Coveo Platform 7.0

Sitecore Legacy Connector Guide

Notice

Document part number:	PM-120813-EN
Revision:	C
Publication date:	2012-11-23
Publication date:	1/3/2019

Table of Contents

1. Sitecore Legacy Connector

Deprecated

The Coveo legacy connector for Sitecore allows users to index Sitecore CMS content as well as Sitecore Web and Master database content. All the site items, the referenced media library items, and security elements can be indexed.

Note: This section describes the legacy connector for Sitecore. This connector remains available for use in existing instance but it is strongly recommended to rather use the Sitecore 2nd generation connector that offers several advantages).

Supported Sitecore content

- All the site items
- Referenced media library items
- Security elements

Site item metadata indexing

The Coveo Platform can index Sitecore metadata to allow the Coveo administrator to create facets, custom fields, sorting, and group by, based on this metadata (see "About Sitecore Metadata with the Legacy Connector" on page 33 and "Defining a Sitecore Mapping File for the Legacy Connector" on page 29).

Media library item indexing

The Coveo Platform can index the complete media library of Sitecore including files such as images, PDF, and other binary documents (see "Include Media Library" on page 21).

Security

The Coveo Platform can index security permissions from the permission model of Sitecore (see "Configuring a Sitecore Security Provider for the Legacy Connector" on page 13 and "How CES Handles the Sitecore Permission Model with the Legacy Connector" on page 29).

Incremental refresh

The connector periodically queries Sitecore for the latest changes, keeping the index content up-to-date (see "Enabling Incremental Refresh on a Sitecore Database for the Legacy Connector" on page 9).

Multilanguage support for items

The Coveo Platform can index the various language versions of an item, or optionally index the specific language version that is needed (see "Languages" on page 19).

What's Next?

Review the overview of the deployment process for this connector (see "Sitecore Legacy Connector Deployment Overview" on page 1).

2. Sitecore Legacy Connector Deployment Overview

Deprecated

The following procedure outlines the steps required to bring content into the Coveo unified index using the Sitecore connector. The steps indicate the order in which you must perform configuration tasks.

Note: It is strongly recommended to rather use the Sitecore 2nd generation connector or Coveo for Sitecore based on your Sitecore version.

To deploy the Sitecore connector

1. Validate that your environment meets the requirements (see "Sitecore Legacy Connector Requirements" on page 5).
2. On the Sitecore server, install the Coveo Web Service for Sitecore that allows the connector to retrieve information from Sitecore (see "Installing the Coveo Legacy Web Service Plugin for Sitecore" on page 6).
3. On the Coveo server:
   a. Configure the user identity

      The connector needs to know the account to use to crawl the Sitecore content. By default, the connector uses the `extranet\Anonymous` account to retrieve content from Sitecore. Depending on the security configuration, this account might not have access to all Sitecore items.

      It is recommended to configure a CES user identity to use a dedicated Sitecore account that has the `Read` permission to all the Sitecore content. This account does not need to be a Sitecore administrator. It is also recommended, but not mandatory, to specify a user part of the same domain as the crawled source (see "Adding a User Identity" on page 11).

   b. Optionally configure a security provider

      When you want to index security permissions and use the search interface integration within a Sitecore website you need to create a Sitecore security provider (see "Configuring a Sitecore Security Provider for the Legacy Connector" on page 13).

   c. Configure and index the Sitecore source

      A Sitecore source only targets one Sitecore website, therefore it is recommended to create one source for each Sitecore website to index (see "Configuring and Indexing a Sitecore Source for the Legacy Connector" on page 15).

   d. Optionally modify hidden source parameters

      Once your Sitecore connector source is up and running, when you encounter specific issues or you want to achieve specific crawling behavior, consider modifying some hidden source parameters to resolve the issues (see "Modifying Hidden Sitecore Source Parameters for the Legacy Connector" on page 24).

   e. Optionally create a mapping file

When you want to modify how referenced Sitecore field values are resolved, create a mapping file to specify how to map Sitecore metadata to CES fields (see "Defining a Sitecore Mapping File for the Legacy Connector" on page 29 and "About Sitecore Metadata with the Legacy Connector" on page 33).

4. Optionally enable incremental refresh

   When you want to maintain the indexed Sitecore content up-to-date between index refreshes, while the Sitecore connector supports incremental refreshes, you must configure the Sitecore database to allow incremental refresh (see "Enabling Incremental Refresh on a Sitecore Database for the Legacy Connector" on page 9).

5. Optionally integrate the Coveo search interfaces in a Sitecore website:

   When Coveo search interfaces are integrated in a Sitecore website, end-users can use all Coveo search functionalities directly in the website (see "Integrating the Coveo .NET Search Interface in a Sitecore Website" on page 35).

3. Security Update for Sitecore and Sitecore Legacy Connectors

CES 7.0.6684– (May 2014)

This topic describes how to resolve a security vulnerability in our Sitecore Connector plugin that could allow code to access Sitecore content that would otherwise be inaccessible.

CES 7.0.6767+ (June 2014) The monthly release includes the fix for this security vulnerability.

Prerequisites to apply the security update

Before you can apply this security update, if it is not already the case, your Coveo server must run either of the following CES 6 or 7 monthly releases:

- CES 7.0.6684 May 2014
  - 64-bit version download
  - 32-bit version download
- CES 6.5.4898 Dec 2013
  - 64-bit version download
  - 32-bit version download

For help installing updates, contact Coveo Support.

To resolve the vulnerability

1. For CES 6.5 only, download and install the applicable security vulnerability hotfix for CES 6.5.4898 (Dec 2013):
   - 64-bit version download
   - 32-bit version download
2. Download the appropriate Sitecore package (ZIP file):
   - For Sitecore 7.x and 6.x:
     - Using the Sitecore Connector:
       - 32-bit or 64-bit application pool for CES 7.0 or CES 6.5
     - Using the Sitecore Legacy Connector:
       - 32-bit application pool for CES 7.0 or CES 6.5
       - 64-bit application pool for CES 7.0 or CES 6.5
   - For Sitecore 5.3, using the Sitecore Legacy Connector:

   - 32-bit application pool CES 7.0 or CES 6.5
   - 64-bit application pool CES 7.0 or CES 6.5

3. Install the appropriate downloaded Sitecore package on your Sitecore server:
   - For the Sitecore Connector, install the Coveo web service plugin in Sitecore.
   - For the Sitecore Legacy Connector, install the Coveo Legacy web service plugin for Sitecore.

4. Sitecore Legacy Connector Requirements

Deprecated

Your environment must meet the following requirements to be able to use the Sitecore connector:

- Coveo license for the Sitecore Legacy connector

  Your Coveo license must include support for the Sitecore Legacy connector to be able to use this connector.

- Sitecore

  The Sitecore Legacy connector supports Sitecore version 5.3, and 6.0 to 6.5.

What's Next?

Install the Coveo web service on your Sitecore server (see "Installing the Coveo Legacy Web Service Plugin for Sitecore" on page 6).

5. Installing the Coveo Legacy Web Service Plugin for Sitecore

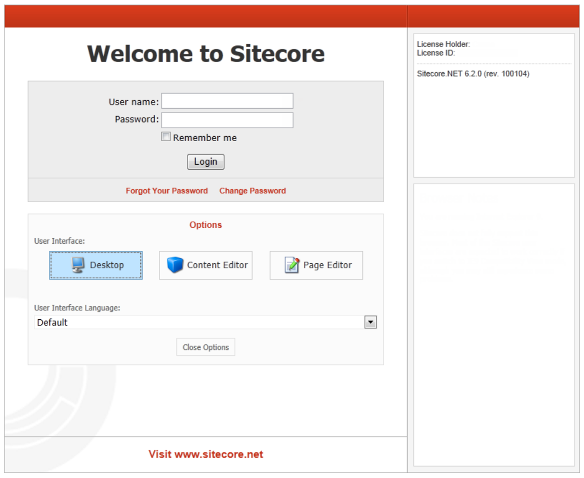
Deprecated

The Coveo Sitecore connector comes with a web service that you must upload and install on your Sitecore server. The Coveo connector needs to use this web service to securely and efficiently crawl the Sitecore content.

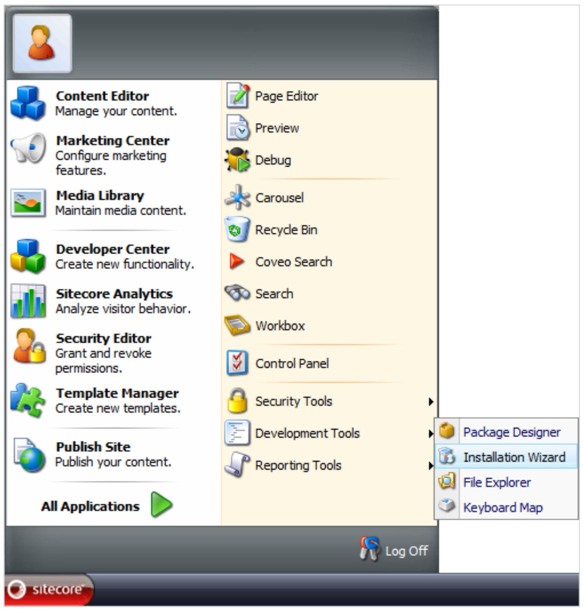
Important: When you upgrade Coveo Enterprise Search (CES) on the Coveo Master server, you must perform this procedure to also upgrade the Coveo web service on your Sitecore server. The Sitecore connector will not work when an incompatible version of the plugin is installed on the Sitecore server.

To install the Coveo web service plugin on your Sitecore server

1. Using a Sitecore administrative account that has the necessary rights to install a plugin:
   a. Access the **Welcome to Sitecore** login screen.
   b. Click **Options** to make the **User interface** buttons appear.
   c. Select the **Desktop** user interface.
   d. In the **User name** and **Password** boxes, enter the administrative account credentials.
   e. Click **Login**.

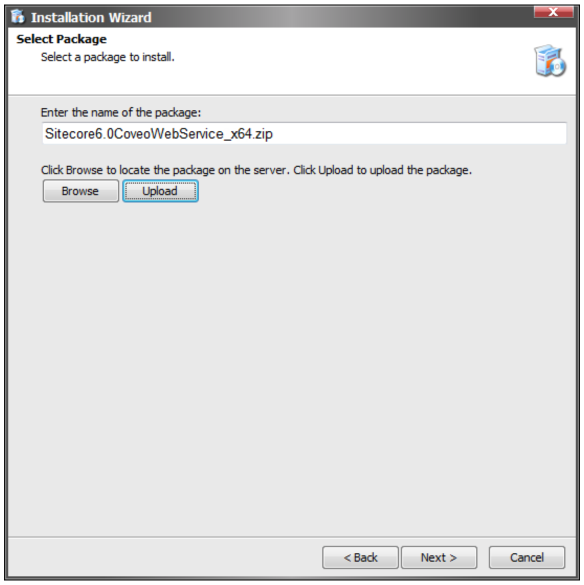


2. On the Sitecore Desktop, select **Sitecore** > **Development Tools** > **Installation Wizard**.



3. In the **Installation Wizard** dialog box:

    a. In the **Select Package** screen:



Select Package

Select a package to install.

Enter the name of the package:

Sitecore6.0CoveoWebService_x64.zip

Click Browse to locate the package on the server. Click Upload to upload the package.

i. Click **Upload**, and then browse the Coveo Master server to select the appropriate version of the Coveo Web Service plugin package distributed with CES in the `[CES_Install_Path]\Bin\` folder:

- `Sitecore6.0CoveoWebService_x64.zip` for Sitecore 6.x on a 64-bit server
- `Sitecore6.0CoveoWebService_x86.zip` for Sitecore 6.x on a 32-bit server

- `Sitecore5.3CoveoWebService_x64.zip` for Sitecore 5.3 on a 64-bit server
- `Sitecore5.3CoveoWebService_x86.zip` for Sitecore 5.3 on a 32-bit servers

Note: Select `Sitecore6.0CoveoWebService2_x[nn].zip` for a Sitecore 6.x server when using the new Sitecore connector.

Important: When you are updating the web service, in the browsing dialog box, ensure to select the **Overwrite existing file** check box.

ii. Click **Next**.

b. In the **Ready to Install** screen, click **Install**.

Note: If an `Access Denied error` is displayed for the `bin_install` folder while installing the package, verify the security permissions for `ASP.NET` in the bin folder of Sitecore or install the files manually.

4. Once the installation is completed, using a text editor, open the `web.config` file at the root of the Sitecore website, and then ensure that the following line is present inside the `<sitecore><settings>` tags:

```
<setting name="VersionFilePath" value="/sitecore/shell/sitecore.version.xml"/>
```

5. Restart both the Sitecore client and server.

What's Next?

When you want the connector to use incremental refresh to maintain the index up-to-date, configure Sitecore to allow it (see "Enabling Incremental Refresh on a Sitecore Database for the Legacy Connector" on page 9).

6. Enabling Incremental Refresh on a Sitecore Database for the Legacy Connector

Deprecated

The legacy connector for Sitecore supports incremental refresh to allow your Sitecore source to remain up-to-date as Sitecore content changes.

You must define the Sitecore `Engines.HistoryEngine.Storage` class in your Sitecore `web.config` file for the database that the connector source targets to make incremental refresh work.

To enable incremental refresh on a Sitecore database

1. Using and administrator account, connect to the Sitecore server.
2. Using a text editor:
   a. Open the Sitecore `web.config` file.

      Note: It is recommended to make a backup of the `web.config` file before editing it.

   b. In the `<database>` section for the desired publishing target, add the `Engines.HistoryEngine.Storage` as shown in the following examples.

      Example: When your Sitecore source targets the site named website and the database serving this site is named `web`:

      - For Sitecore 6.0+

```
<database id="web" singleInstance="true" type="Sitecore.Data.Database,
…
  <Engines.HistoryEngine.Storage>
    <obj type="Sitecore.Data.$(database).$(database)HistoryStorage, Sitecore.Kernel">
      <param connectionStringName="$(id)"/>
      <EntryLifeTime>30.00:00:00</EntryLifeTime>
    </obj>
  </Engines.HistoryEngine.Storage>
…
</database>
```

      - For Sitecore 5.3

```
<database id="web" singleInstance="true" type="Sitecore.Data.Database,
 …
  <Engines.HistoryEngine.Storage>
    <obj type="Sitecore.Data.$(database).$(database)HistoryStorage,
Sitecore.$(database)">
    <param desc="connection" ref="connections/$(id)"></param>
    <EntryLifeTime>30.00:00:00</EntryLifeTime>
    </obj>
  </Engines.HistoryEngine.Storage>
…
</database>
```

Note: Every modification to the database of the site is considered when indexing on an incremental refresh run.
You might experience a slow incremental refresh operation in some special design of your Sitecore website.

Example: When your website counts page visits by modifying a metadata field, this marks the item as `modified`. The item is re-indexed on the next incremental refresh run. When CES crawls a Sitecore site, it might load every page to retrieve its HTML content (depending of your source options). In this case, the page visited counter is updated for each single page, causing each page to be marked as `modified` and re-indexed on the next incremental refresh run.

If you must rely on metadata item fields that have to be modified on-the-fly, you can work around that limitation by checking the **http request user agent** before updating the database. When the Sitecore connector loads a page from the website, it uses the Coveo Sitecore crawler user agent.

What's Next?

Create a Sitecore user identity (see "Adding a User Identity" on page 11) and a Sitecore security provider (see "Configuring a Sitecore Security Provider for the Legacy Connector" on page 13).

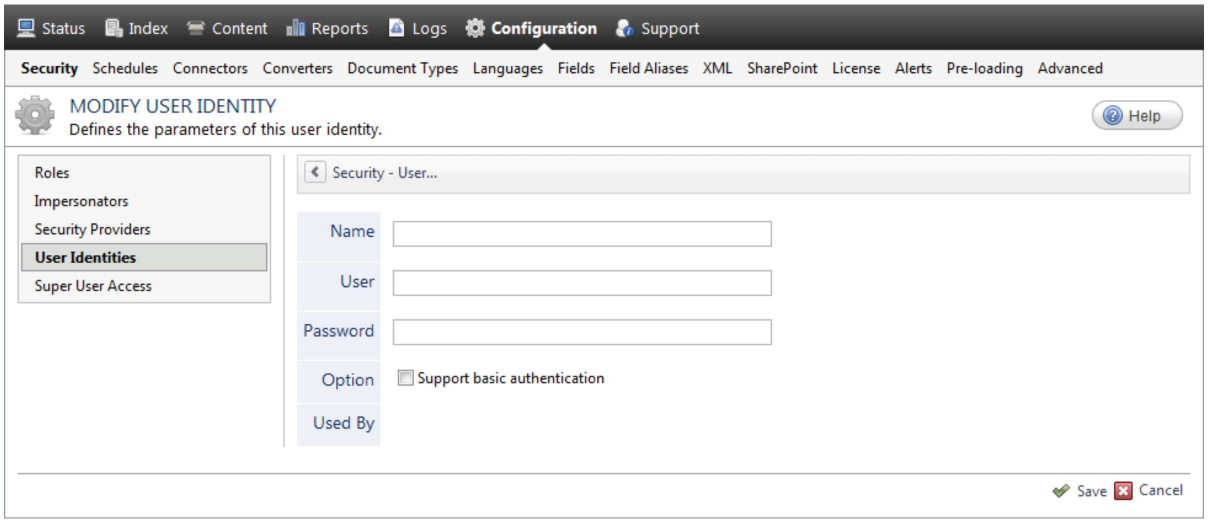
7. Adding a User Identity

A user identity is a set of credentials for a given repository or system that you enter once in CES and can then associate with one or more sources or security providers.

A user identity typically holds the credentials of an account that has read access to all the repository items that you want to index. It is a best practice to create an account to be used exclusively by the Coveo processes and for which the password does not change. If the password of this account changes in the repository, you must also change it in the CES user identity.

To add a user identity

1. On the Coveo server, access the Administration Tool.
2. In the Administration Tool, select **Configuration** > **Security**.
3. In the navigation panel on the left, click **User Identities**.
4. In the **User Identities** page, click **Add**.
5. In the **Modify User Identity** page:



   a. In the **Name** box, enter a name of your choice to describe the account that you selected or created in the repository to allow CES to access the repository.

      Note: This name appears only in the Coveo Administration Tool, in the **Authentication** or **User Identity** drop-down lists, when you respectively define a source or a security provider.

   b. In the **User** box, enter the username for the account that you selected or created to crawl the repository content that you want to index.

   c. In the **Password** box, enter the password for the account.

   d. In the **Options** section, the **Support basic authentication** check box is deprecated and not applicable for most types of repositories. You should select it only when you need to allow CES to send the username and password as unencrypted text.

e. Click **Save**.

Important: When you use Firefox to access the Administration Tool and it proposes to remember the password for the user identity that you just created, select to never remember the password for this site to prevent issues with automatic filling of username and password fields within the Coveo Administration Tool.

8. Configuring a Sitecore Security Provider for the Legacy Connector

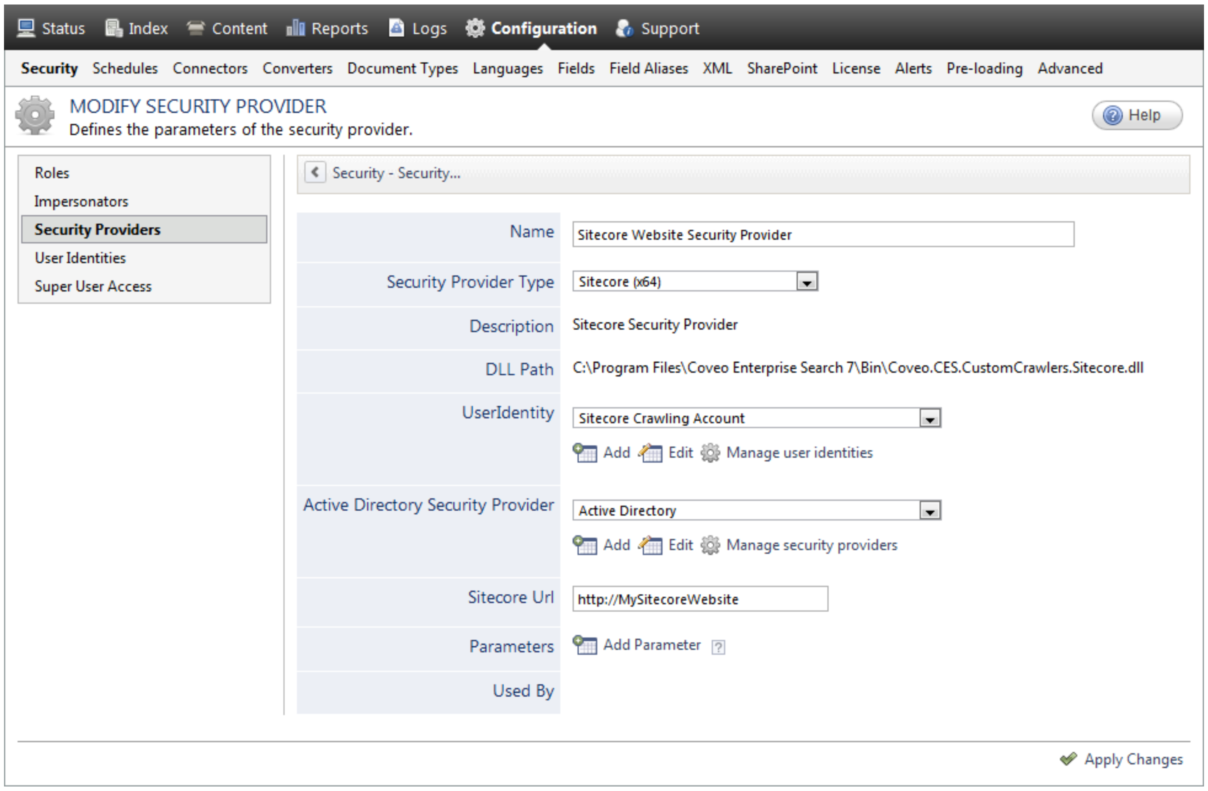
Deprecated

A security provider is required to index Sitecore security permissions and use the search interface integration within a Sitecore website.

Note: You can get familiar with how Coveo components deal with permissions on documents both at indexing and query time.

To configure a Sitecore security provider

1. On the Coveo server, access the Administration Tool.
2. Select **Configuration** > **Security**.
3. In the navigation panel on the left, click **Security Providers**.
4. In the **Security Providers** page, click **Add** to create a new security provider.
5. In the **Modify Security Provider** page:



MODIFY SECURITY PROVIDER
Defines the parameters of the security provider.

Name	Sitecore Website Security Provider
Security Provider Type	Sitecore (x64)
Description	Sitecore Security Provider
DLL Path	C:\Program Files\Coveo Enterprise Search 7\Bin\Coveo.CES.CustomCrawlers.Sitecore.dll
UserIdentity	Sitecore Crawling Account
Active Directory Security Provider	Active Directory
Sitecore Url	http://MySitecoreWebsite
Parameters	
Used By	

a. In the **Name** box, enter a significant name to identify the security provider.

Example: `Sitecore Website Security Provider`

b. In the **Security Provider Type** drop-down list, select **Sitecore (Legacy)**:

Note: CES 7.0.4863 to 7.0.5785 (July 2012 to August 2013) The Sitecore Legacy connector appeared in the list as **Sitecore** while the new Sitecore connector appeared as **Sitecore2**.

c. In the **User Identity** drop-down list, select the user identity that you created for this source or, select **(none)** to use the default `extranet\Anonymous` account that has access to the Roles and Users definitions.

d. In the **Active Directory Security Provider** drop-down list:

   i. Select the appropriate security provider that this security provider uses to resolve and expand the groups.

   Coveo Enterprise Search (CES) comes with an Active Directory security provider that you can configure to connect to the default domain. When you environment contains more than one domain, you can select an Active Directory security provider that you created for other untrusted domains.

   ii. When an appropriate security provider is missing, click **Add**, **Edit**, or **Manage security providers** respectively to create, modify, or manage security providers.

e. In the **Sitecore URL** box, enter the URL of your Sitecore website.

Example: `http://MySitecoreWebsite`

Note: Entering the wrong URL for the **Sitecore URL** box, like the one of your Coveo server rather than the one of your Sitecore server, can cause repetitive unanswered calls to this URL and make this server unresponsive.

f. In the **Parameters** section, in rare cases the Coveo Support could instruct you to click **Add Parameters** to specify other security provider parameter names and values that could help to troubleshoot security provider issues.

g. Leave the **Allow Complex Identities** option cleared as it does not apply to this type of security provider.

h. Click **Apply Changes**.

What's Next?

Create your Sitecore source ("Configuring and Indexing a Sitecore Source for the Legacy Connector" on page 15).

9. Configuring and Indexing a Sitecore Source for the Legacy Connector

Deprecated

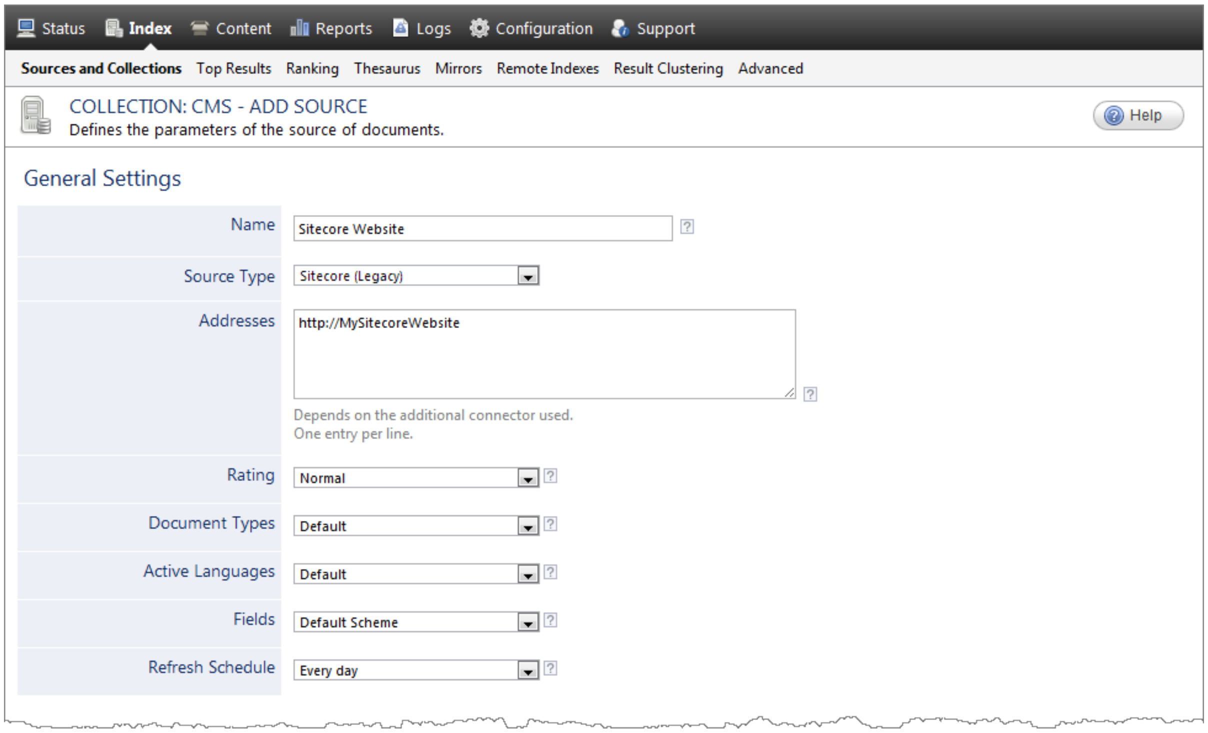
A Sitecore source only targets one Sitecore website. It is recommended to configure one source for each Sitecore website to index.

To configure and index a Sitecore source

1. On the Coveo server, access the Administration Tool.
2. Select **Index** > **Sources and Collections**.
3. In the **Collections** section:
    a. Select an existing collection in which you want to add a new source.

       OR

    b. Click **Add** to create a new collection.
4. In the **Sources** section, click **Add**.
5. In the **General Settings** section of the **Add Source** page:



COLLECTION: CMS - ADD SOURCE
Defines the parameters of the source of documents.

General Settings

Name	Sitecore Website
Source Type	Sitecore (Legacy)
Addresses	http://MySitecoreWebsite Depends on the additional connector used. One entry per line.
Rating	Normal
Document Types	Default
Active Languages	Default
Fields	Default Scheme
Refresh Schedule	Every day

a. Enter the appropriate value for the following required parameters:

Name

Enter a descriptive name of your choice for the connector source.

Example: `Sitecore Website (English)`

Source Type

CES 7.0.7814+ (August 2015) The connector used by this source. In this case, select **Sitecore (deprecated)**.

Notes:

- If you do not see **Sitecore (deprecated)** in the **Source Type** list, your environment does not meet the requirements (see "Sitecore Legacy Connector Requirements" on page 5).
- CES 7.0.5935+ (September 2013) Select **Sitecore** when you want to use the second generation Sitecore connector.
- CES 7.0.7711– (June 2015) The Sitecore Legacy connector appears in the list as **Sitecore (Legacy)**.
- CES 7.0.5785– (August 2013) The Sitecore Legacy connector appeared in the list as **Sitecore** while the second generation Sitecore connector appeared as **Sitecore2**.

Addresses

The base address of the Sitecore installation. Enter one address in the following form:

```
http://SitecoreWebsite
```

The connector supports both http and https.

Important: While the value in the **Addresses** box points to your Sitecore server, the **Target Site** box by default specifies to index the website site hosted in this server. In Sitecore, website is the default name for a site. When the site you want to index has a different name, like when your server hosts more than one site, you must specify the site name in the **Target Site** box.

You can also use the **Content Start Path** box to restrict indexing to one or more branch of the content tree.

Tip: Once you indexed your Sitecore content, If you obtain clickable URIs containing `http` twice such as `http://http/www.MyServer.com`, in the `site` definition of your Sitecore `web.config` file, ensure `hostName` does not contain `http://`. If you want to explicitly specify the protocol, use the `scheme` parameter (ex: `<site name="WWWPortal" hostName="www.mysite.com" scheme="http" rootPath="/sitecore/content/Home" startItem="/Portal" contentStartItem="/Portal" />`).

b. Review the value for the following parameters that often do not need to be modified:

Rating

Change this value only when you want to globally change the rating associated with all items in this source relative to the rating to other sources.

Example: If this source was for a legacy website, you may want to set this parameter to **Low**, so that in the search interface, results from this source appear later in the list compared to those from other sources.

Document Types

If you defined custom document type sets, ensure to select the most appropriate for this source.

Active Languages

If you defined custom active language sets, ensure to select the most appropriate for this source.

Fields

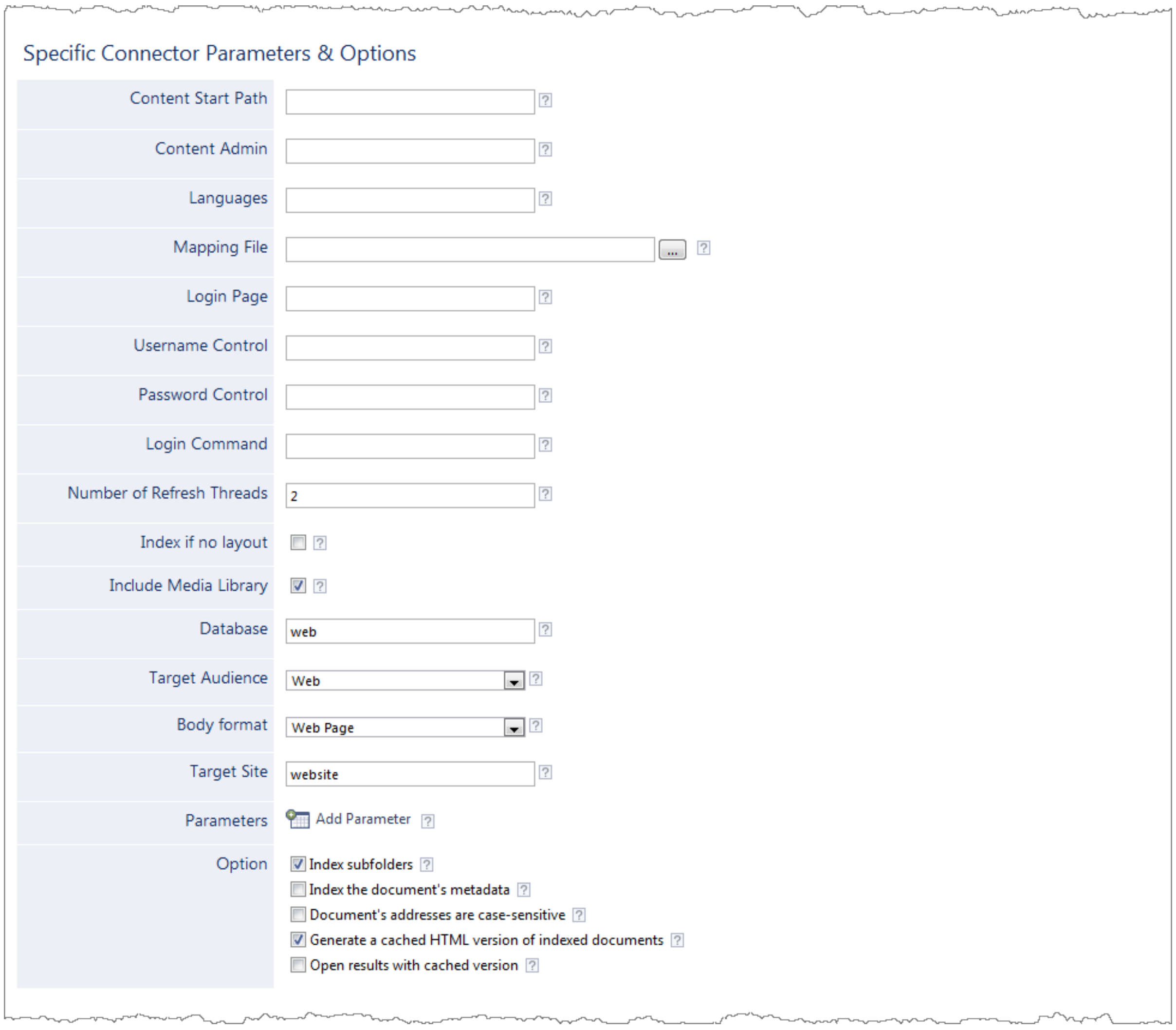
If you defined custom field sets, ensure to select the most appropriate for this source.

Refresh Schedule

Time interval at which the index is automatically refreshed to keep the index content up-to-date. By default, the **Every day** option instructs CES to refresh the source every day at 12 AM.

Note: You can create new or modify existing source refresh schedules.

6. In the **Specific Connector Parameters & Options** section of the **Add Source** page, review if you need to change the parameter default values:



Specific Connector Parameters & Options

Content Start Path	
Content Admin	
Languages	
Mapping File	
Login Page	
Username Control	
Password Control	
Login Command	
Number of Refresh Threads	2
Index if no layout	☐
Include Media Library	✓
Database	web
Target Audience	Web
Body format	Web Page
Target Site	website
Parameters	Add Parameter
Option	✓ Index subfolders ☐ Index the document's metadata ☐ Document's addresses are case-sensitive ✓ Generate a cached HTML version of indexed documents ☐ Open results with cached version

a. Enter the appropriate value for the following optional parameters:

Content Start Path

The starting point of indexing in the Sitecore content tree. When left blank, the default value corresponds to the default root path of the **Target Site**. You can specify one or more starting path by separating multiple root nodes with a semicolon (`;`).

Example: `/sitecore/content/home/MyNewRootNode;/sitecore/content/Resources`

Tip: You can determine the default root path in the Sitecore `web.config` file, by concatenating the `rootPath` and `startItem` attributes for a target site.

Content Admin

The Windows user that can see all indexed documents. By default, it is impossible to see the indexed content of a document when a source is using a security provider to index the permissions. Use this parameter when the source uses a security provider and you need to see the indexed document content in the Index Browser.

Enter the user name in the following form: `DomainName\UserName`

Languages

Indicates the indexed languages. You can specify the languages to crawl by entering one or more language codes separated by a semicolon (`;`). Enter the `*` wildcard character to index all languages. A document is indexed for each language. By default, when the box is empty, a single document is indexed using the default language of the **Target Site**. If there is no language set on the site, English is used.

Example: `en;fr-CA`

Note: Items of the media library are always indexed with the site default language.

Mapping File

Enter the path of a valid XML mapping file that defines how the connector handles metadata.

Configuring a complete mapping file is a key element to leverage the Sitecore metadata to produce a feature-rich search interface (see "Defining a Sitecore Mapping File for the Legacy Connector" on page 29).

b. When your Sitecore website contains secured sections, use the following forms authentication parameters to allow the connector to authenticate itself and gain access to secured pages:

Note: When using form authentication, the **Body Format** source option must be set to **Web Page** and **Generate a cached HTML version of indexed documents** must be selected.

Login Page

URL of the page where users log on for forms authentication.

Username Control

ID of the control where users enter their username for forms authentication.

Tip: You can get the ID by inspecting the corresponding `input` HTML tag from the source of the web page using your browser inspection features.

Some websites use dynamic content (AJAX) in which case the page source might not be enough to retrieve the control ID. You can then use an external web debugger such as Fiddler to find what are the values passed to the server when the login command is invoked.

Example: On Internet Explorer, select **View** > **Source**, locate the corresponding `input` tag, and extract the `id` (`ctl00_ctlContentPlaceHolder_ctl00_ctlLogonControl_ctlPanelBar_txtUserName` in the sample code below)

```
<input name="ctl00$ctlContentPlaceHolder$ctl00$ctlLogonControl$ctlPanelBar$txtUserName"
type="text" id="ctl00_ctlContentPlaceHolder_ctl00_ctlLogonControl_ctlPanelBar_
txtUserName" class="FormInputText" Focus="True" style="width:" />
```

Password Control

ID of the control where users to enter their password for forms authentication.

Login Command

Login command sent by the forms authentication page.

c. Revise the default value of the following parameters:

Number of Refresh Threads

Determines the number of simultaneous downloads handled by the connector for this source. The default value is `2`.

Index if no Layout

By default, items with no layout cannot be directly found from a web browser and are therefore not indexed. Select the check box to index items that have no defined layout. This is useful to index the content of the blog post module.

Tip: For blog posts items, you can change the clickable URL using a mapping file (see "Defining a Sitecore Mapping File for the Legacy Connector" on page 29).

Example: When using the Sitecore blog module in the `Printers` sample site, the following mapping file can index blog posts when the **Index No Layout** option is selected on the source.

```
<?xml version="1.0" encoding="utf-8" ?>
<Sitecore>
  <CommonMappings>
    <Fields>
      <Title>%[_CESSCDisplayName]</Title>
    </Fields>
  </CommonMappings>
  <Mapping template="{5CF2ED9B-6C32-4FA3-9549-2AB77085B131}"> <!--UserBlog-->
    <Fields>
      <ClickableUri> %[_CESSCServerBaseUrl]/Company/Blogs.aspx?blog=%[Blog
Title]</ClickableUri>
      <PrintableUri> %[_CESSCServerBaseUrl]/Company/Blogs.aspx?blog=%[Blog
Title]</PrintableUri>
    </Fields>
  </Mapping>
  <Mapping template="{1FBDD65D-5029-46F1-8D75-AF3E68810B25}"> <!--Article-->
    <Fields>
      <ClickableUri>%[_CESSCServerBaseUrl]/Company/Blogs.aspx?post=%[Title]&amp;blog=%[_
CESSCParentID.Blog Title]</ClickableUri>
      <PrintableUri>%[_CESSCServerBaseUrl]/Company/Blogs.aspx?post=%[Title]&amp;blog=%[_
CESSCParentID.Blog Title]</PrintableUri>
    </Fields>
  </Mapping>
  <Mapping template="{FB71F255-31D5-417A-BD5C-12D458EB8FDB}"> <!--Comment-->
    <Fields>
      <ClickableUri>%[_CESSCServerBaseUrl]/Company/Blogs.aspx?post=%[_
CESSCParentID.Title]&amp;blog=%[_CESSCParentID._CESSCParentID.Blog Title]</ClickableUri>
      <PrintableUri>%[_CESSCServerBaseUrl]/Company/Blogs.aspx?post=%[_
CESSCParentID.Title]&amp;blog=%[_CESSCParentID._CESSCParentID.Blog Title]</PrintableUri>
    </Fields>
  </Mapping>
</Sitecore>
```

Include Media Library

By default, this check box is selected to index all the content of the media library. This has the same effect as adding the `/Sitecore/content/media library` to the **Content Start Path** value.

Note: When media items are referenced from content items that are indexed, these media items are also indexed even when this check box is cleared.

Database

The name of the Sitecore database to index. You can also enter `master` to index the non-published content of the target site. When left blank, the default value corresponds to the database defined for the **Target Site**.

Note: When you specify a value other than the default and use a security provider, you must set the security provider database parameter to the same value (see "Configuring a Sitecore Security Provider for the Legacy Connector" on page 13).

Target Audience

Indicates to the connector what the targeted audience of the source is. This option affects how the items are opened when a user clicks a search result:

- **Web**: Opens the results as a standard Web page. Default value.
- **Content Editors**: Opens the results for edition in the Sitecore Content Editor.

Body Format

Specifies how the HTML cached version of an indexed document is saved.

- **Web Page**: Sends the HTML version of an item as rendered by Sitecore. This is the default value that produces a nice Quick View.

  It is however important to set the body field in the mapping file. Otherwise, the navigation and other peripheral elements of the pages are indexed and become searchable (see "Defining a Sitecore Mapping File for the Legacy Connector" on page 29).

- **Metadata**: Only sends Sitecore metadata and values. The Quick View presents an unformatted list of all Sitecore metadata and values.

  This option is useful for an administrator to review all the metadata gathered by the connector and help configuring the mapping file. You can use this option in conjunction with **Target Audience** set to **Content Editors**.

Target Site

Specifies the targeted Sitecore site to index. The default value is `website`. When the Sitecore website does not use the default name (`website`), you must use this parameter and provide the appropriate name. You can get the name of the site from the Sitecore `web.config` file.

Example: The following excerpt shows a Sitecore `web.config` file defining five websites. All the websites hosted in a single Sitecore installation are defined under the `<site>` node and the string to enter in the `TargetSite` parameter is the one of the name attribute.

```
<sites>
 ...
 <site name="danish" hostName="da.printers" language="da-DK" virtualFolder="/"
 <site name="german" hostName="de.printers" language="de-DE" virtualFolder="/"
 <site name="english" hostName="en.printers" language="en" virtualFolder="/"
 <site name="british" hostName="gb.printers" language="en-GB" virtualFolder="/"
 <site name="website" virtualFolder="/" physicalFolder="/"
  ...
</sites>
```

d. Click **Add Parameter** when you want to show advanced source parameters (see "Modifying Hidden Sitecore Source Parameters for the Legacy Connector" on page 24).

e. The **Option** check boxes generally do not need to be changed.

Index Subfolders

Keep this check box selected (recommended). By doing so, all subfolders from the specified starting address are indexed.

Index the document's metadata

When selected, CES indexes all the document metadata, even metadata that are not associated with a field. The orphan metadata are added to the body of the document so that they can be searched using free text queries.

When cleared (default), only the values of system and custom fields that have the **Free Text Queries** attribute selected will be searchable without using a field query.

Example: A document has two metadata:

- `LastEditedBy` containing the value `Hector Smith`
- `Department` containing the value `RH`

In CES, the custom field `CorpDepartment` is bound to the metadata `Department` and its **Free Text Queries** attribute is selected.

When the **Index the document's metadata** option is cleared, searching for `RH` returns the document because a field is indexing this value. Searching for `hector` does not return the document because no field is indexing this value.

When the **Index the document's metadata** option is selected, searching for `hector` also returns the document because CES indexed orphan metadata.

Document's addresses are case-sensitive

Leave this check box cleared. This parameter needs to be checked only in rare cases for systems in which distinct documents may have the same name but different casing.

Generate a cached HTML version of indexed documents

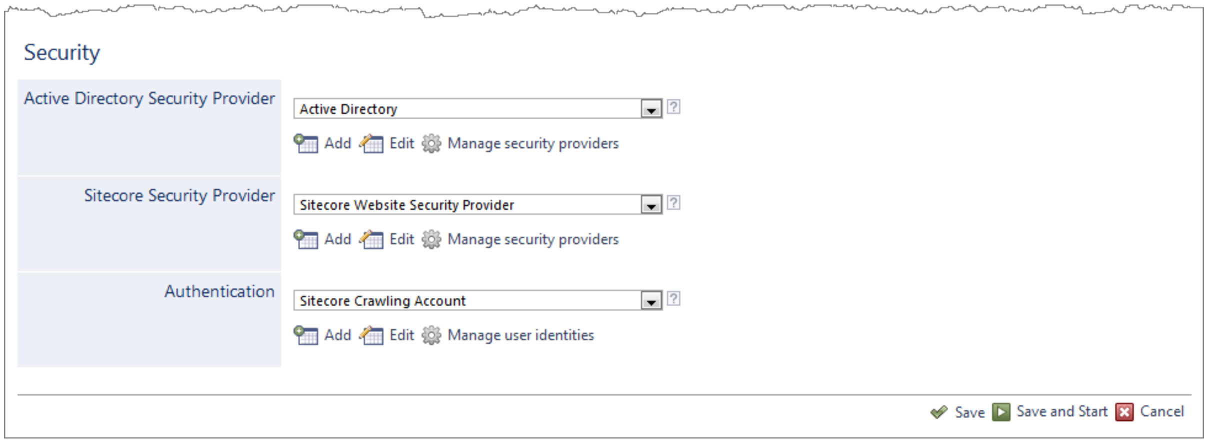
When you select this check box (recommended), at indexing time, CES creates HTML versions of indexed documents. In the search interfaces, users can then more rapidly review the content by clicking the **Quick View** link rather than opening the original document with the original application. Consider clearing this check box only if you do not want to use Quick View links or save resources when building the source.

Open results with cached version

Leave this check box cleared (recommended) so that in the search interfaces, the main search result link opens the original document with the original application. Consider selecting this check box only when you do not want users to be able to open the original document but only see the HTML version of the document as a **Quick View**. In this case, you must also select **Generate a cached HTML version of indexed documents**.

f. In the **Authentication** drop-down list, when you created a Sitecore user identity for this source, select it.

7. In the **Security** section of the **Add Source** page:



Security

Active Directory Security Provider	Active Directory
Sitecore Security Provider	Sitecore Website Security Provider
Authentication	Sitecore Crawling Account

a. In the **Active Directory Security Provider** drop-down list, select **Active Directory** or a custom Active Directory security provider that you created for a specific domain.

b. In the **Sitecore Security Provider** drop-down list, select the security provider that you created for this source (see "Configuring a Sitecore Security Provider for the Legacy Connector" on page 13).

c. In the **Authentication** drop-down list, select the user identity that you created for this Sitecore source.

8. Click **Save** to save the source configuration.

9. Before indexing the source, consider the following optional steps:

   a. Consider showing and modifying advanced source parameters (see "Modifying Hidden Sitecore Source Parameters for the Legacy Connector" on page 24).

   b. Consider using a custom mapping file (see "Defining a Sitecore Mapping File for the Legacy Connector" on page 29).

10. On the button bar, click **Rebuild** to start indexing the source.

11. Validate that the source building process is executed without errors:

    - In the navigation panel on the left, click **Status**, and then validate that the indexing proceeds without errors.

      OR

    - Open the CES Console to monitor the source building activities.

What's Next?

When incremental refresh is enabled (see "Enabling Incremental Refresh on a Sitecore Database for the Legacy Connector" on page 9), set an incremental refresh schedule for your source.

Optionally integrate the Coveo search interface in your Sitecore website (see "Integrating the Coveo .NET Search Interface in a Sitecore Website" on page 35).

9.1 Modifying Hidden Sitecore Source Parameters for the Legacy Connector

Deprecated

The **Add Source** and **Source: ... General** pages of the Administration Tool present the parameters with which you can configure the Sitecore connector. More advanced and more rarely used parameters are hidden. You can choose to make one or more of these parameters appear in the **Add Source** and **Source: ... General** pages of the Administration Tool so that you can change their default value.

The following list describes the available hidden parameters that can be added on a Sitecore source as custom parameters:

WebServiceTimeout (Integer)

Modifies the time the connector waits for a content web service request to complete. Any value in seconds. The default is `100` seconds.

IncrementalRefreshDelay (Integer)

Modifies the delay between incremental refresh runs. Any value in seconds. The default is `300` seconds.

AdminThreshold (Integer)

Specifies the maximum number of users a security group/role can contain. To handle larger groups, increase this number and consider increasing the `WebServiceTimeout` parameter. The default is `250` users.

ADDomains (String)

Specifies to the connector what Sitecore security domains come from the Active Directory module. A list of domain names separated by semicolons (`;`).

ErrorDetails (String)

Changes the error messages verbosity level. Possible values are `Normal` and `Detailed`. The default is `Normal`.

UseCache (Boolean)

Specifies to the connector to use its caching capability to improve crawling performances. The default is `True`.

Note: It is not recommended to set this parameter to `False`. However, caching can be disabled if you are experiencing serious memory problems on your Sitecore installation while crawling.

LoopDelayRefresh, LoopDelayLive, LoopDelayItemsBatch (Integer)

Modifies the time a refresh, incremental refresh, or batched item thread waits between calls to retrieve Sitecore items. These parameters are useful to slow down the crawling when such an operation puts too much load on the Sitecore server. Any value in milliseconds. The default is `0` mS.

UseItemsBatch (Boolean)

Specifies to the connector to make multiple calls to get the list of all Sitecore items to index. The default is `True`.

ItemsBatchSize (Integer)

Specifies the maximum number of items to get at once when the `UseItemsBatch` parameter is set. The default is `300`.

IncrementalRefreshIncludeSecurity (Boolean)

Specifies to the connector to handle the security changes during incremental refresh. Enabling this option slows down the incremental refresh process because of the hierarchical Sitecore security model. The default is `False`.

SkipUnhandledErrors (Boolean)

When enabled and the connector encounters an unhandled error because of a communication error, invalid XML, or other fatal errors, it will skip over them rather than stop. The default is `False`.

IndexStandardFields (Boolean)

Specifies to the connector to index item fields inherited from the *Standard* template. These fields will be available in the metadata of an indexed document in CES. The default is `False`.

UseNewUriFormat (Boolean)

Whether a unique indexed URI is guaranteed for a document. This parameter must be set to `True` when the `ItemPath` property is not unique. In such case, two items with the same name created under the same Sitecore path will have the same indexed URI. When this parameter is enabled, the `ItemID` is appended at the end of the indexed URI to make sure it is unique for all Sitecore documents. The default value is `False`.

SystemMetaPrefix (String)

The prefix for the indexed metadata. The default value is `_CESSC`.

Example: With the default value, when a Sitecore metadata is named `MyField`, then its indexed field counterpart is named `_CESSCMyField`.s

Use the following procedure to modify the above hidden source parameters.

To modify hidden Sitecore source parameters

1. Refer to "Adding an Explicit Connector Parameter" on page 26 to add one or more Sitecore hidden source parameters.
2. For a new Sitecore source, access the **Add Source** page of the Administration Tool to modify the value of the newly added advanced parameter:
   a. Select **Index** > **Sources and Collections**.
   b. Under **Collections**, select the collection in which you want to add the source.
   c. Under **Sources**, click **Add**.
   d. In the **Add Source** page, edit the newly added advanced parameter value.
3. For an existing Sitecore source, access the **Source: ... General** page of the Administration Tool to modify the value of the newly added advanced parameter:
   a. Select **Index** > **Sources and Collections**.
   b. Under **Collections**, select the collection containing the source you want to modify.
   c. Under **Sources**, click the existing Sitecore source in which you want to modify the newly added advanced parameter.
   d. In the **Source: ... General** page, edit the newly added advanced parameter value.

9.2 Adding an Explicit Connector Parameter

Connector parameters applying to all sources indexed using this connector are called explicit parameters.

When you create or configure a source, the Coveo Enterprise Search (CES) 7.0 Administration Tool presents parameters with which you can configure the connector for most setups. For many connectors, more advanced and more rarely used parameters also exist but are hidden by default. CES then uses the default value associated with each of these hidden parameters.

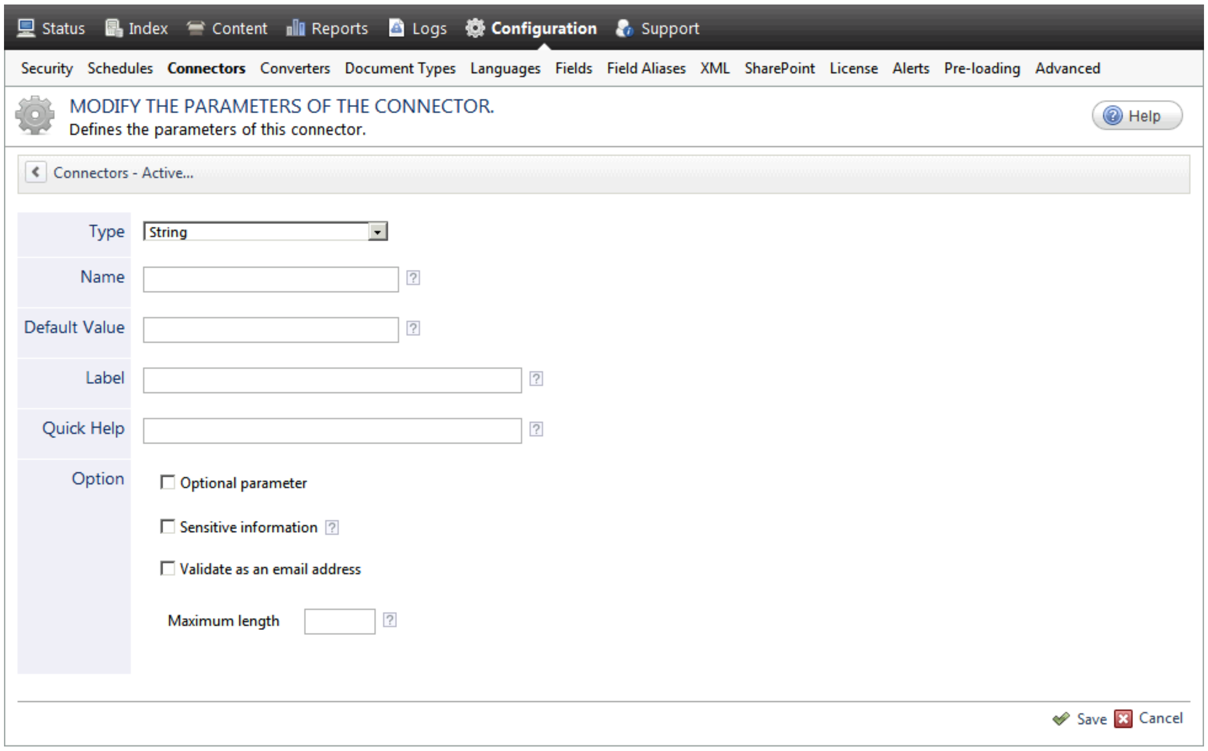
You can however choose to make one or more of these parameters appear in the **Add Source** and **Source: ... General** pages of the Administration Tool so that you can change their default value.

To add an explicit connector parameter

1. On the Coveo server, access the Administration Tool.
2. Select **Configuration** > **Connectors**.
3. In the list on the **Connectors** page, select the connector for which you want to show advanced hidden parameters.
4. In the **Parameters** section of the selected connector page, click **Add Parameter** for each hidden parameter that you want to modify.

   Note: The **Add Parameter** button is present only when hidden parameters are available for the selected connector.

5. In the **Modify the parameters of the connector** page:



MODIFY THE PARAMETERS OF THE CONNECTOR.
Defines the parameters of this connector.

Connectors - Active...

Type	String
Name	
Default Value	
Label	
Quick Help	
Option	☐ Optional parameter ☐ Sensitive information ☐ Validate as an email address Maximum length

a. In the **Type** list, select the parameter type as specified in the parameter description.

b. In the **Name** box, type the parameter name exactly as it appears in the parameter description. Parameter names are case sensitive.

c. In the **Default Value** box, enter the default value specified in the parameter description.

Important: Do not set the value that you want to use for a specific source. The value that you enter here will be used for all sources defined using this connector so it must be set to the recommended default value. You will be able to change the value for each source later, in the **Add Source** and **Source: ... General** pages of the Administration Tool.

d. In the **Label** box, enter the label that you want to see for this parameter.

Example: To easily link the label to the hidden parameter, you can simply use the parameter name, and if applicable, insert spaces between concatenated words. For the **BatchSize** hidden parameter, enter `Batch Size` for the label.

Note: To create multilingual labels and quick help messages, use the following syntax: `<@ln>text</@>`, where *ln* is replaced by the language initials—the languages of the Administration Tool are English (en) and French (fr).

Example: `<@fr>Chemin d'accès du fichier de configuration</@><@en>Configuration File Path</@>` is a label which is displayed differently in the French and English versions of the Administration Tool.

Tip: The language of the Administration Tool can be modified by pressing the following key combination: `Ctrl+Alt+Page Up`.

e. Optionally, in **Quick Help**, enter the help text that you want to see for this parameter when clicking the question mark button that will appear beside the parameter value.

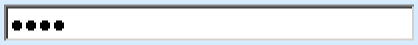
Tip: Copy and paste key elements of the parameter description.

f. When **Predefined values** is selected in the **Type** parameter, in the **Value** box that appears, enter the parameter values that you want to see available in the drop-down parameter that will appear in the Administration Tool interface. Enter one value per line. The entered values must exactly match the values listed in the hidden parameter description.

g. Select the **Optional parameter** check box when you want to identify this parameter as an optional parameter. When cleared, CES does not allow you to save changes when the parameter is empty. This parameter does not appear for **Boolean** and **Predefined values** parameter types.

h. Select the **Sensitive information** check box for password or other sensitive parameter so that, in the Administration Tool pages where the parameter appears, the typed characters appear as dots to mask them. This parameter appears only for the **String** type.

Example: When you select the **Sensitive information** check box for a parameter, the characters typed appear as follows in the text box:



i. Select the **Validate as an email address** check box when you want CES to validate that the text string that a user enters in this parameter respects the format of a valid email address. This parameter appears only for the **String** type.

j. In the **Maximum length** box, enter the maximum number of characters for the string. This parameter appears only for the **String** type. When you enter `0`, the length of the string is not limited.

k. Click **Save**.

6. Back in the **Connector** page, click **Apply Changes**.

   The hidden parameter now appears in the **Add Source** and **Source: ... General** pages of the Administration Tool for the selected source. You can change the parameter value from these pages. Refer to the documentation for each connector for details.

Note: When you want to modify a hidden source parameter, you must first delete it, and then redefine it with the modified values.

9.3 How CES Handles the Sitecore Permission Model with the Legacy Connector

Deprecated

Sitecore handles security access rights slightly different than CES. CES denies access to a document whenever the user (or one of its roles) is denied access on this document, always overruling `allow`. In Sitecore, the same rule applies, but there are exceptions no matter if the access rule is explicitly set on an item or not, inherited from another item, or if the user is an administrator, etc.

CES overcomes this problem by implementing heuristic detecting cases where a user might have the right to see a document; even if that user is part of the role that does not have the right to see the same document.

The counterpart of that solution is that security changes on the Sitecore side are not considered by CES in an incremental refresh run by default. You must have a refresh schedule set on your sources if such security is present in your site. Fortunately, CES logs a warning during the source indexing if such security is encountered. If your security model works like the one expected by CES, the connector operates normally.

Note: You can also set the `IncrementalRefreshIncludeSecurity` advanced parameter to `True` (see "Modifying Hidden Sitecore Source Parameters for the Legacy Connector" on page 24).

9.4 Defining a Sitecore Mapping File for the Legacy Connector

Deprecated

Sitecore metadata can be organized in various ways. Depending on how the metadata is organized, it might be difficult for the Sitecore connector to retrieve all the fields that compose a Sitecore item.

Example: A single Sitecore item can be composed of multiple fields as well as multiple references to other Sitecore items. Those referenced items can also have many fields and references, etc.

This topic explains how you can write an XML mapping file which can be used to map metadata fields to CES fields. You can also use this file to modify the way referenced Sitecore field values are resolved.

Example: You can change the clickable URL of indexed elements using a mapping file when the **Index if no Layout** source option is enabled (see "Index if no Layout" on page 20).

The following code presents a basic XML structure of a mapping file.

```
<?xml version="1.0" encoding="utf-8" ?>
<Sitecore>
  <CommonMappings>
    <Fields>
      <FieldName>mapping value</FieldName>
    </Fields>
  </CommonMappings>
  <Mapping template="{00000000-0000-0000-0000-000000000000}">
    <Fields>
      <FieldName>mapping value</FieldName>
    </Fields>
  </Mapping>
  <Mapping item="{00000000-0000-0000-0000-000000000000}">
    <Fields>
      <FieldName>mapping value</FieldName>
    </Fields>
  </Mapping>
  <Ignore template="{00000000-0000-0000-0000-000000000000}">
  <Ignore item="{00000000-0000-0000-0000-000000000000}">
```

<CommonMappings>

Defines field mappings that apply to all Sitecore items.

<Mapping>

Defines field mappings that apply to all Sitecore items that use the templates specified in the template attribute. Field mapping can also be applied directly on items by specifying the item attribute. Templates and items are identified by their Sitecore ID (GUID) and separated by a pipe (|) character. If both attributes are set, only the template attribute is considered. It is a better approach to set mappings on templates rather than on items.

<Ignore>

Specifies templates and items that should be ignored by the connector. Templates and items are identified by their Sitecore ID (GUID). Each entry accepts only one template or item. If both attributes are set, only the template attribute is considered.

<FieldName>

Defines the field targeted by the mapping. The name of the entry corresponds to the name of the Coveo system or custom field defined in the field set used by the source.

mapping value

This value is the mapping itself. It is composed of free text and metadata tags. The connector supports different metadata tag syntaxes:

- `%[<metadata>]`
- `%{<metadata>:<mapping value>}`
- `@[<fully_qualified_class_name>].`

The metadata tag syntaxes are illustrated in the following examples.

Example: The following mapping tells the connector to set the `Title` field of a document with the value of the `_CESSCDisplayName` metadata. This applies to all Sitecore documents. It is possible to browse the item hierarchy using the referenced metadata fields.

```
<?xml version="1.0" encoding="utf-8" ?>
<Sitecore>
  <CommonMappings>
    <Fields>
      <Title>%[_CESSCDisplayName]</Title>
      <Body>%[Description]</Body>
    </Fields>
  </CommonMappings>
</Sitecore>
```

Example: A Sitecore item is based on the `Book` template. This template defines a `Contributor` field which is a reference to a `Person` item somewhere else in the content tree. The `Person` item contains several fields, including the `Firstname` and the `Lastname` fields.
Here are the different possibilities:

- The mapping file is left as is, so the metadata `Contributor` is ignored. This is the default connector behavior.
- The **Index the document's metadata** source parameter is selected, so the `Contributor` metadata is indexed and contains a reference to the `Person` item (GUID).
- The mapping file is updated such that the `Firstname` and `Lastname` fields are concatenated to form the full name of the contributor, which is then put into the `Contributor` custom field.

```
<?xml version="1.0" encoding="utf-8" ?>
<Sitecore>
  <Mapping template="{AF0AD32B-8188-45E0-B354-5226F41D801B}"> <!--BookTemplate-->
    <Fields>
      <Contributor>%[Contributor.Lastname], %[Contributor.Firstname]</Contributor>
    </Fields>
  </Mapping>
</Sitecore>
```

When the field `Contributor` is a multiple value reference (i.e., the book might have more than one contributor), the user can tell the connector to concatenate the values by using the following mapping syntax `%{<metadata>:<mapping value>}`:

```
<?xml version="1.0" encoding="utf-8" ?>
<Sitecore>
  <Mapping template="{AF0AD32B-8188-45E0-B354-5226F41D801B}"> <!--BookTemplate-->
    <Fields>
      <Contributors>%{Contributor:%[Lastname], %[Firstname]}</Contributors>
    </Fields>
  </Mapping>
</Sitecore>
```

Multiple value references can be organized by template.

Example: In the *Nicam* demo site (default demo website installed with Sitecore), a camera contains a field called `Accessories`. These accessories are built from different templates: `Lenses`, `Flash`, or `Other Accessories`.
To organize the value references by template, use the following syntax `%{<metadata|template ID1|template ID2|…>:<mapping value>}`. Template IDs must be used without the curly brackets `{, }`:

```
<?xml version="1.0" encoding="utf-8" ?>
<Sitecore>
 <Mapping template="{B072B7C7-6F3F-4316-B8D7-010629AEBEF1}"> <!--SLR-->
  <Fields>
   <Accessories>%[Accessories.Title]</Accessories>
   <Lenses>%{Accessories|8FAC8E12-7459-43F8-97E8-1BC6840B9226:%[Title],%[Focal length]}</Lenses >
   <Flash>%{Accessories|95681CF6-3635-49EC-A09A-CC548FA62389:%[Title],%[Guide number]}</Flash>
   <Misc>%{Accessories|A93FA2C4-3AE4-45C2-8C3F-EFA7E129537E:%[Title]}</Misc>
  </Fields>
 </Mapping>
</Sitecore>
```

Finally, the mapping can be handled programmatically by using the `@[<fully_qualified_class_name>]` syntax. The connector can then use a custom class to resolve a field value if none of the out-of-the-box mapping solutions match your needs. Although this can be a slower approach (the connector has no control over code execution inside the class), it is a very powerful way to resolve mappings when complex rules apply.

Example: A Sitecore item has the following fields: `long description`, `short description`, `title` and `name`. Only the `name` is required in Sitecore, but you want to use the `long description` as well as the `short description` if one of them is present. If not, you will use the title before the name.

```
<?xml version="1.0" encoding="utf-8" ?>
<Sitecore>
  <Mapping template="{AF0AD32B-8188-45E0-B354-5226F41D801B}">
    <Fields>
      <Description>@[MyNamespace.MyDescResolver, MyAssembly]</Description >
    </Fields>
  </Mapping>
</Sitecore>
```

This is the code of the class contained in `MyAssembly.dll` used to resolve the description field:

```
using System;
using System.Collections;
using System.Collections.Generic;
using Coveo.CES.CustomCrawlers.Sitecore.ContentService;
using Coveo.CES.CustomCrawlers.Sitecore.Interfaces;

namespace MyNamespace {

public class MyDescResolver: ISitecoreMappingResolver
{
    [CLSCompliant(false)]
    public string ResolveMapping(ContentItemMeta p_Meta,
                                 Hashtable p_ProcessedMeta,
                                 List<string> p_MediaReferences)
    {
        string desc = null;
        string shortdesc = null;
        string title = null;

        foreach (ContentItemField field in p_Meta.Fields) {
            if (field.Name == "long description") {
                desc = field.Value.StringValue;
                   } else if (field.Name == "short description") {
                shortdesc = field.Value.StringValue;
                   } else if (field.Name == "title") {
                title = field.Value.StringValue;
                   }
        }

        return desc ?? shortdesc ?? title ?? p_ProcessedMeta["_CESSCName"].ToString();
    }
}
} //namespace
```

The class must implement the `ISitecoreMappingResolver` interface. Your assembly will have to reference `Coveo.CES.CustomCrawlers.Sitecore.dll`, after which you only need to implement the `ResolveMapping` method. This method has the following parameters:

- `ContentItemMeta`: The Sitecore metadata of the item.
- `Hashtable`: The already processed system metadata (see "About Sitecore Metadata with the Legacy Connector" on page 33).
- `List<string>`: The list of media items referenced by the items. This list can be modified if required.

Note: Mapping values are case-sensitive. If the `Sitecore` field is called `Short Description`, the following syntax does not work:

```
<?xml version="1.0" encoding="utf-8" ?>
<Sitecore>
  <Mapping template="{AF0AD32B-8188-45E0-B354-5226F41D801B}">
    <Fields>
      <ShortDesc>%[short description]</ShortDesc > <!—No field will match-->
    </Fields>
  </Mapping>
</Sitecore>
```

9.5 About Sitecore Metadata with the Legacy Connector

Deprecated

Essentially, a Sitecore item is made of several metadata fields and values. A small subset of this metadata is mapped to CES custom and system fields in such a way that once combined together, they form a new document in the index.

You can select the **Index the document's metadata** Sitecore source check box to be able to perform free text queries against all available metadata (see "Configuring and Indexing a Sitecore Source for the Legacy Connector" on page 15). However, this approach does not allow sorting, filtering, or grouping by metadata fields. To do so, you must create a custom field in CES that maps to a specific metadata field from Sitecore. Then you will be able to sort, filter and group search results by this custom field.

The following list presents the basic metadata fields that exist for any Sitecore item.

Note: The field names are basically the concatenation of the `_CESSC` prefix and their Sitecore item property field name. Some of the available metadata fields are automatically mapped to CES system fields (shown as values enclosed between parentheses in the list).

- _CESSCChildrenIDs
- _CESSCContentPath
- _CESSCCreated (sysaddeddate)
- _CESSCCreatedBy
- _CESSCDisplayName
- _CESSCFriendlyUrl
- _CESSCFullPath
- _CESSCHasLayout
- _CESSCID
- _CESSCIsContentItem
- _CESSCIsFullyQualified
- _CESSCIsMasterPart

- _CESSCIsMediaItem
- _CESSCKey
- _CESSCLanguageName
- _CESSCLanguages
- _CESSCLanguageTitle
- _CESSCLocker
- _CESSCLongID
- _CESSCMediaPath
- _CESSCMediaUrl
- _CESSCName
- _CESSCParentID
- _CESSCParentPath
- _CESSCPath (sysuri)
- _CESSCRevision
- _CESSCServerBaseUrl
- _CESSCTemplateID
- _CESSCTemplateName
- _CESSCUpdated (sysdate)
- _CESSCUpdatedBy (sysauthorloginname)
- _CESSCUri
- _CESSCVersion
- _CESSCWorkflowstate

Any other item field in Sitecore is available as a metadata field in CES and is also preceded by the _CESSC prefix. Reference type fields in Sitecore are also available as metadata fields in CES, with the distinction that their value contains the reference to the other item as a GUID (Globally Unique Identifier).

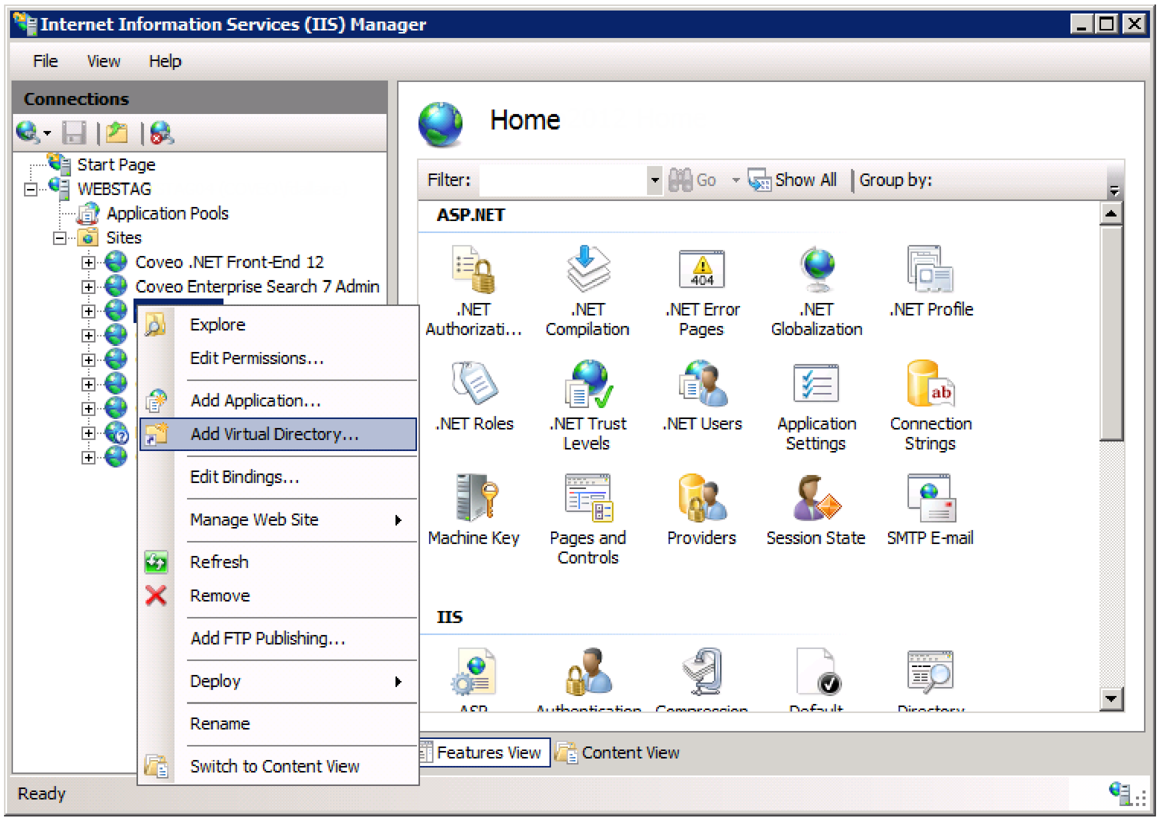
10. Integrating the Coveo .NET Search Interface in a Sitecore Website

Integrating the Coveo .NET search interface into Sitecore allows your end-users to search your Coveo index content directly from your Sitecore website using a feature-rich Coveo search interface.

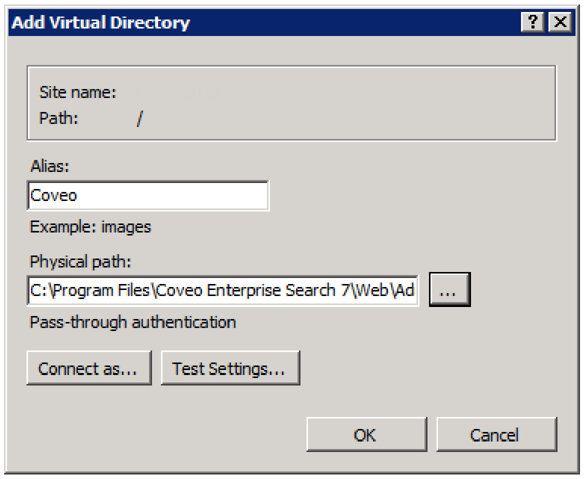
Note: If not already done, index the content of your Sitecore site so you can include it in the scope of the search interface to integrate to Sitecore.

To integrate the Coveo search interface in a Sitecore website

1. Using an administrator account, log on to the Sitecore server.
2. If not already done, install or update the Coveo search interfaces on the Sitecore server.
3. Start the IIS Manager (on the Windows taskbar, select **Start** > **Administrative Tools** > **Internet Information Services (IIS) Manager**).
4. In **Internet Information Services (IIS) Manager**, create a virtual directory for your Sitecore web application:
   a. In the **Connections** panel, expand to the root of your Sitecore web application, right-click it and select **Add Virtual Directory**



b. In the **Add Virtual Directory** dialog box, enter `Coveo` in the **Alias** box and the `[.NET_Front-End_Path]\Web\Coveo` folder in the **Physical path** box.



5. Using a text editor:

   a. Open the Sitecore `web.config` file.

   Note: It is recommended to make a backup of the `web.config` file before editing it.

   b. In the `configuration\configSections` section, add the following code:

```
<sectionGroup name="coveoCnlWeb">
  <section name="customContent" type="System.Configuration.SingleTagSectionHandler, System,
Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
</sectionGroup>
<sectionGroup name="coveoEnterpriseSearch">
  <section name="database" type="System.Configuration.SingleTagSectionHandler, System,
Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <section name="analytics" type="System.Configuration.SingleTagSectionHandler, System,
Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <section name="locations" type="System.Configuration.SingleTagSectionHandler, System,
Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <section name="server" type="System.Configuration.SingleTagSectionHandler, System,
Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
</sectionGroup>
```

   c. In the `configuration\system.web\pages` section, add the following code:

```
<controls>
  <add tagPrefix="cnla" namespace="Coveo.CNL.Web.Ajax" assembly="Coveo.CNL.Web,
Version=12.0.0.0, Culture=neutral, PublicKeyToken=44110d16825221f2" />
  <add tagPrefix="cnlb" namespace="Coveo.CNL.Web.BetterControls" assembly="Coveo.CNL.Web,
Version=12.0.0.0, Culture=neutral, PublicKeyToken=44110d16825221f2" />
  <add tagPrefix="cnlm" namespace="Coveo.CNL.Web.Misc" assembly="Coveo.CNL.Web,
Version=12.0.0.0, Culture=neutral, PublicKeyToken=44110d16825221f2" />
  <add tagPrefix="cnlv" namespace="Coveo.CNL.Web.Validators" assembly="Coveo.CNL.Web,
Version=12.0.0.0, Culture=neutral, PublicKeyToken=44110d16825221f2" />
  <add tagPrefix="cnlvs" namespace="Coveo.CNL.Web.Validators.ServerSide"
assembly="Coveo.CNL.Web, Version=12.0.0.0, Culture=neutral, PublicKeyToken=44110d16825221f2"
/>
  <add tagPrefix="cnlw" namespace="Coveo.CNL.Web.Widgets" assembly="Coveo.CNL.Web,
Version=12.0.0.0, Culture=neutral, PublicKeyToken=44110d16825221f2" />
  <add tagPrefix="ces" namespace="Coveo.CES.Web.Search.Controls"
assembly="Coveo.CES.Web.Search, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=44110d16825221f2" />
  <add tagPrefix="cs" namespace="Coveo.CES.Web.Search.Controls.CustomerService"
assembly="Coveo.CES.Web.Search, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=44110d16825221f2" />
</controls>
<namespaces>
  <add namespace="Coveo.CES.Web.Search" />
  <add namespace="Coveo.CNL.Web" />
</namespaces>
```

d. In the `configuration\system.web\compilation` section, add the following code:

```
<assemblies>
  <add assembly="Coveo.CNL, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=44110d16825221f2" />
  <add assembly="Coveo.CNL.Web, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=44110d16825221f2" />
  <add assembly="Coveo.CES.Common, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=44110d16825221f2" />
  <add assembly="Coveo.CES.Web.Search, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=44110d16825221f2" />
</assemblies>
```

e. In the `configuration` section, add the following code:

```
<coveoCnlWeb>
  <customContent uri="~/Coveo/" anonymousUri="~/Coveo/Anonymous/" />
</coveoCnlWeb>
<coveoEnterpriseSearch>
  <server hostname="localhost" port="52800" servicesHostname="localhost" servicesPort="52810"
instance="default" mirrorName="default" sslCertificatePath="C:\Program Files\Coveo .NET Front-
End 12\Web\certificate.p12" />
  <database enabled="false" connectionString="mongodb://localhost/databaseName" />
  <analytics enabled="false" connectionString="Data Source=yourServerName;Initial
Catalog=CoveoAnalytics;Integrated Security=SSPI;" />
</coveoEnterpriseSearch>
```

f. Save the file.

6. Integrate the search interface in a Sitecore layout or sublayout:

- Add the following code at the appropriate location in the Sitecore `.aspx` layout or sublayout file in which you want to integrate the Coveo search interface control.

```
<%@ Register TagPrefix="ces" Namespace="Coveo.CES.Web.Search.Controls"
Assembly="Coveo.CES.Web.Search, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=44110d16825221f2" %>
<%@ Assembly Name="Coveo.CNL, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=44110d16825221f2" %>
<%@ Assembly Name="Coveo.CNL.Web, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=44110d16825221f2" %>

<asp:Panel ID="pnResultsPanel" runat="server">
      <ces:SearchHub id="c" runat="server"/>
</asp:Panel>
```

OR

a. When you chose to index Sitecore security using a security provider, you must rather add the following code that includes a script to allow the search interface to display the search results corresponding to the currently logged on user.

Using the Sitecore API, the `c_OverrideUser` script retrieves the currently logged on user and passes this user to the Coveo search Interface.

Note: CES security elements are case sensitive. The Sitecore connector always indexes security elements in lower-case. Ensure to provide a lower-case username for the security to be resolved correctly.

```
<%@ Control Language="C#" AutoEventWireup="true" Inherits="layouts_PFCEnergy_
CoveoSearchResults" Codebehind="CoveoSearchResults.ascx.cs" %>
<%@ Register TagPrefix="ces" Namespace="Coveo.CES.Web.Search.Controls"
Assembly="Coveo.CES.Web.Search, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=44110d16825221f2" %>
<%@ Assembly Name="Coveo.CNL, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=44110d16825221f2" %>
<%@ Assembly Name="Coveo.CNL.Web, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=44110d16825221f2" %>

<script runat="server">
void c_OverrideUser(object p_Sender, Coveo.CES.Web.Search.Controls.OverrideUserEventArgs p_
Args)
{
  // The name of the security provider defined under the Admin Tool
  string securityProviderName = "Sitecore Security Provider";
  Coveo.CES.Web.Search.Providers.ICESUserIdentityFactory factory =
Coveo.CES.Web.Search.Providers.SearchProviderFactory.CreateDefaultUserIdentityFactory();
  // Get the username from SitecoreAPI
  string userName = Sitecore.Context.GetUserName().ToLower();
  // Add to the collection of identities
  Coveo.CES.Web.Search.Providers.IUserIdentity user = factory.CreateSecurityProviderUser
(userName, securityProviderName, false, null);
  p_Args.AdditionalIdentities.Add(user);
}
</script>

<div id="contentMain">
  <asp:Panel ID="pnResultsPanel" runat="server">
    <ces:SearchHub id="c" runat="server" OnOverrideUser="c_OverrideUser"/>
  </asp:Panel>
</div>
```

Note: `OnOverrideUser` event is not available for all Coveo search controls such as `QuickSearch` and `SearchBox` controls.

The search interface can target an explicit index source as shown in the following script.

```
<script runat="server">
override void OnInit(EventArgs p_Args)
{
   SearchBinding.MainSearchObject.SetupSearchBuilder += this.Search_SetupSearchBuilder;
   base.OnInit(p_Args);
}

void Search_SetupSearchBuilder(object p_Sender, SetupSearchBuilderEventArgs p_Args)
{
    p_Args.Builder.AddConstantExpression("@Source=MySourceName");
}
</script>
```

7. Using a browser, access the modified page to view the search hub.

   The **Front-End Server Configuration** first time setup page appears to allow you to complete the Coveo Front-End installation.

Tip: If you notice that some of Coveo images or JavaScript are missing because they do not load in the search interface, you can fix this problem with a simple edit of the Sitecore `web.config` file (not the Coveo `web.config` file). In the file, locate the `<setting name="IgnoreUrlPrefixes"...>` tag and add the `|/Coveo/` string to the `value` attribute.

Example:

```
<setting name="IgnoreUrlPrefixes"
value="/sitecore/default.aspx|/trace.axd|/webresource.axd|/Coveo/"/>
```

What's Next?

Consider also integrating the Coveo search in the Sitecore Content Editor (see "Integrating the Coveo .NET Search in the Sitecore Content Editor" on page 40).

11. Integrating the Coveo .NET Search in the Sitecore Content Editor

You can integrate the Coveo .NET search interfaces in your Sitecore website to provide a better search experience to your website users (see "Integrating the Coveo .NET Search Interface in a Sitecore Website" on page 35).

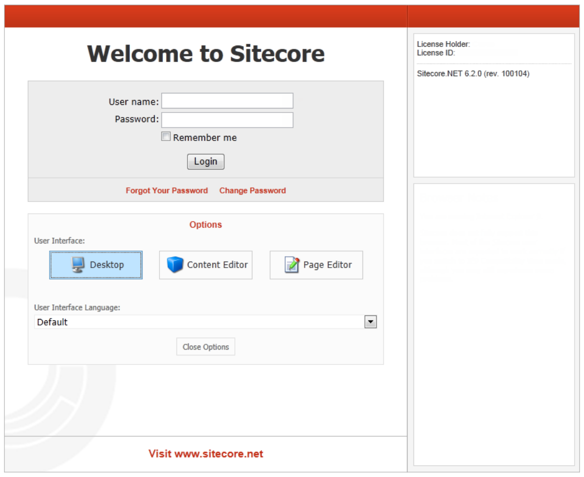
You can also integrate the Coveo search in the Sitecore Content Editor so that people contributing to the site content can also take advantage of the Coveo search features and more easily find information in the whole content, not just published content.

Note: The integration described in this topic works with Coveo .NET Front-End 12.0.61+.

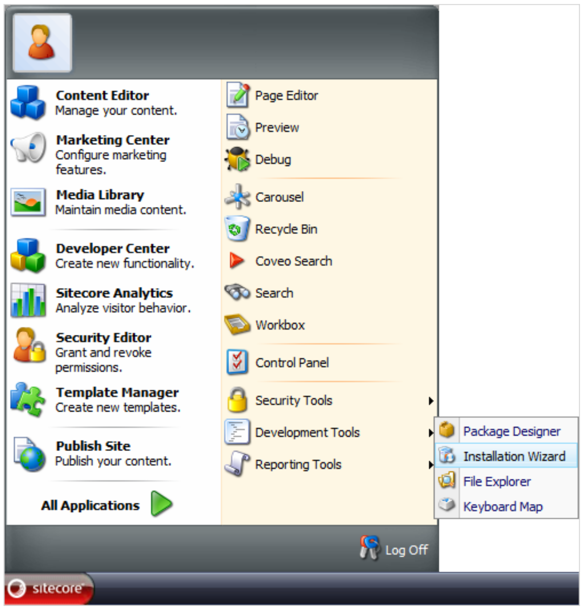
Tip: Typically, a Sitecore site is managed with a `Master` database containing the whole content and a `Web` database containing only the published content. You can create separate sources for these two databases, and then assign the `Web` source to the scope of the search interface integrated in the website, and the `Master` source to the scope of the search interface integrated in the Content Editor. This way, all of the unpublished content is searchable from the Sitecore Desktop.

To integrate the Coveo search in the Sitecore Content Editor

1. If not already done, install the Coveo Front-End components on your Sitecore server.
2. Install the Coveo Content Search for Sitecore:
   a. Log in to the Sitecore Desktop user interface using an administrative account.



b. On the Sitecore Desktop, select **Sitecore** > **Development Tools** > **Installation Wizard**.

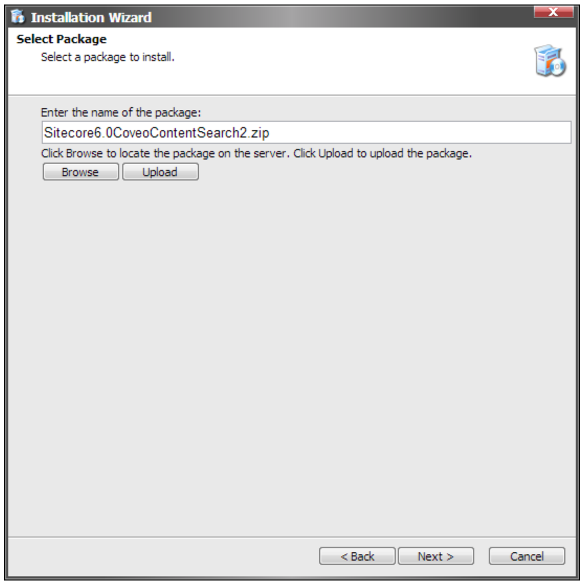


c. In the **Installation Wizard** dialog box:

i. In the **Welcome to the Install Package Wizard** screen, click **Next**.

ii. In the **Select Package** screen:

i. Click **Upload**, and then browse the Coveo Master server to select the Coveo Content Search package distributed with CES:



Enter the name of the package:
Sitecore6.0CoveoContentSearch2.zip
Click Browse to locate the package on the server. Click Upload to upload the package.

```
[CES_Install_Path]\Bin\Sitecore6.0CoveoContentSearch2.zip
```

Note: Use the `Sitecore6.0CoveoContentSearch.zip` package only when you use the Sitecore legacy connector.

ii. Click **Next**.

iii. In the **Ready to Install** screen, click **Install**.

Note: If an `Access Denied error` is displayed for the `bin_install` folder while installing the package, verify the security permissions for `ASP.NET` in the bin folder of Sitecore or install the files manually.

  d. Select the **Restart the Sitecore client** option, and then click **Finish**.

3. Using an administrator account, connect to your Sitecore server.

4. Create the Coveo virtual directory:

  a. Open IIS.

  b. In IIS, locate and right-click the website that corresponds to your Sitecore instance, and then select **Add virtual directory**.

  c. In the **Create Virtual Directory** dialog box:

    i. In the **Alias** box, enter `Coveo`.

    ii. In the **Physical path** box, enter `[.NET_Front-End_Path]\Web\Coveo`, typically `C:\Program Files\Coveo .NET Front-End 12\Web\Coveo`.

  d. Close IIS.

5. Using a text editor, edit the Sitecore `web.config` file:

Example: For an instance named `Sitecore`, the file is located in the `C:\inetpub\wwwroot\Sitecore\WebSite` folder.

  a. Copy the following code and paste it just before the `</configSections>` tag.

```
<sectionGroup name="coveoCnlWeb">
  <section name="customContent" type="System.Configuration.SingleTagSectionHandler, System,
Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
</sectionGroup>
<sectionGroup name="coveoEnterpriseSearch">
  <section name="database" type="System.Configuration.SingleTagSectionHandler, System,
Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <section name="analytics" type="System.Configuration.SingleTagSectionHandler, System,
Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <section name="locations" type="System.Configuration.SingleTagSectionHandler, System,
Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <section name="server" type="System.Configuration.SingleTagSectionHandler, System,
Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
</sectionGroup>
```

  b. Copy the following code and paste it just before the `</controls>` tag, a child of the `<pages>` tag.

```
<add tagPrefix="cnla" namespace="Coveo.CNL.Web.Ajax" assembly="Coveo.CNL.Web,
Version=12.0.0.0, Culture=neutral, PublicKeyToken=44110d16825221f2"/>
<add tagPrefix="cnlb" namespace="Coveo.CNL.Web.BetterControls" assembly="Coveo.CNL.Web,
Version=12.0.0.0, Culture=neutral, PublicKeyToken=44110d16825221f2"/>
<add tagPrefix="cnlm" namespace="Coveo.CNL.Web.Misc" assembly="Coveo.CNL.Web,
Version=12.0.0.0, Culture=neutral, PublicKeyToken=44110d16825221f2"/>
<add tagPrefix="cnlv" namespace="Coveo.CNL.Web.Validators" assembly="Coveo.CNL.Web,
Version=12.0.0.0, Culture=neutral, PublicKeyToken=44110d16825221f2"/>
<add tagPrefix="cnlvs" namespace="Coveo.CNL.Web.Validators.ServerSide"
assembly="Coveo.CNL.Web, Version=12.0.0.0, Culture=neutral, PublicKeyToken=44110d16825221f2"/>
```

```
<add tagPrefix="cnlw" namespace="Coveo.CNL.Web.Widgets" assembly="Coveo.CNL.Web,
Version=12.0.0.0, Culture=neutral, PublicKeyToken=44110d16825221f2" />
<add tagPrefix="ces" namespace="Coveo.CES.Web.Search.Controls" assembly="Coveo.CES.Web.Search,
Version=12.0.0.0, Culture=neutral, PublicKeyToken=44110d16825221f2"/>
<add tagPrefix="cs" namespace="Coveo.CES.Web.Search.Controls.CustomerService"
assembly="Coveo.CES.Web.Search, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=44110d16825221f2" />
```

c. Copy the following code and paste it just before the `</namespace>` tag, a child of the `<namespaces>` tag.

```
<add namespace="Coveo.CES.Web.Search"/>
<add namespace="Coveo.CES.Web.Search.Controls" />
<add namespace="Coveo.CNL.Web" />
```

d. Copy the following code and paste it just before the `</assemblies>` tag, a child of the `<compilation>` tag.

```
<add assembly="Coveo.CNL, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=44110d16825221f2"/>
<add assembly="Coveo.CES.Common, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=44110d16825221f2" />
<add assembly="Coveo.CNL.Web, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=44110d16825221f2"/>
<add assembly="Coveo.CES.Web.Search, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=44110d16825221f2"/>
```

e. Copy the following code and paste it at the end of the file, just before the `</configuration>` tag.

```
<coveoCnlWeb>
  <customContent uri="~/Coveo/" anonymousUri="~/Coveo/Anonymous/"/>
</coveoCnlWeb>
<coveoEnterpriseSearch>
  <server hostname="localhost" port="52800"/>
  <database enabled="false" connectionString="mongodb://localhost/databaseName"/>
  <analytics enabled="false" connectionString="Data Source=yourServerName;Initial
Catalog=CoveoAnalytics;Integrated Security=SSPI;"/>
</coveoEnterpriseSearch>
```

f. Save the file.

6. When CES and Sitecore are not installed on the same server, copy the index certificate file from the Coveo Master server to the Sitecore server:

   a. Using an administrator account, connect to the Coveo Master server.

   b. Copy the index certificate file `[Index_Path]\Config\Certificates\cert-iis.p12`, typically `C:\CES7\Config\Certificates\cert-iis.p12`.

   c. Paste the certificate file on the Sitecore server into the Sitecore instance website.

   Example: For an instance named `Sitecore`, the default path is `C:\inetpub\wwwroot\Sitecore\Website`. You can also rename the certificate file such as `ces-certificate.p12`.

7. Configure the connection to the CES index. Using a text editor:

   a. Open the Sitecore instance `web.config` file.

   b. Under the `<coveoEnterpriseSearch>` tag, edit the `<server>` tag to include the following attributes:

- `hostname=”MyCoveoMasterServerHostName”` where you replace `MyCoveoMasterServerHostName` with the name of your Coveo Master server or `localhost` when CES and Sitecore are on the same machine.
- `sslCertificatePath=”C:\inetpub\wwwroot\Sitecore\Website\ces-certificate.p12”`, ensuring that the path points to your certificate file.
- `port=”52800”`, the default CES port number.

Example: When CES and Sitecore are on the same server.

```
<server hostname="localhost" port="52800" servicesHostname="localhost" servicesPort="52810"
instance="default" mirrorName="default" sslCertificatePath="C:\Program Files\Coveo .NET
Front-End 12\Web\ces-certificate.p12"/>
```

8. Configure the Sitecore source used by Coveo search:

   a. Using a text editor, open the `CoveoSearch.aspx` file.

   Example: For a Sitecore instance named `Sitecore`, the file is typically located in the `C:\inetpub\wwwroot\Sitecore\Website\sitecore\shell\Applications\Coveo\` folder.

   b. Locate the following code:

```
/* UNCOMMENT THIS SECTION BEFORE FIRST USE
p_Args.Builder.AddConstantExpression("@Source=MySourceName");
*/
```

   and replace it with:

```
p_Args.Builder.AddConstantExpression("@Source=\”YOUR SITECORE SOURCE NAME HERE\”");
```

   where you replace `YOUR SITECORE SOURCE NAME HERE` by the name of the source you created to contain the whole Sitecore content, typically indexing the `master` database.

   c. Save and close the file.

12. Resolving Slow Search Interface Loading in Sitecore

When a Coveo .NET Front-End search interface is integrated in a Sitecore website (see "Integrating the Coveo .NET Search Interface in a Sitecore Website" on page 35), long search interface page loading time can be experienced.

Note: This topic is not applicable to Coveo for Sitecore, but rather to a legacy integration of a Coveo .NET Front-End search interface in Sitecore.

This unusual slow rendering of the Coveo search interface pages may be caused by the Sitecore cache settings. The Coveo search interfaces take advantage of browser caching to prevent repeatedly loading frequently used components, and therefore significantly reduce page loading time.

To resolve slow search interface loading in Sitecore

1. Using an administrator account, connect to the Sitecore server.
2. Using a text editor:
    a. Open the Sitecore `web.config` file.
    b. In the file, as shown in the following `web.config` file excerpt, ensure that `DisableBrowserCaching` is set to `false`.

```
<!--  DISABLE BROWSER CACHING
 If true, all pages will have:
  Cache-Control: no-cache, no-store
  Pragma: no-cache
 in the http header
-->
 <setting name="DisableBrowserCaching" value="false" />
```

    c. Save the file.