFERENCE` that might be indexed.

To create a flex table without any column definitions, run the following query:

```
create flex table my_table();
```

To improve query performance, create real columns for the fields that you query frequently. For documents indexed by a connector, this is likely to include the `DREREFERENCE`:

```
create flex table my_table(DREREFERENCE varchar NOT NULL);
```

You can add new column definitions to a flex table at any time. Vertica automatically populates new columns with values for existing records. The values for existing records are extracted from the `__raw__` column.

For more information about creating and using flex tables, refer to the Vertica Documentation or contact Vertica technical support.

Send Data to Vertica

To send documents to a Vertica database, follow these steps.

To send data to Vertica

1. Stop the connector.
2. Open the connector's configuration file in a text editor.
3. In the `[Ingestion]` section, set the following parameters:

    EnableIngestion   To enable ingestion, set this parameter to `true`.

    IngesterType   To send data to a Vertica database, set this parameter to `Indexer`.

    For example:

    ```
    [Ingestion]
    EnableIngestion=TRUE
    IngesterType=Indexer
    ```

4. In the `[Indexing]` section, set the following parameters:

    IndexerType   To send data to a Vertica database, set this parameter to `Library`.

    LibraryDirectory The directory that contains the library to use to index data.

    LibraryName   The name of the library to use to index data. You can omit the `.dll` or `.so` file extension. Set this parameter to `verticaIndexer`.

    ConnectionString The connection string to use to connect to the Vertica database.

    TableName   The name of the table in the Vertica database to index the documents into. The table must be a flex table and must exist before you start indexing documents. For more information, see Prepare the Vertica Database, on the previous page.

    For example:

    ```
    [Indexing]
    IndexerType=Library
    LibraryDirectory=indexerdlls
    LibraryName=verticaIndexer
    ConnectionString=DSN=VERTICA
    TableName=my_flex_table
    ```

5. Save and close the configuration file.

Send Data to a MetaStore

You can configure a connector to send documents to a MetaStore. When you send data to a Metastore, any files associated with documents are ignored.

> **TIP:**
> In most cases, Micro Focus recommends sending documents to a Connector Framework Server (CFS). CFS extracts metadata and content from any files that the connector has retrieved, and can manipulate and enrich documents before they are indexed. CFS also has the capability to insert documents into more than one index, for example IDOL Server and a MetaStore. For information about sending documents to CFS, see

To send data to a MetaStore

1. Stop the connector.
2. Open the connector's configuration file in a text editor.
3. In the `[Ingestion]` section, set the following parameters:

`EnableIngestion`	To enable ingestion, set this parameter to `true`.
`IngesterType`	To send data to a MetaStore, set this parameter to `Indexer`.

   For example:

   ```
   [Ingestion]
   EnableIngestion=True
   IngesterType=Indexer
   ```

4. In the `[Indexing]` section, set the following parameters:

`IndexerType`	To send data to a MetaStore, set this parameter to `MetaStore`.
`Host`	The host name of the machine hosting the MetaStore.
`Port`	The port of the MetaStore.

   For example:

   ```
   [Indexing]
   IndexerType=Metastore
   Host=MyMetaStore
   Port=8000
   ```

5. Save and close the configuration file.

Run a Lua Script after Ingestion

You can configure the connector to run a Lua script after batches of documents are successfully sent to the ingestion server. This can be useful if you need to log information about documents that were processed, for monitoring and reporting purposes.

To configure the file name of the Lua script to run, set the `IngestBatchActions` configuration parameter in the connector's configuration file.

- To run the script for all batches of documents that are ingested, set the parameter in the `[Ingestion]` section.
- To run the script for batches of documents retrieved by a specific task, set the parameter in the

[*TaskName*] section.

> **NOTE:**
> If you set the parameter in a [*TaskName*] section, the connector does not run any scripts
> specified in the [Ingestion] section for that task.

For example:

```
[Ingestion]
IngestBatchActions0=LUA:./scripts/myScript.lua
```

For more information about this parameter, refer to the *SharePoint OData Connector Reference*.

The Lua script must have the following structure:

```
function batchhandler(documents, ingesttype)
    ...
end
```

The `batchhandler` function is called after each batch of documents is sent to the ingestion server. The function is passed the following arguments:

Argument	Description
documents	A table of document objects, where each object represents a document that was sent to the ingestion server.
	A document object is an internal representation of a document. You can modify the document object and this changes the document. However, as the script runs after the documents are sent to the ingestion server, any changes you make are not sent to CFS or IDOL.
ingesttype	A string that contains the ingest type for the documents. The `batchhandler` function is called multiple times if different document types are sent.

For example, the following script prints the ingest type (ADD, DELETE, or UPDATE) and the reference for all successfully processed documents to `stdout`:

```
function batchhandler(documents, ingesttype)
    for i,document in ipairs(documents) do
        local ref = document:getReference()
        print(ingesttype..": "..ref)
    end
 end
```

Chapter 10: Monitor the Connector

This section describes how to monitor the connector.

IDOL Admin

IDOL Admin is an administration interface for performing ACI server administration tasks, such as gathering status information, monitoring performance, and controlling the service. IDOL Admin provides an alternative to constructing actions and sending them from your web browser.

Prerequisites

By default, the latest version of SharePoint OData Connector should include the `admin.dat` file that is required to run IDOL Admin. If you do not have this file, you must download it separately.

Supported Browsers

IDOL Admin supports the following browsers:

- Internet Explorer 11 and later
- Edge
- Chrome (latest version)
- Firefox (latest version)

Install IDOL Admin

You must install IDOL Admin on the same host that the ACI server or component is installed on. To set up a component to use IDOL Admin, you must configure the location of the `admin.dat` file and enable Cross Origin Resource Sharing.

To install IDOL Admin

1. Stop the ACI server.

2. Save the `admin.dat` file to any directory on the host.

3. Using a text editor, open the ACI server or component configuration file. For the location of the configuration file, see the ACI server documentation.

4. In the `[Paths]` section of the configuration file, set the `AdminFile` parameter to the location of the `admin.dat` file. If you do not set this parameter, the ACI server attempts to find the `admin.dat` file in its working directory when you call the IDOL Admin interface.

5. Enable Cross Origin Resource Sharing.

6. In the `[Service]` section, add the `Access-Control-Allow-Origin` parameter and set its value to the URLs that you want to use to access the interface.

    Each URL must include:

    - the `http://` or `https://` prefix

      > **NOTE:**
      > URLs can contain the `https://` prefix if the ACI server or component has SSL enabled.

    - The host that IDOL Admin is installed on

    - The ACI port of the component that you are using IDOL Admin for

    Separate multiple URLs with spaces.

    For example, you could specify different URLs for the local host and remote hosts:

    ```
    Access-Control-Allow-Origin=http://localhost:9010
    http://Computer1.Company.com:9010
    ```

    Alternatively, you can set `Access-Control-Allow-Origin=*`, which allows you to access IDOL Admin using any valid URL (for example, `localhost`, direct IP address, or the host name). The wildcard character (*) is supported only if no other entries are specified.

    If you do not set the `Access-Control-Allow-Origin` parameter, IDOL Admin can communicate only with the server's ACI port, and not the index or service ports.

7. Start the ACI server.

You can now access IDOL Admin (see Access IDOL Admin, below).

Access IDOL Admin

You access IDOL Admin from a web browser. You can access the interface only through URLs that are set in the `Access-Control-Allow-Origin` parameter in the ACI server or component configuration file. For more information about configuring URL access, see Install IDOL Admin, on the previous page.

To access IDOL Admin from the host that it is installed on

- Type the following URL into the address bar of your web browser:

  ```
  http://localhost:port/action=admin
  ```

  where *port* is the ACI server or component ACI port.

To access IDOL Admin from a different host

- Type the following URL into the address bar of your web browser:

  `http://`*`host`*`:`*`port`*`/action=admin`

  where:

  host   is the name or IP address of the host that IDOL Admin is installed on.

  port   is the ACI server or component ACI port of the IDOL Admin host.

View Connector Statistics

SharePoint OData Connector collects statistics about the work it has completed. The statistics that are available depend on the connector you are using, but all connectors provide information about the number and frequency of ingest-adds, ingest-updates, and ingest-deletes.

To view connector statistics

- Use the `GetStatistics` service action, for example:

  `http://`*`host`*`:`*`serviceport`*`/action=GetStatistics`

  where *host* is the host name or IP address of the machine where the connector is installed, and *serviceport* is the connector's service port.

  For information about the statistics that are returned, refer to the documentation for the `GetStatistics` service action.

The connector includes an XSL template (`ConnectorStatistics.tmpl`) that you can use to visualize the statistics. You can use the template by adding the `template` parameter to the request:

`http://`*`host`*`:`*`serviceport`*`/action=GetStatistics&template=ConnectorStatistics`

When you are using the `ConnectorStatistics` template, you can also add the `filter` parameter to the request to return specific statistics. The `filter` parameter accepts a regular expression that matches against the string *autnid*`::`*name*, where *autnid* and *name* are the values of the corresponding attributes in the XML returned by the `GetStatistics` action. For example, the following request returns statistics only for synchronize actions:

```
http://host:serviceport/action=GetStatistics&template=ConnectorStatistics
                                          &filter=^synchronize:
```

The following request returns statistics only for the task `mytask`:

```
http://host:serviceport/action=GetStatistics&template=ConnectorStatistics
                                          &filter=:mytask:
```

The following image shows some example statistics returned by a connector:
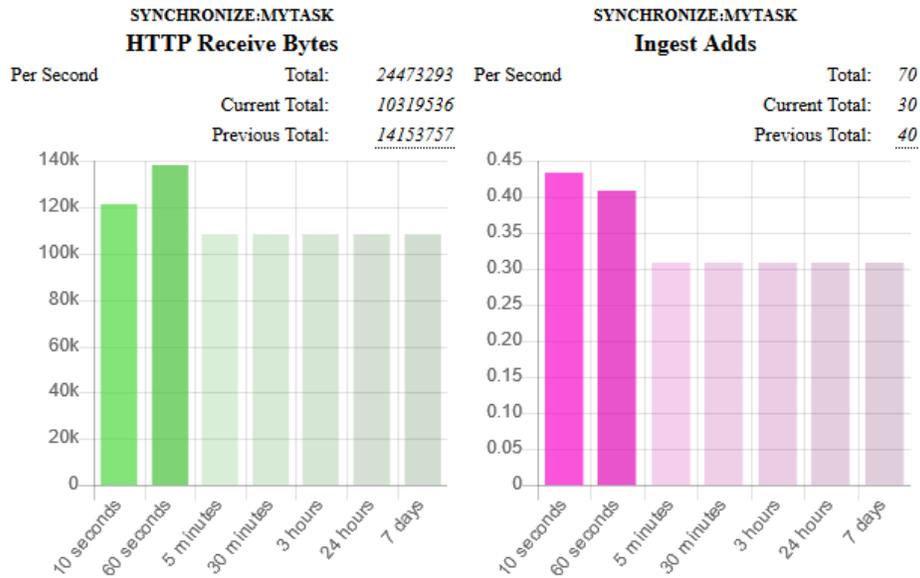
Above each chart is a title, for example SYNCHRONIZE:MYTASK, that specifies the action and task to which the statistics belong.

You can see from the example that in the last 60 seconds, the connector has generated an average of approximately 0.4 ingest-adds per second. In the charts, partially transparent bars indicate that the connector has not completed collecting information for those time intervals. The information used to generate statistics is stored in memory, so is lost if you stop the connector.

The following information is presented above the chart for each statistic:

- **Total** is a running total since the connector started. In the example above, there have been 70 ingest-adds in total.

- **Current Total** is the total for the actions that are currently running. In the example above, the synchronize action that is running has resulted in 30 ingest-adds being sent to CFS.

- **Previous Total** provides the totals for previous actions. In the example above, the previous synchronize cycle resulted in 40 ingest-adds. To see the totals for the 24 most recent actions, hover the mouse pointer over the value.

Use the Connector Logs

As the SharePoint OData Connector runs, it outputs messages to its logs. Most log messages occur due to normal operation, for example when the connector starts, receives actions, or sends documents for ingestion. If the connector encounters an error, the logs are the first place to look for information to help troubleshoot the problem.

The connector separates messages into the following message types, each of which relates to specific features:

Log Message Type	Description

Action	Logs actions that are received by the connector, and related messages.
Application	Logs application-related occurrences, such as when the connector starts.
Collect	Messages related to the `Collect` fetch action.
Delete	Messages related to the `Delete` fetch action.
Identifiers	Messages related to the `Identifiers` fetch action.
Insert	Messages related to the `Insert` fetch action.
Synchronize	Messages related to the `Synchronize` fetch action.
SynchronizeGroups	Messages related to the `SynchronizeGroups` fetch action.
Update	Messages related to the `Update` fetch action.
View	Messages related to the `View` action.

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 SharePoint OData Connector 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 *SharePoint OData Connector 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.

Monitor the Progress of a Task

This section describes how to monitor the progress of a task.

> **NOTE:**
> Progress reporting is not available for every action.

To monitor the progress of a task

- Send the following action to the connector:

  `action=QueueInfo&QueueName=fetch&QueueAction=progress&Token=...`

  where,

  Token   The token of the task that you want to monitor. If you started the task by sending an action to the connector, the token was returned in the response. If the connector started the task according to the schedule in its configuration file, you can use the `QueueInfo` action to find the token (use `/action=QueueInfo&QueueName=fetch&QueueAction=getstatus`).

The connector returns the progress report, inside the `<progress>` element of the response. The following example is for a File System Connector synchronize task.

```
<autnresponse>
  <action>QUEUEINFO</action>
  <response>SUCCESS</response>
  <responsedata>
    <action>
      <token>MTAuMi4xMDUuMTAzOjEyMzQ2Q6RkVUQ0g6MTAxNzM0MzgzOQ==</token>
      <status>Processing</status>
      <progress>
        <building_mode>false</building_mode>
        <percent>7.5595</percent>
```

```
            <time_processing>18</time_processing>
            <estimated_time_remaining>194</estimated_time_remaining>
            <stage title="MYTASK" status="Processing" weight="1" percent="7.5595">
                <stage title="Ingestion" status="Processing" weight="999"
  percent="7.567">
                    <stage title="C:\Test Files\" status="Processing" weight="6601"
  percent="7.567" progress="0" maximum="6601">
                        <stage title="Folder01" status="Processing" weight="2317"
  percent="43.116" progress="999" maximum="2317"/>
                        <stage title="Folder02" status="Pending" weight="2567"/>
                        <stage title="Folder03" status="Pending" weight="1715"/>
                        <stage title="." status="Pending" weight="2"/>
                    </stage>
                </stage>
                <stage title="Deletion" status="Pending" weight="1"/>
            </stage>
        </progress>
    </action>
  </responsedata>
</autnresponse>
```

To read the progress report

The information provided in the progress report is unique to each connector and each action. For example, the File System Connector reports the progress of a synchronize task by listing the folders that require processing.

A progress report can include several *stages*:

- A *stage* represents part of a task.

- A stage can have sub-stages. In the previous example, the stage "C:\Test Files\" has three stages that represent sub-folders ("Folder01", "Folder02", and "Folder03") and one stage that represents the contents of the folder itself ("."). You can limit the depth of the sub-stages in the progress report by setting the MaxDepth parameter in the QueueInfo action.

- The weight attribute indicates the amount of work included in a stage, relative to other stages at the same level.

- The status attribute shows the status of a stage. The status can be "Pending", "Processing", or "Finished".

- The progress attribute shows the number of items that have been processed for the stage.

- The maximum attribute shows the total number of items that must be processed to complete the stage.

- The percent attribute shows the progress of a stage (percentage complete). In the previous example, the progress report shows that MYTASK is 7.5595% complete.

- Finished stages are grouped, and pending stages are not expanded into sub-stages, unless you set the action parameter AllStages=true in the QueueInfo action.

Monitor Asynchronous Actions using Event Handlers

The fetch actions sent to a connector are asynchronous. Asynchronous actions do not run immediately, but are added to a queue. This means that the person or application that sends the action does not receive an immediate response. However, you can configure the connector to call an event handler when an asynchronous action starts, finishes, or encounters an error.

You can use an event handler to:

- return data about an event back to the application that sent the action.
- write event data to a text file, to log any errors that occur.

You can also use event handlers to monitor the size of asynchronous action queues. If a queue becomes full this might indicate a problem, or that applications are making requests to SharePoint OData Connector faster than they can be processed.

SharePoint OData Connector can call an event handler for the following events.

OnStart   The `OnStart` event handler is called when SharePoint OData Connector starts processing an asynchronous action.

OnFinish   The `OnFinish` event handler is called when SharePoint OData Connector successfully finishes processing an asynchronous action.

OnError   The `OnError` event handler is called when an asynchronous action fails and cannot continue.

OnQueueEvent   The `OnQueueEvent` handler is called when an asynchronous action queue becomes full, becomes empty, or the queue size passes certain thresholds.

- A `QueueFull` event occurs when the action queue becomes full.

- A `QueueFilling` event occurs when the queue size exceeds a configurable threshold (`QueueFillingThreshold`) and the last event was a `QueueEmpty` or `QueueEmptying` event.

- A `QueueEmptying` event occurs when the queue size falls below a configurable threshold (`QueueEmptyingThreshold`) and the last event was a `QueueFull` or `QueueFilling` event.

- A `QueueEmpty` event occurs when the action queue becomes empty.

SharePoint OData Connector supports the following types of event handler:

- The `TextFileHandler` writes event data to a text file.
- The `HttpHandler` sends event data to a URL.
- The `LuaHandler` runs a Lua script. The event data is passed into the script.

Configure an Event Handler

To configure an event handler, follow these steps.

To configure an event handler

1. Stop the connector.

2. Open the connector's configuration file in a text editor.

3. Set the `OnStart`, `OnFinish`, `OnError`, or `OnQueueEvent` parameter to specify the name of a section in the configuration file that contains the event handler settings.

   - To run an event handler for all asynchronous actions, set these parameters in the `[Actions]` section. For example:

     ```
     [Actions]
     OnStart=NormalEvents
     OnFinish=NormalEvents
     OnError=ErrorEvents
     ```

   - To run an event handler for specific actions, use the action name as a section in the configuration file. The following example calls an event handler when the *Fetch* action starts and finishes successfully:

     ```
     [Fetch]
     OnStart=NormalEvents
     OnFinish=NormalEvents
     ```

4. Create a new section in the configuration file to contain the settings for your event handler. You must name the section using the name you specified with the `OnStart`, `OnFinish`, `OnError`, or `OnQueueEvent` parameter.

5. In the new section, set the `LibraryName` parameter.

   `LibraryName`   The type of event handler to use to handle the event:

   - To write event data to a text file, set this parameter to **TextFileHandler**, and then set the `FilePath` parameter to specify the path of the file.

   - To send event data to a URL, set this parameter to **HttpHandler**, and then use the HTTP event handler parameters to specify the URL, proxy server settings, credentials and so on.

   - To run a Lua script, set this parameter to **LuaHandler**, and then set the `LuaScript` parameter to specify the script to run. For information about writing the script, see Write a Lua Script to Handle Events, on the next page.

   For example:

   ```
   [NormalEvents]
   LibraryName=TextFileHandler
   FilePath=./events.txt

   [ErrorEvents]
   LibraryName=LuaHandler
   LuaScript=./error.lua
   ```

6. Save and close the configuration file. You must restart SharePoint OData Connector for your changes to take effect.

Write a Lua Script to Handle Events

The Lua event handler runs a Lua script to handle events. The Lua script must contain a function named `handler` with the arguments `request` and `xml`, as shown below:

```
function handler(request, xml)
        ...
end
```

- `request` is a table holding the request parameters. For example, if the request was `action=Example&MyParam=Value`, the table will contain a key `MyParam` with the value `Value`. Some events, for example queue size events, are not related to a specific action and so the table might be empty.
- `xml` is a string of XML that contains information about the event.

Set Up Performance Monitoring

You can configure a connector to pause tasks temporarily if performance indicators on the local machine or a remote machine breach certain limits. For example, if there is a high load on the CPU or memory of the repository from which you are retrieving information, you might want the connector to pause until the machine recovers.

> **NOTE:**
> Performance monitoring is available on Windows platforms only. To monitor a remote machine, both the connector machine and remote machine must be running Windows.

Configure the Connector to Pause

To configure the connector to pause

1. Open the configuration file in a text editor.
2. Find the `[FetchTasks]` section, or a `[TaskName]` section.
   - To pause all tasks, use the `[FetchTasks]` section.
   - To specify settings for a single task, find the `[TaskName]` section for the task.
3. Set the following configuration parameters:

`PerfMonCounterNameN`	The names of the performance counters that you want the connector to monitor. You can use any counter that is available in the Windows `perfmon` utility.
`PerfMonCounterMinN`	The minimum value permitted for the specified performance counter. If the counter falls below this value, the connector pauses until the counter meets the limits again.

PerfMonCounterMaxN	The maximum value permitted for the specified performance counter. If the counter exceeds this value, the connector pauses until the counter meets the limits again.
PerfMonAvgOverReadings	(Optional) The number of readings that the connector averages before checking a performance counter against the specified limits. For example, if you set this parameter to 5, the connector averages the last five readings and pauses only if the average breaches the limits. Increasing this value makes the connector less likely to pause if the limits are breached for a short time. Decreasing this value allows the connector to continue working faster following a pause.
PerfMonQueryFrequency	(Optional) The amount of time, in seconds, that the connector waits between taking readings from a performance counter.

For example:

```
[FetchTasks]
PerfMonCounterName0=\\machine-hostname\Memory\Available MBytes
PerfMonCounterMin0=1024
PerfMonCounterMax0=1024000
PerfMonCounterName1=\\machine-hostname\Processor(_Total)\% Processor Time
PerfMonCounterMin1=0
PerfMonCounterMax1=70
PerfMonAvgOverReadings=5
PerfMonQueryFrequency=10
```

> **NOTE:**
> You must set both a minimum and maximum value for each performance counter. You can not set only a minimum or only a maximum.

4. Save and close the configuration file.

Determine if an Action is Paused

To determine whether an action has been paused for performance reasons, use the `QueueInfo` action:

/action=queueInfo&queueAction=getStatus&queueName=fetch

You can also include the optional `token` parameter to return information about a single action:

/action=queueInfo&queueAction=getStatus&queueName=fetch&token=...

The connector returns the status, for example:

```
<autnresponse>
  <action>QUEUEINFO</action>
  <response>SUCCESS</response>
  <responsedata>
    <actions>
      <action owner="2266112570">
```

```
        <status>Processing</status>
        <queued_time>2016-Jul-27 14:49:40</queued_time>
        <time_in_queue>1</time_in_queue>
        <process_start_time>2016-Jul-27 14:49:41</process_start_time>
        <time_processing>219</time_processing>
        <documentcounts>
          <documentcount errors="0" task="MYTASK"/>
        </documentcounts>
        <fetchaction>SYNCHRONIZE</fetchaction>
        <pausedforperformance>true</pausedforperformance>
        <token>...</token>
      </action>
    </actions>
  </responsedata>
</autnresponse>
```

When the element `pausedforperformance` has a value of `true`, the connector has paused the task for performance reasons. If the `pausedforperformance` element is not present in the response, the connector has not paused the task.

Set Up Document Tracking

Document tracking reports metadata about documents when they pass through various stages in the indexing process. For example, when a connector finds a new document and sends it for ingestion, a document tracking event is created that shows the document has been added. Document tracking can help you detect problems with the indexing process.

You can write document tracking events to a database, log file, or IDOL Server. For information about how to set up a database to store document tracking events, refer to the *IDOL Server Administration Guide*.

To enable Document Tracking

1. Open the connector's configuration file.
2. Create a new section in the configuration file, named `[DocumentTracking]`.
3. In the new section, specify where the document tracking events are sent.
   - To send document tracking events to a database through ODBC, set the following parameters:

`Backend`	To send document tracking events to a database, set this parameter to **Library**.
`LibraryPath`	Specify the location of the ODBC document tracking library. This is included with IDOL Server.
`ConnectionString`	The ODBC connection string for the database.

     For example:

```
[DocumentTracking]
```
Backend=Library
LibraryPath=C:\Autonomy\IDOLServer\IDOL\modules\dt_odbc.dll
ConnectionString=DSN=MyDatabase

- To send document tracking events to the connector's synchronize log, set the following parameters:

  `Backend`   To send document tracking events to the connector's logs, set this parameter to **Log**.

  `DatabaseName`   The name of the log stream to send the document tracking events to. Set this parameter to **synchronize**.

For example:

```
[DocumentTracking]
```
Backend=Log
DatabaseName=synchronize

- To send document tracking events to an IDOL Server, set the following parameters:

  `Backend`   To send document tracking events to an IDOL Server, set this parameter to **IDOL**.

  `TargetHost`   The host name or IP address of the IDOL Server.

  `TargetPort`   The index port of the IDOL Server.

For example:

```
[DocumentTracking]
```
Backend=IDOL
TargetHost=idol
TargetPort=9001

For more information about the parameters you can use to configure document tracking, refer to the *SharePoint OData Connector Reference*.

4. Save and close the configuration file.

Appendix A: Document Fields

The connector adds the following fields to each document that it ingests:

Field Name	Description
AUTN_IDENTIFIER	An identifier that allows a connector to extract the document from the repository again, for example during the collect or view actions.
AUTN_MODIFICATIONS	Provides information about how many times an item has been modified. For example: `<AUTN_MODIFICATIONS modified="3" modified_history="FO9"/>` This field is not supported by all connectors, so it might not be present, and might not include all of the attributes described below: • `modified` - the number of modifications that have been observed by the connector. This might be a minimum number; if an item is modified more than once between synchronize cycles the connector might only observe a single change (this depends on the information available from the repository). • `modified_history` - the time intervals between recent modifications (up to 50, one character per interval, with the most recent change at the end of the list). To convert a character into a time duration: 1. Convert the character to an integer, n, (0 to 61) by the position in this string: `0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz`. 2. Calculate using floating point arithmetic $(13/9)^n$. 3. Read the resulting number as a number of seconds giving the minimum duration. Due to rounding the actual value will be in the range $(13/9)^n$ to $(13/9)^{(n+1)}$. For example, to read "F": ○ $n=15$, because "F" is the 16th character in the string above. ○ $(13/9)^{15} = (1.44444...)^{15} = 248.6...$ This gives a minimum duration of 0:04:08. The full range would be 248.6 to 359.1 seconds, or a duration between 0:04:08 and 0:05:59. • `attributesmodified` - the number of times the attributes of a file have been modified. This might be a minimum number; if an item is modified more than once between synchronize cycles the connector might only observe a single change (this depends on the information available from the repository). • `attributesmodified_history` - the time intervals between attributes being modified. This field has the same format as `modified_history`.
DocTrackingId	An identifier used for document tracking functionality.
DREREFERENCE	A reference for the document. This is the standard IDOL reference field, which is used for deduplication.

Field Name	Description
source_connector_ run_id	(Added only when `IngestSourceConnectorFields=TRUE`). The asynchronous action token of the `fetch` action that ingested the document.
source_connector_ server_id	(Added only when `IngestSourceConnectorFields=TRUE`). A token that identifies the instance of the connector that retrieved the document (different installations of the same connector populate this field with different IDs). You can retrieve the UID of a connector through `action=GetVersion`.

Glossary

A

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.

C

Category component
The IDOL Server component that manages categorization and clustering.

Community component
The IDOL Server component that manages users and communities.

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.

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.

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.

I

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.

Import

Importing is the process where CFS, using KeyView, extracts metadata, content, and sub-files from items retrieved by a connector. CFS adds the information to documents so that it is indexed into IDOL Server. Importing allows IDOL server to use the information in a repository, without needing to process the information in its native format.

Ingest

Ingestion converts information that exists in a repository into documents that can be indexed into IDOL Server. Ingestion starts when a connector finds new documents in a repository, or documents that have been updated or deleted, and sends this information to CFS. Ingestion includes the import process, and processing tasks that can modify and enrich the information in a document.

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.

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.

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

primary domain controller
    A server computer in a Microsoft Windows domain that controls various computer resources. See also: active directory, LDAP.

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 Administration Guide (Micro Focus SharePoint OData Connector 11.6)

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!