IDOL
Web Connector

Software Version 12.8

Administration Guide

MICRO FOCUS

Document Release Date: February 2021
Software Release Date: February 2021

Legal notices

Copyright notice

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

Documentation updates

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

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

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

Support

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

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

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

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

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 Web Connector.

Web Connector

The Web Connector is an IDOL connector that retrieves information from the World Wide Web. The connector can crawl the Web by following the links that exist on each page, or retrieve the resources listed in a site map.

The Web Connector uses an embedded browser to process Web pages and therefore provides several advantages over the HTTP Connector:

- The Web Connector supports Javascript and dynamic pages. If Javascript is used on a page, the script runs before the connector crawls the page for links and ingests the page. As a result the connector can crawl links generated by the script and the page that is ingested is the final source after any scripts have run.
- The Web Connector can interpret the structure of a page, as defined by the HTML markup, in addition to reading the content. As a result, when configuring the connector to extract links or submit forms, you can identify page elements using CSS selectors rather than regular expressions.

Features and Capabilities

- The Web Connector retrieves information from Web sites over HTTP.
- The embedded web browser in Web Connector 12.8 is Chromium 78.0.3886.1.
- The connector can crawl a Web site starting from a page that you specify, or retrieve the pages listed in a site map. When crawling a web site, the connector can follow links that exist in the HTML source of the page or in Adobe Flash content.
- The connector can extract information contained in data URIs. For example, images might be base-64 encoded and included in the source of an HTML page:

  ```
  <img src="...">
  ```

- The connector can log on to Web sites to retrieve content. The connector supports:
  - Basic authentication
  - HTTP Digest authentication.
  - NTLMv2 authentication.
- You can configure the connector to populate and submit HTML forms. This feature is useful when the connector must log on to a Web site.
- The connector uses canonical links in web pages and response headers to de-duplicate content retrieved from the web.

Supported Actions

The Web Connector supports the following actions:

Action	Supported
Synchronize	✓
Synchronize (identifiers)	✓
Synchronize Groups	✗
Collect	✓
Identifiers	✓
Insert	✗
Delete/Remove	✗
Hold/ReleaseHold	✗
Update	✗
Stub	✗
GetURI	✓
View	✓

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 Web Connector to index documents into IDOL Server, you must have a CFS. When you install the Web 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.

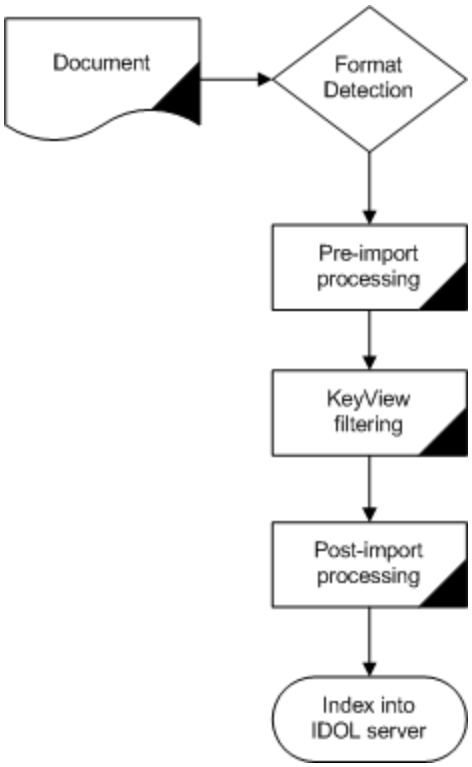
Index Documents

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).
- **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 Web 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.

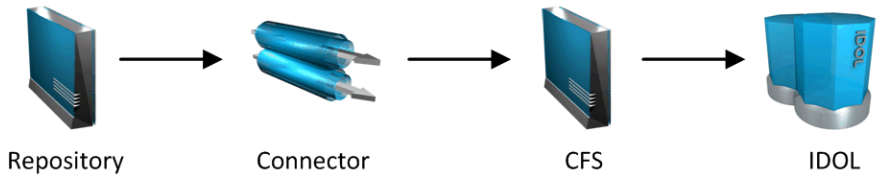
For more information on IDOL, see the *IDOL Getting Started Guide*.

System Architecture

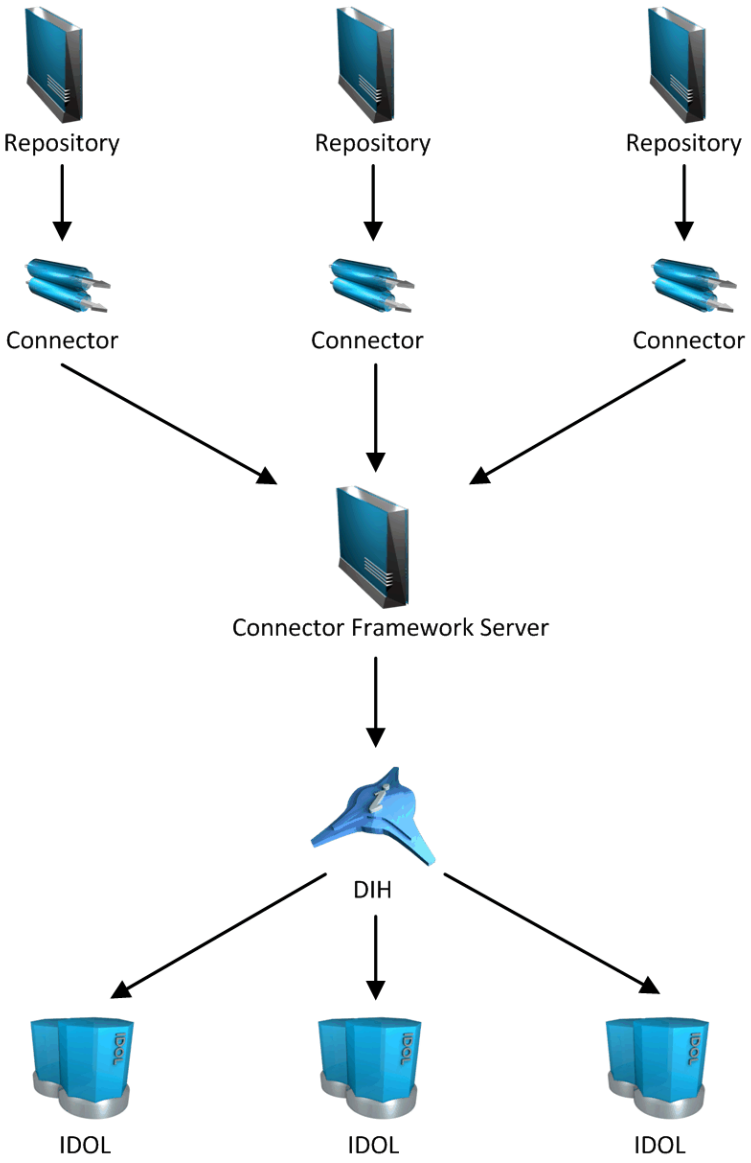
An IDOL infrastructure can include the following components:

- **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.



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

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

To display help for Web Connector

1. Start Web 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 Web Connector is installed.

   port is the ACI port by which you send actions to Web 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 Web Connector.

- *IDOL NiFi Ingest Help*

  The *IDOL NiFi Ingest Help* describes how to ingest data using IDOL NiFi Ingest, a set of IDOL components for data retrieval and enrichment, that run within an open-source framework called Apache NiFi. NiFi Ingest provides a new way to ingest data into IDOL, and can be used instead of a Connector Framework Server.

- *Connector Framework Server Administration Guide*

  Connector Framework Server (CFS) processes documents that are retrieved by connectors. CFS then indexes the documents into an IDOL index. 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 Web Connector

This section describes how to install the Web Connector.

System Requirements

Web 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.

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

The additional requirements for Web Connector are:

- At least one font must be installed on the connector machine.
- On Linux platforms the Web Connector requires NSS version 3.26 or later.
- On Linux platforms you must install the following dependencies, which are required by the embedded browser:

CentOS 7	Debian or Ubuntu
libatomic libX11 libXtst libXScrnSaver libXcomposite atk at-spi2-core at-spi2-atk cups cairo pango alsa-lib-devel	libatomic1 libx11-6 libxtst6 libxss1 libxcomposite1 libatk1.0-0 at-spi2-core libatk-adaptor cups libcairo2 libpango-1.0-0 libpci3

For example, on CentOS 7:

```
sudo yum install libatomic libX11 libXtst libXScrnSaver libXcomposite atk
at-spi2-core at-spi2-atk cups cairo pango alsa-lib-devel
```

Install Web Connector

The Web Connector can be installed using the IDOL Server installer.

For information about installing the Web Connector using this installer, refer to the *IDOL Getting Started Guide*.

Configure the License Server Host and Port

Web Connector is licensed through License Server. In the Web Connector configuration file, specify the information required to connect to the License Server.

To specify the license server host and port

1. Open your configuration file in a text editor.
2. In the `[License]` section, modify the following parameters to point to your License Server.

`LicenseServerHost`	The host name or IP address of your License Server.
`LicenseServerACIPort`	The ACI port of your License Server.

   For example:

   ```
   [License]
   LicenseServerHost=licenses
   LicenseServerACIPort=20000
   ```

3. Save and close the configuration file.

Chapter 3: Configure Web Connector

This section describes how to configure the Web Connector.

Web Connector Configuration File

You can configure the Web 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 Web Connector, refer to the *Web 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, IDOL NiFi Ingest, 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 102

Modify Configuration Parameter Values

You modify Web 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 Web 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 Web 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 (including the section headings) from an external configuration file at a specified point in your configuration file. You can include a whole configuration section in this way, but the configuration section name in the external file must exactly match what you want to use in your file. If you want to use a configuration section from the external file with a different name, see Merge a Section from an External Configuration File, on the next page.

To include sections of an external configuration file

1. Open your configuration file in a text editor.
2. Find the place in the configuration file where you want to add the external configuration file section.
3. On a new line, type a left angle bracket (<), followed by the path of the external configuration file, in quotation marks (""). You can use relative paths and network locations. After the configuration file path, add the configuration section name that you want to include. For example:

   ```
   < "K:\sharedconfig\extrasettings.cfg" [License]
   ```

   NOTE: You cannot include a section that already exists in your configuration file.
4. Save and close the configuration file.

Include Parameters from an External Configuration File

This method allows you to import one or more parameters from an external configuration file at a specified point in your configuration file. You can import a single parameter or use wildcards to specify multiple parameters. The parameter values in the external file must match what you want to use in your file. This method does not import the section heading, such as `[License]` in the following examples.

To include parameters from an external configuration file

1. Open your configuration file in a text editor.
2. Find the place in the configuration file where you want to add the parameters from the external configuration file.
3. On a new line, type a left angle bracket (<), followed by the path of the external configuration file, in quotation marks (""). You can use relative paths and network locations. After the configuration file path, add the name of the section that contains the parameter, followed by the parameter name. For example:

   ```
   < "license.cfg" [License] LicenseServerHost
   ```

   To specify a default value for the parameter, in case it does not exist in the external configuration file, specify the configuration section, parameter name, and then an equals sign (=) followed by the default value. For example:

   ```
   < "license.cfg" [License] LicenseServerHost=localhost
   ```

   You can use wildcards to import multiple parameters, but this method does not support default values. The * wildcard matches zero or more characters. The ? wildcard matches any single character. Use the pipe character | as a separator between wildcard strings. For example:

   ```
   < "license.cfg" [License] LicenseServer*
   ```
4. Save and close the configuration file.

Merge a Section from an External Configuration File

This method allows you to include a configuration section from an external configuration file as part of your Web 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, Web Connector uses the values in the `[SharedSSLOptions]` section of the external configuration file as the values in the `[SSLOptions1]` section of the Web 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, Web 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 Web 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. Web 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 Web Connector installation folder.
2. At the command line, type:

   `autpassword -e -t`*`EncryptionType`* `[-oKeyFile] [-c`*`FILE`* `-s`*`SECTION`* `-p`*`PARAMETER`*`]` *`PasswordString`*

   where:

Option	Description
`-t` *`EncryptionType`*	The type of encryption to use: • **`Basic`** • **`AES`** - AES256 For example: **`-tAES`** **NOTE:** AES is more secure than basic encryption.
`-oKeyFile`	AES encryption requires a key file. This option specifies the path and file name of a key file. The key file must contain 64 hexadecimal characters. For example: **`-oKeyFile=./key.ky`** **NOTE:** The full (absolute) path of the key file is included in the encrypted value, because Web Connector requires the key to decrypt the password. If you move or rename the key file, this path becomes invalid and you must update the encrypted value.
`-cFILE -sSECTION -pPARAMETER`	(Optional) You can use these options to write the password directly into a configuration file. You must specify all three options. • **`-c`**. The configuration file in which to write the encrypted password. • **`-s`**. The name of the section in the configuration file in which to write the password. • **`-p`**. The name of the parameter in which to write the encrypted password. For example: **`-c./Config.cfg -sMyTask -pPassword`**
`PasswordString`	The password to encrypt.

   For example:

   `autpassword -e -tBASIC` *`MyPassword`*

   `autpassword -e -tAES -oKeyFile=./key.ky` *`MyPassword`*

```
autpassword -e -tAES -oKeyFile=./key.ky -c./Config.cfg -sDefault -pPassword MyPassword
```

The password is returned, or written to the configuration file.

Decrypt a Password

The following procedure describes how to decrypt a password.

To decrypt a password

1. Open a command-line window and change directory to the Web Connector installation folder.
2. At the command line, type:

   ```
   autpassword -d -tEncryptionType PasswordString
   ```

   where:

Option	Description
`-tEncryptionType`	The type of encryption: • **`Basic`** • **`AES`** - AES256 For example: **`-tAES`**
`PasswordString`	The password to decrypt.

   For example:

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

   ```
   autpassword -d -tAES PasswordString
   ```

   The password is returned in plain text.

Configure Client Authorization

You can configure Web 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 *Web Connector Reference*.

> **IMPORTANT:** To ensure that Web Connector allows only the options that you configure in `[AuthorizationRoles]`, make sure that you delete any deprecated *`Role`*`Clients` parameters from

your configuration (where *Role* corresponds to a standard role name, for example `AdminClients`).

To configure authorization roles

1. Open your configuration file in a text editor.
2. Find the `[AuthorizationRoles]` section, or create one if it does not exist.
3. In the `[AuthorizationRoles]` section, list the user authorization roles that you want to create. For example:

   ```
   [AuthorizationRoles]
   0=AdminRole
   1=UserRole
   ```

4. Create a section for each authorization role that you listed. The section name must match the name that you set in the `[AuthorizationRoles]` list. For example:

   ```
   [AdminRole]
   ```

5. In the section for each role, define the operations that you want the role to be able to perform. You can set `StandardRoles` to a list of appropriate values, or specify an explicit list of allowed actions by using `Actions`, 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 Web Connector for your changes to take effect.

IMPORTANT: If you do not provide any authorization roles for a standard role, Web Connector uses the default client authorization for the role (localhost for `Admin` and `ServiceControl`, all clients for `Query` and `ServiceStatus`). If you define authorization only by actions, Micro Focus recommends that you configure an authorization role that disallows all users for all roles by default.

For example:

```
[ForbidAllRoles]
StandardRoles=*
Clients=""
```

This configuration ensures that Web Connector uses only your action-based authorizations.

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 Web 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=TLSV1.3
SSLCertificate=host1.crt
SSLPrivateKey=host1.key
```

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

Related Topics

- Start and Stop the Connector, on page 31

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=TLSV1.3
   SSLCertificate=host1.crt
   SSLPrivateKey=host1.key
   ```

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

Related Topics

- Start and Stop the Connector, on page 31

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, Web 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 Web Connector:

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

  where:

Host	is the host name or IP address of the machine where Web 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 Web Connector.

> **NOTE:** You must start and stop the Connector Framework Server separately from the Web 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 Web Connector service is running, enter the following command:

   `ps -aef | grep ConnectorInstallName`

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

Verify that Web Connector is Running

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

- GetStatus
- GetLicenseInfo

GetStatus

You can use the `GetStatus` service action to verify the Web 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 Web Connector setup.

GetLicenseInfo

You can send a `GetLicenseInfo` action to Web 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 Web 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 Web Connector

This section describes how to send actions to Web Connector.

Send Actions to Web Connector

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

Asynchronous Actions

When you send an asynchronous action to Web Connector, the connector adds the task to a queue and returns a token. Web 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 Web Connector with the `QueueInfo` action. For more information about the `QueueInfo` action, refer to the *Web Connector Reference*.

To check the status of an asynchronous action

- Send the `QueueInfo` action to Web 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, Web 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 Web 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 Web 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

Web Connector provides asynchronous actions. Each asynchronous action has a queue to store requests until threads become available to process them. You can configure Web 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.

- On each machine that hosts Web Connector, you must install an ODBC driver for your chosen database. On Linux you must also install the unixODBC driver manager and configure the name and path of the ODBC driver in the unixODBC `odbcinst.ini` configuration file.
- 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 Web Connector

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

To store action queues in an external database

1. Stop Web Connector, if it is running.
2. Open the Web 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=ActionStore`
	or
	`ConnectionString=Driver={PostgreSQL}; Server=10.0.0.1; Port=9876; Database=SharedActions; Uid=user; Pwd=password;`

`MaxVarcharSize=0;`

If your connection string includes a password, Micro Focus recommends encrypting the value of the parameter before entering it into the configuration file. Encrypt the entire connection string. For information about how to encrypt parameter values, see Encrypt Passwords, on page 22.

For example:

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

5. You can use the same database to store action queues for more than one type of IDOL component (for example, a group of File System Connectors and a group of Media Servers). To use a database 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 Web Connectors to share their action queues, set this parameter to the same value in every Web 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=MediaServer
```

6. Save and close the configuration file.

   When you start Web Connector it connects to the shared database.

Store Action Queues in Memory

Web 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 Web 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 Web Connector cleanly, Web Connector writes the action queues from memory to disk so that it can resume processing when it is next started.

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

To store action queues in memory

1. Stop Web Connector, if it is running.
2. Open the Web 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 Web 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 Web Connector, it stores action queues in memory.

Use XSL Templates to Transform Action Responses

You can transform the action responses returned by Web 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 Web 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 Web 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 Web 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, Web Connector applies the XSL template `myTemplate` to the response from a `QueueInfo` action.

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

Example XSL Templates

Web 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 Information by Crawling the Web

This section describes how to retrieve content from a Web site by crawling a site (following links from one page to another).

> **TIP:** *IDOL Expert* includes a full step-by-step guide that demonstrates how to ingest information from a website. The *IDOL Expert* guide discusses strategies for selecting the content to crawl and ingest, and includes an example configuration that you can run.

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.

`Url`	The URL of the page to start crawling from. You can specify multiple URLs to start crawling from by setting a comma-separated list of values or using a numbered list of parameters (`Url0=`, `Url1=`, and so on).
`Depth`	(Optional) The maximum depth to which the connector follows links when crawling. For example, to index all pages that can be reached from the `Url` by following no more than three links, set **`Depth=3`**. The default value of this parameter (**`Depth=-1`**) specifies no limit.
`StayOnSite`	(Optional) A Boolean value that specifies whether the connector stays on the Web site identified by the `Url` parameter. To allow the connector to follow links to other Web sites, set this parameter to **`false`**.
`SpiderUrlCantHaveRegex`	(Optional) A regular expression to restrict the pages that are crawled by the connector. If the full URL of a page matches the regular expression, the page is not downloaded, crawled, or ingested.
`SpiderUrlMustHaveRegex`	(Optional) A regular expression to restrict the pages that are crawled by the connector. The full URL of a page must match the regular expression, otherwise it is not downloaded, crawled, or ingested.

For example:

```
[MyTask]
Url=http://www.example.com/
StayOnSite=true
SpiderUrlCantHaveRegex=.*subdomain\.example\.com.*
```

6. (Optional) If you want to index specific pages, you can use the following configuration parameters:

`PageCantHaveRegex`	A regular expression to restrict the content retrieved by the connector. If the content of a page matches the regular expression, the page is not ingested. This parameter applies to all file types.
`PageMustHaveRegex`	A regular expression to restrict the content retrieved by the connector. The content of a page must match the regular expression, otherwise the page is not ingested. This parameter applies to all file types.
`UrlCantHaveRegex`	A regular expression to restrict the content retrieved by the connector. If the full URL of a page matches the

	regular expression, the page is not ingested. This parameter applies to all file types, for example HTML pages, images, text documents, and so on.
`UrlMustHaveRegex`	A regular expression to restrict the content retrieved by the connector. The full URL of a page must match the regular expression, otherwise the page is not ingested. This parameter applies to all file types, for example HTML pages, images, text documents, and so on.

7. (Optional) If the connector is installed on a machine that is behind a proxy server, see Retrieve Information through a Proxy Server, on page 51.
8. Save and close the configuration file.

Related Topics

- Retrieve Information using a Sitemap, on page 47
- Schedule Fetch Tasks, on page 73
- Choose the Content to Index, below

Choose the Content to Index

When you configure a fetch task to retrieve information from the Web, you can exclude pages from being downloaded and crawled, and exclude pages from being ingested.

- The configuration parameters `SpiderUrlMustHaveRegex` and `SpiderUrlCantHaveRegex` exclude pages from being downloaded and crawled for links.
- The configuration parameters `UrlMustHaveRegex` and `UrlCantHaveRegex` exclude pages from being ingested.

There is an important difference between these two pairs of parameters. Pages can be crawled (the links on the page are followed by the connector) but not ingested.

For example, consider the following site structure:

```
index.html
 |- products/software.html
 |     |- products/software/product1.html
 |     |- products/software/product2.html
 |     |- products/software/product3.html
 |
 |- products/hardware.html
       |- products/hardware/hardware1.html
       |- products/hardware/hardware2.html
       |- products/hardware/hardware3.html
```

If you set **`SpiderUrlCantHaveRegex=.*software\.html`**, the connector does not download or crawl the page `products/software.html`, so the links to the pages `product1.html`, `product2.html`, and `product3.html` are not followed. The pages highlighted below are therefore not ingested:

```
index.html
 |- products/software.html
 |    |- products/software/product1.html
 |    |- products/software/product2.html
 |    |- products/software/product3.html
 |
 |- products/hardware.html
      |- products/hardware/hardware1.html
      |- products/hardware/hardware2.html
      |- products/hardware/hardware3.html
```

TIP: Site structures are usually more complex than the example shown here. If the page `hardware1.html` contained a link to `product1.html`, `product1.html` would still be crawled and ingested.

Alternatively, if you set **`UrlCantHaveRegex=.*software\.html`**, the connector does not ingest the page `products/software.html`, but the page is still crawled for links. The pages `product1.html`, `product2.html` and `product3.html` do not match the regular expression and so they are still ingested. Only the single page highlighted below is excluded from being ingested:

```
index.html
 |- products/software.html
 |    |- products/software/product1.html
 |    |- products/software/product2.html
 |    |- products/software/product3.html
 |
 |- products/hardware.html
      |- products/hardware/hardware1.html
      |- products/hardware/hardware2.html
      |- products/hardware/hardware3.html
```

Choose the Content to Index with a Lua Script

Web Connector supports dynamic corpus functionality. This means that you can use IDOL analytics such as categorization to decide whether to ingest content. You can also use this feature to filter the links that are extracted from a page. The connector runs a Lua script to decide whether to ingest the page and which links to follow, so you can also implement a custom algorithm for deciding which pages to index.

NOTE: This feature is available only if your Web Connector license includes dynamic corpus functionality.

The script must contain a function named `shouldIngestPage` that returns `true` to ingest the page or `false` to ignore it. You can optionally return a list of links to override the links were extracted from the page by the connector. For example, if you want to ingest the page but not follow any of the links on the page, you can return `true` but specify an empty list.

The function should look like this:

```
function shouldIngestPage(url, contentType, contentFilename, textContentFilename,
links, depth)
  -- do something to decide return value...

  -- to ingest the page and follow links extracted by the connector
  return true

  -- to ingest the page but not follow any links
  return true, {}

  -- to ignore the page
  return false
end
```

The arguments supplied to the function are:

Argument	Type	Description
`url`	string	The page URL.
`contentType`	string	The MIME content type.
`contentFilename`	string	The path to the file that contains the page content.
`textContentFilename`	string	The path to the file that contains the text that was extracted from the page (or `nil` if text could not be extracted).
`links`	list of strings	The links that were extracted from the page.
`depth`	integer	The page depth (the number of links that were followed from the starting point in order to reach the page).

An example script, `FilterPages_binarycat.lua`, is included with the connector. This script decides whether to ingest a page by calling the IDOL Category component and running the action `BinaryCatQuery`.

To configure the connector to run your script, set the configuration parameter `FilterPagesLuaScript`.

Retrieve Information using a Sitemap

This section describes how to retrieve content from a Web site by retrieving URLs contained in a sitemap. If you configure the connector to use a sitemap, the connector does not follow links between pages.

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. Choose whether to provide the URL of the sitemap, or configure the connector to automatically find the sitemap by reading the robots.txt file.

   - Specify the URL of the sitemap:

SitemapUrl	The URL of a sitemap that lists the pages to ingest. If you set this parameter, only the pages on the sitemap are ingested. The connector does not crawl the site by following links.

   - Configure the connector to find the sitemap(s) from robots.txt:

UseSitemapFromRobots	To configure the connector to look in robots.txt for the URL of the sitemap, set this parameter to **TRUE**. If the connector cannot find the location of the sitemap in robots.txt, it falls back to crawling the site from the specified URL.
Url	The URL of the site. You must specify only one URL.

   For example:

   ```
   [MyTask]
   UseSitemapFromRobots=TRUE
   Url=https://www.hpe.com/
   ```

6. (Optional) If you want to index specific pages, you can use the following configuration parameters:

PageCantHaveRegex	A regular expression to restrict the content retrieved by the connector. If the content of a page matches the

	regular expression, the page is not ingested. This parameter applies to all file types.
`PageMustHaveRegex`	A regular expression to restrict the content retrieved by the connector. The content of a page must match the regular expression, otherwise the page is not ingested. This parameter applies to all file types.
`UrlCantHaveRegex`	A regular expression to restrict the content retrieved by the connector. If the full URL of a page matches the regular expression, the page is not ingested. This parameter applies to all file types, for example HTML pages, images, text documents, and so on.
`UrlMustHaveRegex`	A regular expression to restrict the content retrieved by the connector. The full URL of a page must match the regular expression, otherwise the page is not ingested. This parameter applies to all file types, for example HTML pages, images, text documents, and so on.

7. (Optional) If the connector is installed on a machine that is behind a proxy server, see Retrieve Information through a Proxy Server, on page 51.
8. Save and close the configuration file.

Related Topics

- Retrieve Information by Crawling the Web, on page 42
- Schedule Fetch Tasks, on page 73

Retrieve Information using a URL File

This section describes how to retrieve content from the Web by providing the connector with a list of URLs. When you provide a list of URLs (a URL file), the connector does not follow links to other pages.

Providing a list of URLs is usually impractical for a large site, but you might want to do this if you have an external process generating the URLs. You must create a text file that contains the URLs of the pages to ingest, with one URL on each line.

You can update the URL file without changing the connector's configuration. If a URL is removed from the file, the connector sends an ingest-delete for that page on the next synchronize cycle.

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:

`SitemapFile`	The path to a plain text file that contains a list of URLs of pages to ingest. The file must contain one URL on each line.
`UrlCantHaveRegex`	(Optional) A Perl-compatible regular expression to restrict the content retrieved by the connector. If the full URL of a page matches the regular expression, the page is not ingested. You can set this parameter to filter the list of URLs.
`UrlMustHaveRegex`	(Optional) A Perl-compatible regular expression to restrict the content retrieved by the connector. The full URL of a page must match the regular expression, otherwise the page is not ingested. You can set this parameter to filter the list of URLs.

For example:

```
[MyTask]
SitemapFile=my-list-of-urls.txt
RemoveNoscripts=TRUE
RemoveScripts=TRUE
SynchronizeThreads=5
PageDelay=5s
```

For a complete list of configuration parameters that you can use, refer to the *Web Connector Reference*.

6. (Optional) If the connector is installed on a machine that is behind a proxy server, see Retrieve Information through a Proxy Server, on the next page.
7. Save and close the configuration file.

Related Topics

- Schedule Fetch Tasks, on page 73

Retrieve Information through a Proxy Server

If the connector is installed on a machine that is behind a proxy server, use one of the following procedures to configure the connector so that it can reach the Web.

To configure the proxy server to use by host name and port

1. In the configuration file, add the following parameters to your fetch task:

`ProxyHost`	The host name or IP address of the proxy server to use.
`ProxyPort`	The port of the proxy server to use.
`ProxyUsername`	The user name to use to authenticate with the proxy server.
`ProxyPassword`	The password to use to authenticate with the proxy server.

   For example:

```
[MyTask]
...
ProxyHost=10.0.0.1
ProxyPort=80
ProxyUsername=username
ProxyPassword=password
```

2. Save and close the configuration file.

To configure the proxy server to use with a proxy automatic configuration script

1. In the configuration file, add the following parameter to your fetch task:

`ProxyAutoConfigurationUrl`	The URL of a proxy automatic configuration script, accessible from the connector machine, that specifies the proxy settings to use for accessing the Web.

   For example:

```
ProxyAutoConfigurationUrl=http://proxy.domain.com/proxy.pac
```

2. Save and close the configuration file.

Provide a Client-Side SSL Certificate

Most web sites do not require clients to provide an SSL certificate. To crawl a site that requires a client-side certificate, set the configuration parameters `SSLClientCertificatePath` and `SSLClientCertificatePassword`.

> **NOTE:** The file specified by `SSLClientCertificatePath` must be in PKCS12 format and contain both the client-side certificate and private key. `SSLClientCertificatePassword` specifies the password for the private key.

You can set these parameters in the task section, for example:

```
[MyTask]
Url=...
SSLClientCertificatePath=my_cert.p12
SSLClientCertificatePassword=encrypted-password
```

Alternatively, you can set the parameters in a separate section and then refer to that section by setting `ClientCertificateSections`. This allows you to specify different certificates for different sites. For example:

```
[MyTask]
Url=...
StayOnSite=False
ClientCertificateSections=ClientCert1,ClientCert2

[ClientCert1]
SSLClientCertificateAuthorityRegex=.*site1\.example\.com.*
SSLClientCertificatePath=my_cert.p12
SSLClientCertificatePassword=encrypted-password

[ClientCert2]
SSLClientCertificateAuthorityRegex=.*site2\.example\.com.*
SSLClientCertificatePath=a_different_cert.p12
SSLClientCertificatePassword=encrypted-password
```

Incremental Synchronization

The Web Connector supports incremental synchronization. The first time the connector synchronizes with a web site, it ingests all of the content that is requested by the task configuration. Subsequently, the connector only ingests information that is new or has changed.

The connector stores the date and time when it requests a page. The next time it requests the page, it uses this information to set the `If-Modified-Since` header in the request sent to the web server. If the

web server supports the `If-Modified-Since` header and does not consider the page to have been modified, it returns an HTTP 304 response code, which tells the connector that the page has not changed. Otherwise, it returns the page.

If a page is returned, the connector performs a check to determine whether the page content has changed. Only if the connector determines that the page has changed does it ingest the page. This check occurs after clipping (if you have configured clipping) so that changes to parts of the page that are clipped do not cause the page to be re-ingested.

> **TIP:** Many web pages have dynamic content, and can appear to change every time they are requested.

Remove Irrelevant Content

Sometimes a page has changes, but the changes are located in part of the page that you consider irrelevant. You can configure the connector to discard irrelevant content so that changes in this content do not cause a page to be re-ingested. Web Connector has several features that you can use to do this:

- Clipping removes irrelevant content, such as headers, footers, navigation bars, and advertisements.
- You can set the following parameters to remove parts of a page:
  - `RemoveComments` removes HTML comments.
  - `RemoveScripts` removes scripts.
  - `RemoveNoframes` and `RemoveNoscripts` remove `<noframes>` and `<noscript>` content.
- You can set the parameter `IngestAsPlainText` so that the connector extracts text from a web page, adds it to the `DRECONTENT` field, and then ingests a metadata-only document (a document without an associated HTML file).

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 *Web Connector Reference*.

Page Depth for New URLs

The `synchronize` action, with the `identifiers` parameter, might be used to synchronize a page with a URL that the connector has never seen before. When you use the `synchronize` action with the `identifiers` parameter, the connector does not crawl the website, so in this case the connector sets the `DEPTH` document field to a large value. The next time the page is synchronized the field is updated with the depth from the start URL. If the page cannot be reached from the start URL then the connector sends an ingest-delete command to remove the document from the IDOL index.

Block Resource Downloads

You can configure Web Connector so that it does not download certain resources, such as scripts or advertisements. This can result in fewer HTTP requests, reduce the amount of time required for a synchronize task, and prevent unwanted content from contaminating your IDOL index.

Resources include stylesheets, JavaScript files, images, and other items that are used to construct a page. You can block resources from being downloaded by regular expression or by using block lists.

To block page resources

- Set the configuration parameters `ResourceUrlCantHaveRegex` and `ResourceUrlMustHaveRegex`. These block resources where the URL of the resource matches, or does not match, a regular expression. For example, you could exclude all resources that do not originate from your own domain:

  ```
  ResourceUrlMustHaveRegex=https://www\.example\.com.*
  ```

- Set the configuration parameter `BlockingFiltersDirectory` to the path of a directory that contains filter lists. You can use filter lists from https://easylist.to, or create your own in the same format. The filter lists are used to block undesirable content.

Clip Pages

The content on most web pages includes headers, footers, navigation bars, and advertisements. Unless these are removed from pages, the text from these items could reduce the quality of the documents indexed into IDOL Server and reduce the effectiveness of operations such as categorization.

You can configure Web Connector to remove irrelevant content from pages before they are ingested.

Clip Pages Automatically

Web Connector can clip pages automatically, using an algorithm to decide which parts of the page to keep and which to discard. To clip pages automatically, add `Clipped=TRUE` to your task configuration.

The algorithm evaluates each section of the page and decides whether to clip it based on several factors, including:

- The position of the section on the page (central content is preferred).
- The ratio of links to words (a smaller proportion of links is preferred).

Automatic clipping works best with common page designs, such as pages where the content is in the center and there are navigation panels to the top and left, with extra content to the right.

Clip Pages using CSS Selectors

The automatic clipping algorithm has been designed to work with many different pages, but this means that automatic clipping might not give the best results for every page. For this reason, you can use CSS selectors to choose which parts of the page to keep and which to discard. To clip pages with CSS selectors, add `Clipped=TRUE` to your task configuration, and then set the parameters `ClipPageUsingCssSelect` and `ClipPageUsingCssUnselect`.

`ClipPageUsingCssSelect`	A comma-separated list of CSS selectors that specify parts of the page to keep. The connector also keeps all descendents of these elements.
`ClipPageUsingCssUnselect`	A comma-separated list of CSS selectors that specify parts of the page to remove. The connector also removes all descendents of these elements. The `ClipPageUsingCssSelect` parameter is applied before `ClipPageUsingCssUnslect`, so you can use this parameter to remove unwanted descendants of elements identified by `ClipPageUsingCssSelect`.

The Web Connector supports standard CSS selectors. To construct the selectors, view the source HTML of the pages that you need to clip. CSS allows you to select elements based on the structure of the page. For example, you can select elements of a certain type that are descendents of another element. Also, the designer of the page might have added classes to the relevant elements in order to style them, and you can use these same classes to clip the page.

The following example shows a simple page:

```
<html>
<head>
</head>
<body>
   <nav>
     <!-- navigation and links -->
   </nav>
   <div class="maincontent">
     <p>Some content</p>
   </div>
   <div class="footer">
     <!-- footer -->
   </div>
```

```
</body>
</html>
```

To select the main content but exclude the navigation element and the footer, you could use the following configuration:

```
[MyTask]
...
Clipped=TRUE
ClipPageUsingCssSelect=div.maincontent
ClipPageUsingCssUnselect=nav,div.footer
```

TIP: Web Connector includes an example tool that can help you find the CSS selectors you need to clip web pages. For more information about this utility, see Find Selectors using the CSS Selector Builder Tool, below.

Find Selectors using the CSS Selector Builder Tool

The CSS Selector Builder Tool is an example tool, provided with Web Connector, that can help you find CSS selectors for parts of a web page. The tool is a plug-in for the Chrome web browser.

NOTE: Micro Focus does not support the CSS Selector Builder Tool, it is provided only as an example of a tool that you could build.

To install the CSS Selector Builder Tool

1. Open the Chrome web browser.
2. In the address bar, type **`chrome://extensions`** and press ENTER.

   The Extensions page opens.
3. Select the **Developer mode** check box.
4. Click **Load unpacked extension...**

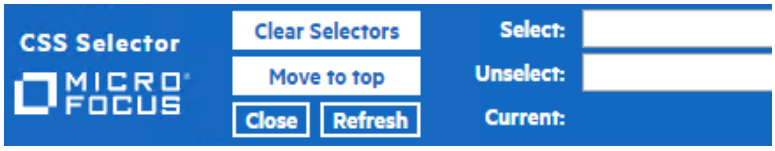
   The Browse For Folder dialog box opens.
5. Select the CSS Selector Builder Tool folder in the Web Connector installation directory, and click **OK**.

   The tool is installed and you should see **Micro Focus CSS Selector Builder Tool** listed as an extension.

To find CSS Selectors

1. In Chrome, navigate to the web site that you want to index.
2. On the toolbar, click the **CSS Selector Builder Tool**  button.

The tool opens at the bottom of the browser window (but you can move it to the top of the window if you prefer, by clicking **Move to top**).



3. Move the mouse pointer over the page. Elements on the page are highlighted in blue as you hover over them, and the selector for the current element is displayed in the **Current** field.
4. Choose the page elements to keep and discard:
   - Right-click on each element that you want to keep when the page is clipped. The page elements to keep are highlighted in green and selectors are added to the **Select** box.
   - Right-click twice on each element that you want to discard when the page is clipped. The page elements to discard are highlighted in red and selectors are added to the **Unselect** box.
   - You can type one or more selectors in the **Select** and **Unselect** boxes and press **Refresh**. The tool will then show which parts of the page are clipped based on the selectors you have entered.
   - To clear all selectors, click **Clear Selectors**.
5. (Optional) Click on any hyperlink to navigate around the site. The tool highlights the selected and unselected elements on the current page, so that you can see whether the selectors you have chosen clip multiple pages as desired.
6. When you have optimized your selections, use the value in the **Select** box to set the parameter `ClipPageUsingCssSelect` in the connector's configuration file, and the value in the **Unselect** box to set the parameter `ClipPageUsingCssUnselect`.
7. Click **Close**.

Choose How Long to Wait for Scripts

The Web Connector allows time for scripts to complete before processing a page. However, some web pages run scripts that never finish. For example, a page might contain a script that performs an AJAX request and then schedules itself to run again five seconds later. During the five-second interval between the script completing and running again, the script is inactive. The connector can determine that scripts are inactive, but it is impossible for the connector to determine whether the page has completely loaded or whether the script might add more content.

When the Web Connector detects that scripts have been inactive for a certain period of time, by default half a second, it considers a page to have finished loading. You can configure the duration of inactivity required by setting the parameter `ScriptActivityTimeout`. For example, if you set `ScriptActivityTimeout=1s`, all scripts must be inactive for a second before the connector processes the page.

> **NOTE:** Micro Focus recommends setting a short duration for this parameter, because the connector

> must wait for at least the specified amount of time for every page that contains scripts. Increasing the duration of `ScriptInactivityTimeout` could significantly increase the amount of time required to complete a synchronize cycle.

If scripts do not become inactive or remain inactive for long enough, the connector eventually terminates the scripts. You can configure the maximum amount of time to wait for scripts by setting the configuration parameter `MaxWaitForScripts`. For example, to wait no longer than 10 seconds, set `MaxWaitForScripts=10s`. This time limit does not affect custom scripts specified by the `CustomJSScriptPath` configuration parameter.

As described above, some web pages run scripts that never finish, so by default the Web Connector does not consider having to terminate scripts as an error. The connector continues to process the page as if the scripts had completed successfully. In some cases, scripts are used to generate content and add links, so terminating a script could result in content being unreachable. You can therefore configure the connector to report an error and attempt to process the page again on the next synchronize cycle. To do this, set the configuration parameter `FailOnScriptTimeout=TRUE`.

Log On to a Web Site

Some Web sites require that you log on before you can access content. To retrieve content from these Web sites, configure the connector to log on to the site.

Basic, HTTP Digest, and NTLMv2 Authentication

To log on to a Web site that uses Basic, HTTP Digest, or NTLM version 2 authentication, specify a user name and password by setting the configuration parameters `AuthUser` and `AuthPassword`. For example:

```
[MyTask]
Url=http://www.example.com/
AuthUser=user
AuthPassword=pass
```

Alternatively, you can set the `AuthSection` parameter and set the `AuthUser` and `AuthPassword` parameters in a different section of the configuration file. This means you can use the same user name and password for several tasks:

```
[MyTask]
Url=http://www.example.com/
AuthSection=MyAuthSection

[AnotherTask]
Url=http://www.another-example.com/
AuthSection=MyAuthSection

[MyAuthSection]
AuthUser=user
AuthPassword=pass
```

If you create a fetch task to crawl more than one site, or you need to specify more than one set of credentials for a site, you can use multiple sections containing authentication details. The `AuthSection` parameter accepts multiple values. Set the `AuthUrlRegex` parameter in each section to specify the URLs that the authentication details can be used against. For example:

```
[MyTask]
Url=http://www.example.com/
AuthSection0=LogOnAuthExample
AuthSection1=LogOnAuthExampleSubDomain

[LogOnAuthExample]
AuthUrlRegex=.*www\.example\.com/.*
AuthUser=MyUsername
AuthPassword=MyP4ssw0rd

[LogOnAuthExampleSubDomain]
AuthUrlRegex=.*subsite\.example\.com/.*
AuthUser=MySubsiteUsername
AuthPassword=MySubsiteP4ssw0rd
```

Submit a Form

If the Web site does not use Basic, HTTP Digest or NTLMv2 authentication, the connector might be able to log on by submitting a form.

Configure the connector to submit a form by setting the following configuration parameters:

`FormUrlRegex`	A regular expression to identify the page that contains the log-on form. The connector does not attempt to submit form data unless the URL of a page matches the regular expression.
`InputSelector`	A list of CSS selectors to identify the form fields to populate. Specify the selectors in a comma-separated list or by using numbered parameters.
`InputValue`	The values to use for the form fields specified by the `InputSelector` parameter. Specify the values in a comma-separated list or by using numbered parameters.
`SubmitSelector`	A CSS selector that identifies the form element to use to submit the form.
`ValidateFormData`	A Boolean value that specifies whether the connector attempts to validate the data supplied to complete a form. The connector can validate the data based on the types of the input elements.

You can set these parameters in the `[`*`TaskName`*`]` section of the configuration file, for example:

```
[MyTask]
Url=http://www.example.com/
FormUrlRegex=.*login\.php
InputSelector0=input[name=username]
InputSelector1=input[name=password]
```

```
InputValue0=MyUsername
InputValue1=MyP4ssw0rd!
SubmitSelector=input[name=login]
```

To specify the information in a separate section of the configuration file, set the `FormsSection` parameter:

```
[MyTask]
Url=http://www.example.com/
FormsSection=LogOnForm

[LogOnForm]
FormUrlRegex=.*login\.php
InputSelector0=input[name=username]
InputSelector1=input[name=password]
InputValue0=MyUsername
InputValue1=MyP4ssw0rd!
SubmitSelector=input[name=login]
```

To submit different forms during a single task, you can create multiple sections containing form settings. The `FormsSection` parameter accepts multiple values. In each section, use the `FormUrlRegex` parameter to identify the page that contains the form:

```
[MyTask]
Url=http://www.example.com/
FormsSection0=LogOnFormExample
FormsSection1=LogOnFormExampleSubDomain

[LogOnFormExample]
FormUrlRegex=.*www\.example\.com/.*login\.php
InputSelector0=input[name=username]
InputSelector1=input[name=password]
InputValue0=MyUsername
InputValue1=MyP4ssw0rd!
SubmitSelector=input[name=login]

[LogOnFormExampleSubDomain]
FormUrlRegex=.*subsite\.example\.com/.*login\.php
InputSelector0=input[name=username]
InputSelector1=input[name=password]
InputValue0=MySubsiteUsername
InputValue1=MySubsiteP4ssw0rd!
SubmitSelector=input[name=login]
```

Authenticate using Cookies

To authenticate with a site that uses cookies, use the configuration parameters `CustomHeaderName` and `CustomHeaderValue` to send the cookie data to the web server with each request. For example:

```
[MyTask]
Url=https://my-site/index.htm
CustomHeaderName0=Cookie
CustomHeaderValue0=session=abc123;userID=12345
```

Limit the Download Rate

To reduce the load on the Web site you are indexing, you can limit the maximum rate at which the connector downloads content, using the configuration parameter `MaxKBytesPerSec`. This parameter does not limit the download rate for any individual page, it limits the maximum download rate by pausing between pages.

A Web server might be configured to deny requests that arrive from the same host with little or no time between them. The connector requests information from a Web site much faster than a human visitor, so you might find that the connector's requests are denied. To prevent a server from denying requests from the connector, you can specify a time interval that a synchronize thread must wait between requests. To do this, set the configuration parameter `PageDelay`. If you set a delay of one second, each synchronize thread must wait at least one second between requests. By default, the connector uses five synchronize threads so might request up to five pages per second.

You can also limit the maximum number of pages that are ingested in a single synchronize cycle, by setting the configuration parameter `MaxPages`. If the limit is reached, the remaining pages are ingested in future synchronize cycles.

> **TIP:** For more information about these configuration parameters, refer to the *Web Connector Reference*.

Retrieve Recently Updated Content

In some cases you might want to retrieve information from the Web only when it has been recently updated. For example, if you are crawling a news site, you might want to restrict the synchronize task to retrieve pages that were updated in the last 30 days.

To retrieve Web pages based on the date, the Web Connector must be able to extract a date for each page. The connector checks the following sources (in order) and uses the first valid date that it finds:

- The URL of the page (but only if `DateInUrl` is set to `TRUE`).
- The page content (but only if you set `PageDateSelector`).
- The HTTP headers returned with the page. The connector attempts to extract a date from the headers named by the parameter `PageDateHeader`.

To retrieve pages based on the date

1. Stop the connector and open the configuration file.
2. Modify your fetch task by adding the relevant parameters from the following list, so that the connector can extract the date associated with a page:

`DateInUrl`	A Boolean value that specifies whether the date associated with a Web page can be extracted from the page URL. When you set this parameter to `TRUE`, the connector attempts to extract the date from the URL.
`PageDateSelector`	A list of CSS selectors to identify elements in the page content that might contain the date associated with a Web page. • If the date is contained in the element's content, setting this parameter is sufficient to extract the date. • If the date is contained in one of the element's attributes, set this parameter and then set `PageDateAttribute` to identify the attribute.
`PageDateAttribute`	A list of attributes (on elements listed by `PageDateSelector`) that contain the date associated with the Web page.
`PageDateHeader`	A list of headers to retrieve the date from. The headers are checked in the order that you specify and the first valid date that the connector finds is associated with the page. The connector only attempts to extract a date from HTTP headers if no date is found in the page URL (or `DateInUrl=FALSE`) and no date is found in the page content (or `PageDateSelector` is not set).
`DateFormats`	A comma-separated list of date formats to use when searching for a date.

3. Set one or both of the following parameters.

`MinPageDate`	Filters the pages that are ingested by date. The connector only ingests pages that are newer than the specified date.
`MaxPageDate`	Filters the pages that are ingested by date. The connector only ingests pages that are older than the specified date.

   You can configure a fixed limit, for example to retrieve pages that were updated after 01 July 2015. Alternatively you can configure a rolling limit, for example to retrieve pages that are not more than 30 days old when the synchronize task starts.
4. (Optional) By default the connector ingests pages where it cannot determine a date. To prevent the connector from ingesting these pages, set the parameter `IngestPagesWithNoDate` to **`False`**.
5. (Optional) By default the connector does crawl pages that are outside your specified date range, even though they are not ingested. To prevent the connector following links from these pages, set the parameter `SpiderDateFilteredPages` to **`False`**.
6. Save and close the configuration file.

Example

The following example configuration retrieves pages that are less than 30 days old, based on a date contained in the URL or in the page content:

```
[MyTask]
Url=http://www.example.com/
...
DateInUrl=true
PageDateSelector=p[id=date]
MinPageDate=-30 days
IngestPagesWithNoDate=False
SpiderDateFilteredPages=False
```

Render an Image, Thumbnail, or PDF for Ingested Pages

The Web Connector can render an image, thumbnail image, or PDF of each Web page that it ingests.

By default, when you configure the connector to create one or more of these files they are each indexed as the document content of a separate document, alongside the indexed Web page. Alternatively, you can configure the connector to write the files to a folder.

If you write rendered images, PDF files, and thumbnails to a folder the connector adds metadata fields to each document that contain the paths of associated files. The fields are named:

- RENDITION_FILE_IMAGE
- RENDITION_FILE_PDF
- RENDITION_FILE_THUMBNAIL

When the connector sends ingest-removes for deleted documents, the connector deletes any rendered images, PDF files and thumbnails associated with those documents.

To render an image, thumbnail, or PDF for each ingested page

1. Stop the connector and open the configuration file.
2. Modify your fetch task by adding the following parameters:

`CreateImageRendition`	To render an image for each ingested page, set this parameter to **`true`**.
`CreateThumbnailRendition`	To render a thumbnail image for each ingested page, set this parameter to **`true`**.

`CreatePDFRendition`	To render a PDF copy of each ingested page, set this parameter to **`true`**.
`RenditionsFilePath`	The path of the folder to write rendered images, PDF files, and thumbnails to. The folder must already exist, and the user running the connector must have permission to write files to the folder. If you don't set this parameter, the connector indexes each file as the content of a separate document.
`FullPageRender`	Specifies whether images and thumbnail images show the full page (**`true`**), or only the top part of the page (**`false`**), that you would see when viewing the page in a web browser.
`RenditionImageFormat`	The image format for images and thumbnail images.
`RenditionImageQuality`	The image quality for JPEG and PNG images and thumbnail images. Specify an integer value from 0 to 100, where lower values represent higher compression (usually resulting in a smaller file size), and higher values represent higher quality.
`ThumbnailRenditionWidth`	The maximum width for thumbnail images, in pixels.
`ThumbnailRenditionHeight`	The maximum height for thumbnail images, in pixels.

3. Save and close the configuration file.

Example

The following example renders a thumbnail for each page, as a PNG image that has a maximum width of 350 pixels:

```
[MyTask]
Url=http://www.example.com/
...
CreateThumbnailRendition=true
RenditionImageFormat=png
ThumbnailRenditionWidth=350
FullPageRender=FALSE
```

Split Web Pages into Multiple Documents

You might want to split Web pages into multiple documents. For example, if you ingest pages from a discussion board you might want to ingest one document for each message on the page.

The Web Connector can create documents for sections of a Web page identified using CSS selectors. For each Web page, the connector creates a parent document and an associated file that contains the full page source. It then creates a child document for each section of the page. Each child document has an associated file that contains the `<head>` element from the original page, and a `<body>` element

containing the content identified by your CSS selector. The parent document includes metadata fields (named `CHILD_DOCUMENT`) that refer to the child documents, and each child document has a metadata field (`PAGE_REFERENCE`) that refers back to the parent document.

To split Web pages into multiple documents

1. Stop the connector and open the configuration file.
2. Modify your fetch task by adding the following parameters:

`ChildDocumentSection`	(Optional) A comma-separated list of sections in the Web Connector configuration file that contain settings for creating child documents. If you do not set this parameter, the connector uses the settings in the *TaskName* section.
`ChildDocumentUrlRegex`	(Optional) A Perl-compatible regular expression to identify the pages to generate child documents from. The connector does not attempt to generate child documents from a page unless the full URL of a page matches the regular expression.
`ChildDocumentSelector`	A CSS selector that identifies the root element of each child document in the page source.
`ChildReferenceSelector`	(Optional) An element in the child document that contains a value to use as the document reference. The value you extract should be unique for each child document, because it is used as part of the `DREREFERENCE` field in the child document. If you do not set this parameter, the connector uses a GUID. Specify the element using a CSS selector, relative to the element identified by `ChildDocumentSelector`.
`ChildMetadataFieldName`	(Optional) The names to use for document fields (in child documents) that contain information extracted using the parameter `ChildMetadataSelector`. This parameter must have the same number of values as `ChildMetadataSelector`.
`ChildMetadataSelector`	(Optional) A list of elements in the child document that contain metadata. The metadata in these elements are extracted and added to the metadata fields of child documents. To specify the name(s) of the document field(s) to contain the extracted information, set the configuration parameter `ChildMetadataFieldName`. Both parameters must have the same number of values. Specify the elements as a list of CSS selectors, relative to the element identified by `ChildDocumentSelector`.

3. Save and close the configuration file.

Example

Consider the following example page which represents messages on a page of a discussion board:

```
<html>
    <head>
        <title>Example Page</title>
        <meta charset="utf-8">
    </head>
    <body>
        <div>
            <h1>Example Page</h1>
            <div class="content">
                <p>content</p>
            </div>
            <div class="message">
                <h1>Message 1</h1>
                <p class="meta">some metadata</p>
                <p>some content</p>
            </div>
            <div class="message">
                <h1>Message 2</h1>
                <p class="meta">some metadata</p>
                <p>some content</p>
            </div>
            ...
        </div>
    </body>
</html>
```

To create separate documents for the messages contained on this page, you could use the following configuration:

```
[MyTask]
...
ChildDocumentSelector=div.message
ChildReferenceSelector=h1
ChildMetadataFieldName0=my_metadata
ChildMetadataSelector0=p.meta
```

This example would produce the following child document:

```
#DREREFERENCE [child][Message 1]http://www.hp.com/example.htm
#DREFIELD my_metadata="some metadata"
#DREFIELD PAGE_REFERENCE="http://www.hp.com/example.htm"
```

The `PAGE_REFERENCE` field is added automatically by Web Connector and contains the reference of the parent document.

The following file is associated with the child document. The Web Connector includes the `<head>` element from the Web page and adds the content identified by `ChildDocumentSelector` to the `<body>` element. If you send your documents to CFS, this file is filtered by KeyView and the information it contains is used as the document content.

```
<html>
    <head>
        <title>Example Page</title>
```

```
        <meta charset="utf-8">
    </head>
    <body>
        <div class="message">
            <h1>Message 1</h1>
            <p class="meta">some metadata</p>
            <p>some content</p>
        </div>
    </body>
</html>
```

A similar document and file would be ingested for the second message.

The parent document contains the full page content. The Web Connector automatically adds fields named `CHILD_DOCUMENT`, containing the references of associated child documents:

```
#DREREFERENCE http://www.hp.com/example.htm
...
#DREFIELD CHILD_DOCUMENT="[child][Message 1]http://www.hp.com/example.htm"
#DREFIELD CHILD_DOCUMENT="[child][Message 2]http://www.hp.com/example.htm"
...
```

Process XML Feeds

Web sites might have pages, such as RSS feeds, that contain XML rather than HTML. This section describes how you can use Web Connector to process XML pages.

Web Connector identifies XML pages by the MIME type returned in the `Content-Type` header of the response from the web server.

XML pages can include processing instructions that instruct a web browser to apply an XSL transformation to the page, and sometimes the XML is transformed into HTML. The processing instruction looks similar to this:

```
<?xml-stylesheet type="text/xsl" href="transform_to_html.xsl"?>
```

If the `Content-Type` header indicates that the page contains XML but no XSL transformation is provided, the connector ingests the page as an XML document.

If the header indicates that page contains XML and an XSL transformation is provided, the connector applies the transformation and processes the page as if it were HTML. This means that the connector can:

- extract links from the page, and follow those links to retrieve more content.
- split the page into multiple documents.
- clip the page.
- render an image, thumbnail image, or a PDF of the page.

As a result, you can use the Web Connector to process an RSS feed that is transformed into HTML. The Web Connector, like the RSS Connector, retrieves the content contained in the feed, such as page

titles and summaries. In addition, the Web Connector can follow the links contained in the feed and ingest the content on the associated pages.

In the Web Connector task configuration, specify the URL of the feed using the `Url` parameter and set `Depth=1`. Setting `Depth=1` ensures that the connector follows the links from the RSS feed, but does not follow any links that are extracted from the associated web pages:

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

[RssFeed]
Url=http://www.example-news-website.com/feed/rss.xml
Depth=1
```

When the Web Connector applies an XSL transformation to an XML document and logging is configured with `LogLevel=Full`, you will see the following messages in the synchronize log:

```
WKOOP:Applying XSL transfrom
WKOOP:XSL Transform applied to XML document
```

Detect Errors

The IDOL Web Connector makes a request to a web server for each of the pages that you want to process. The response to each request includes an HTTP status code, usually "200" which indicates that the request was successful.

If a problem occurs, the web server should provide an appropriate HTTP status code to the connector, for example "404" if the requested page is not found or "500" if there is a server error.

However, some web servers return an HTTP 200 status code even if there is a problem. Sometimes the web server returns a page that contains an error message. If you are indexing a web site that behaves in this way, you can configure error detection so that the connector can take appropriate action.

To detect error messages

1. Stop the connector and open the configuration file.
2. Modify your fetch task by adding the parameter `ErrorPageSections`. This parameter specifies the names of the sections in the connector's configuration file that contain settings for error detection. For example:

   ```
   [MyTask0]
   ...
   ErrorPageSections0=LoginError
   ```
3. Create a new section in the configuration file, using the name you specified in the previous step, and set the following parameters:

`ErrorPageUrlRegex`	A Perl-compatible regular expression to match the full URLs of the pages on which you want to run error detection.
`ErrorPageCssSelector`	A CSS selector that identifies an element that must exist on the page, for the connector to detect an error. If you do not set this parameter, the connector detects an error for any page that matches `ErrorPageUrlRegex`.
`ErrorPageAttribute`	The name of an attribute that must exist on the element specified by `ElementPageCssSelector`, for the connector to detect an error.
`ErrorPageRegex`	A Perl-compatible regular expression that the value of the selected element or attribute must match, for the connector to detect an error.
`ErrorPageHttpErrorCode`	An HTTP status code that specifies how the connector behaves when an error is detected. For example, if you set this parameter to `401`, the connector behaves as if there is an authentication error and the synchronize task (correctly) fails.

For example, the following configuration detects errors on any page named `login.html`. The connector detects an error when there is an element in the page with an ID of `authresult`, and the element contains the text `failed`.

```
[LoginError]
ErrorPageUrlRegex=.*/login.html
ErrorPageCssSelector=#authresult
ErrorPageRegex=.*failed.*
ErrorPageHttpErrorCode=401
```

One example of a matching element would be:

```
<h1 id="authresult">Login failed</h1>
```

For more information about the configuration parameters you can use to detect and handle errors, refer to the *Web Connector Reference*.

4. Save and close the configuration file.

Handle JavaScript Dialogs and Alerts

Some web sites use JavaScript to present dialog boxes or alert messages. You can configure Web Connector to handle these. The connector automatically dismisses any dialog that you do not handle.

The following example handles a dialog box on a page named `prompt.htm`. Web Connector responds in the same way as a person who enters the text "Here is my response" and clicks the "OK" button.

```
[MyTask]
...
JsDialogSections=JavaScriptDialog0
```

```
[JavaScriptDialog0]
JsDialogUrlRegex=.*/prompt.htm
JsDialogPromptText=Here is my response
JsDialogAccept=TRUE
```

For more information about the configuration parameters that you can use to handle JavaScript dialog boxes, refer to the *Web Connector Reference*.

Scripted Processing

The *scripted processing* features of IDOL Web Connector allow you to control the embedded browser while specific pages are being processed. The browser is controlled by means of a custom Lua script.

A common feature on many web sites is *paging*. Imagine a site presents a list of items but there are too many to show on a single page, so the site presents the first ten and provides a link to the next page. When you click "next", some web sites reload the page. Your browser sends a new request to the web server, perhaps with an additional query parameter that specifies the current position in the list, so that the server returns the next ten items. After clicking "next" a few times, you might see page URLs like:

```
http://www.example.com/articles.htm
http://www.example.com/articles.htm?start=10
http://www.example.com/articles.htm?start=20
http://www.example.com/articles.htm?start=30
```

The IDOL Web Connector considers each of these to be separate pages, so each is crawled for links. The connector follows the link to the next page, and the following one, and so on, creating a separate IDOL document for each page.

Some web sites take a different approach and present the next "page" of content without reloading the page. The "next" button probably isn't a hyperlink at all. When you click the button, an event handler might run some JavaScript that performs an HTTP request and inserts new content into the existing page. No navigation takes place and the page URL does not change. Sometimes these sites might be referred to as using Asynchronous JavaScript and XML (AJAX) techniques.

On sites like these, automated web crawlers might fail to retrieve all of the content. You can use the scripted processing features of IDOL Web Connector to simulate clicking the "next" button and instruct the connector to ingest the page after each click.

To configure scripted processing, set the configuration parameters `ScriptedProcessingUrlRegex` and `ScriptedProcessingLuaScript`. `ScriptedProcessingUrlRegex` identifies the pages on which to use scripted processing. `ScriptedProcessingLuaScript` specifies the path of a Lua script to use for controlling the embedded browser. For example:

```
[MyTask]
ScriptedProcessingSections=ScriptedProcessing

[ScriptedProcessing]
ScriptedProcessingUrlRegex=.*/js_paged_index.html
ScriptedProcessingLuaScript=js_paged_index.lua
```

The Lua script that you write must define a function with the signature `processPage(url, session)`. The `url` argument is a string, containing the URL of the current page. The `session` argument is a LuaClientSession object that provides methods for interacting with the page.

The following is an example implementation that performs the following steps:

1. Creates a variable to contain the page number and sets this to 1.
2. Ingests the page.
3. Enters a loop that performs the following steps until the "next" button is disabled:
   a. Increments the page number.
   b. Clicks the "next" button.
   c. Waits until downloads have finished and the DOM stops changing.
   d. Ingests the page.

```
g_nextButtonCssSelector = "#next-button"
g_quietIntervalMs = 250
g_quietTimeoutMs = 10000

function nextPageButtonActive(taskLog, session)
    local elementsCount = session:countElements(g_nextButtonCssSelector .. ":not([disabled])")
    return elementsCount > 0;
end

function processPage(url, session)
    local taskLog = get_task_log()
    taskLog:full("Scripted Processing: " .. url)

    local pageNumber = 1

    taskLog:full("Snapshotting Page: " .. pageNumber)
    session:snapshotPage("Page" .. tostring(pageNumber))

    while nextPageButtonActive(taskLog, session) do
        pageNumber = pageNumber + 1
        session:clickElement(g_nextButtonCssSelector)
        session:waitForQuiet(g_quietIntervalMs, g_quietTimeoutMs)

        taskLog:full("Snapshotting Page: " .. pageNumber)
        session:snapshotPage("Page" .. tostring(pageNumber))
    end

    taskLog:full("Scripted Processing complete: " .. tostring(pageNumber) .. " pages")
end
```

Run Custom JavaScript on Web Pages

The connector can run custom JavaScript on web pages before it crawls and ingests them. This allows you to manipulate pages or interact with them before they are processed. For example, if a page has a button to load more content you can configure the connector to run a script that clicks the button. The ingested page will then include the additional content.

> **TIP:** Running custom JavaScript on pages is usually only necessary in advanced cases, such as processing unconventional sites.

You must write the custom script and save it to a file that can be accessed by the connector.

Micro Focus recommends that you enclose your script in a `try...catch` block to prevent exceptions being thrown out of the script. For example:

```
try
{
  // custom javascript
  true;
}
catch(e)
{
  console.log("Javascript Error: " + e.message) false;
}
```

Any messages that you write to the JavaScript console are written to the connector's synchronize log, but only when you set the parameter `LogLevel` to **FULL** and `EnableJavascriptConsoleLogging` to **TRUE**. To troubleshoot your script, set these parameters and include a reasonable amount of logging in your script. The connector also logs the return value of the script.

To run custom JavaScript on web pages

1. Stop the connector and open the configuration file.
2. Modify your fetch task by adding the parameter `CustomJSSections`. This parameter specifies the names of the sections in the connector's configuration file that contain settings for running custom JavaScript on pages. For example:

   ```
   [MyTask0]
   ...
   CustomJSSections=MyCustomJS
   ```
3. Create a new section in the configuration file to contain the settings for running the script:

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

`CustomJSUrlRegex`	A Perl-compatible regular expression to match the full URLs of the pages on which you want to run custom JavaScript.
`CustomJSScriptPath`	The path of the file that contains the custom JavaScript to run. The path must be absolute, or relative to the Web Connector executable file.
`CustomJSElementSelector`	(Optional) The page elements on which to run the custom JavaScript. Specify the elements using CSS selectors.
`CustomJSAllSelected`	(Optional) A Boolean value that specifies whether to run the custom JavaScript on all elements that match the selector specified by `CustomJSElementSelector` (the default value, `false`, means that the connector runs the JavaScript only on the first matching element).

For example:

```
[MyCustomJS]
CustomJSUrlRegex=http://www\.hpe\.com/.*
CustomJSScriptPath=javascript/my_script.js
CustomJSElementSelector=div#maincontent
```

5. Save and close the configuration file.

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**. Tasks scheduled to start at the same time run in the order that they are enumerated by the *N* parameter (in the `[FetchTasks]` section of the configuration file).
`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

- Start and Stop the Connector, on page 31

Troubleshoot the Connector

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

The synchronize log shows the error "Address already in use"

You might see this error if the system has allocated all available TCP ports.

Operating systems typically allocate a limited number of ports to applications that need to make outbound connections. Also, when a connection is closed, the operating system waits before a port is released and can be reused.

If you encounter this issue on a Windows system, you can set the Windows registry parameter `MaxUserPort`, so that more ports are available for use. You could also shorten the amount of time that the operating system waits before releasing a port by setting the registry parameter `TcpTimedWaitDelay`. These are both set in:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
```

Chapter 7: 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 or another connector. 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.
- 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*.

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 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>
```

The `ValueFormat` element has the following attributes:

`type`	To convert the date into the IDOL AUTNDATE format, specify `autndate`. To convert the date into a custom format, specify `customdate` and then set the attribute `targetformat`.
`format`	The format to convert the date from. Specify the format using standard IDOL date formats.
`targetformat`	The format to convert the date into, when you set the `type` attribute to `customdate`. Specify the format using standard IDOL date formats.

As demonstrated by the previous 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 `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

- Write a Lua Script, on the previous page

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
```

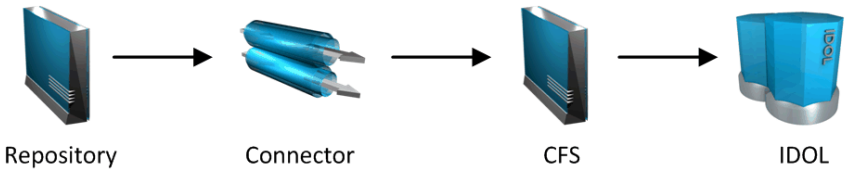
Chapter 8: Ingestion

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 Web 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 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*.



- **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 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

Web 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

- Web 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 Web 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, Web 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, Web 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 Web 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 Send Data to Connector Framework Server, on page 89

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 *Web 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 9: 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

Web Connector includes the `admin.dat` file that is required to run IDOL Admin.

IDOL Admin supports the following browsers:

- 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

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

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

  where:

host is the host name or IP address of the machine where the IDOL component is installed.

port is the ACI port of the IDOL component you want to administer.

View Connector Statistics

Web 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`

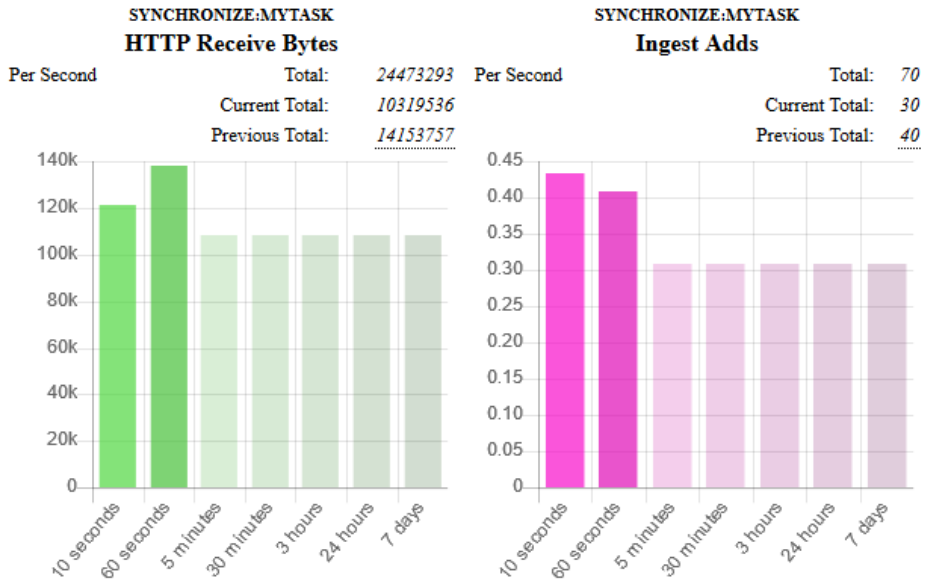
When you are using the `ConnectorStatistics` template, you can also add the `filter` parameter to the request to return specific statistics. The `filter` parameter accepts a regular expression that matches against the string *autnid*::*name*, where *autnid* and *name* are the values of the corresponding 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 Web 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.
`Identifiers`	Messages related to the `Identifiers` fetch action.
`Synchronize`	Messages related to the `Synchronize` 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 Web 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 *Web 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>MTAuMi4xMDUuMTAzOjEyMzQ6RkVUQ0g6MTAxNzM0MzgzOQ==</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 Web Connector faster than they can be processed.

Web Connector can call an event handler for the following events.

OnStart	The `OnStart` event handler is called when Web Connector starts processing an asynchronous action.
OnFinish	The `OnFinish` event handler is called when Web Connector successfully finishes processing an asynchronous action.
OnError	The `OnError` event handler is called when an asynchronous action fails and cannot continue.
OnErrorReport	The `OnErrorReport` event is called when an asynchronous action encounters an error, but the action continues. This event is not available for every connector.
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.

Web 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`, `OnErrorReport`, `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
     OnErrorReport=ErrorEvents
     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`, `OnErrorReport`, `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, below.

For example:

```
[NormalEvents]
LibraryName=TextFileHandler
FilePath=./events.txt

[ErrorEvents]
LibraryName=LuaHandler
LuaScript=./error.lua
```

6. Save and close the configuration file. You must restart Web 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. This parameter is optional but you should set a minimum value, maximum value (with `PerfMonCounterMaxN`), or both.
`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. This parameter is optional but you should set a maximum value, minimum value (with `PerfMonCounterMinN`), or both.
`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

PerfMonCounterName1=\\machine-hostname\Processor(_Total)\% Processor Time
PerfMonCounterMax1=70
```

```
PerfMonAvgOverReadings=5
PerfMonQueryFrequency=10
```

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 *Web 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

Field Name	Description
	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.
`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`.

Appendix B: LuaClientSession Methods

A `LuaClientSession` object provides methods for interacting with a web page that is being processed by the Web Connector. For more information, see Scripted Processing, on page 70.

If you have a `LuaClientSession` object called `session` you can call its methods using the `':'` operator. For example:

```
session:waitForQuiet(250, 10000)
```

Method	Description
attributeValue	Returns the value of an attribute on an element that matches a CSS selector.
clickElement	Clicks an element on a web page.
countElements	Returns the number of elements on a web page that match a CSS selector.
elementText	Returns the inner text of an element on a web page.
executeJavascript	Runs JavaScript on a web page.
snapshotPage	Takes a snapshot of a web page for indexing.
waitForQuiet	Forces the connector to wait until there has been a specified amount of inactivity.

attributeValue

The `attributeValue` method returns the value of an attribute on an element that matches a CSS selector.

Syntax

```
attributeValue( cssSelector, attributeName )
```

Arguments

Argument	Description
`cssSelector`	(string) A CSS selector that identifies the element. If multiple elements match the selector, the first matching element is used.
`attributeName`	(string) The name of the attribute.

Returns

(String). The attribute value. Returns `nil` if the element or attribute was not found.

clickElement

The `clickElement` method clicks an element on a web page.

Syntax

```
clickElement( cssSelector )
```

Arguments

Argument	Description
`cssSelector`	(string) A CSS selector that identifies the element to click. If multiple elements match the selector, the first matching element is clicked.

Returns

(Boolean). Returns true on success, or false if the element was not found.

countElements

The `countElements` method returns the number of elements on a web page that match a CSS selector.

Syntax

```
countElements( cssSelector )
```

Arguments

Argument	Description
`cssSelector`	(string) A CSS selector.

Returns

(Integer). The number of elements.

Example

```
local n = session:countElements("#next-button")
```

elementText

The `elementText` method returns the inner text of an element on a web page.

Syntax

```
elementText( cssSelector )
```

Arguments

Argument	Description
`cssSelector`	(string) A CSS selector that identifies the element. If multiple elements match the selector, the first matching element is used.

Returns

(String). The text. Returns `nil` if the element was not found.

executeJavascript

The `executeJavascript` method runs JavaScript on a web page.

Syntax

```
executeJavascript( scriptSource )
```

Arguments

Argument	Description
`scriptSource`	The JavaScript to run.

Returns

(LuaJsonValue). Returns a `LuaJsonValue` object that contains the result.

Example

```
local ret = session:executeJavascript("document.location.hash")
```

snapshotPage

The `snapshotPage` method takes a snapshot of a web page for indexing.

Syntax

```
snapshotPage( snapshotKey )
```

Arguments

Argument	Description
`snapshotKey`	(string) An identifier that is unique and identifies the snapshot. This is added to the document reference, for example `[snapshot][`*`snapshotKey`*`]http://...`.

Returns

(Boolean). Always returns true.

Example

```
session:snapshotPage("Page" .. tostring(pageNumber))
```

waitForQuiet

The `waitForQuiet` method forces the connector to wait until there has been a specified amount of inactivity (in which no resources are downloaded and there are no changes to the DOM).

Syntax

```
waitForQuiet( quiet, timeout )
```

Arguments

Argument	Description
`quiet`	(integer) The specified amount of "quiet" time, as a number of milliseconds.
`timeout`	(integer) The maximum amount of time to wait, in milliseconds.

Returns

(Boolean). Returns true if the page was quiet for specified amount of time, or false if the timeout was reached.

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.

CSS
Cascading Style Sheets

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 Micro Focus IDOL Web Connector 12.8 Administration Guide

Add your feedback to the email and click **Send**.

If no email client is available, copy the information above to a new message in a web mail client, and send your feedback to swpdl.idoldocsfeedback@microfocus.com.

We appreciate your feedback!