



Coveo Enterprise Search 6.0

Security Provider API

Contents

Overview	4
Retrieving Values from the Configuration	4
Abstract.....	5
Security in CES	5
Group Expansion	5
Security on Each Document	5
The Security Cache	5
Early-Binding vs. Late-Binding.....	5
Early-Binding.....	6
Late-Binding	6
Samples.....	6
Wiki Early	7
WikiEarlyAPI	7
Crawler.cs	7
WikiEarlyCrawler.Crawler.PerformInitialize() Method	7
WikiEarlyCrawler.Crawler.Update() Method	7
WikiEarlyCrawler.Crawler.Run() Method	8
WikiEarlyCrawler.Crawler.AddACLs() Method.....	9
SecurityProvider.cs	9
WikiEarlyCrawler.SecurityProvider.Initialize() Method.....	9
WikiEarlyCrawler.SecurityProvider.Login() Method	10
WikiEarlyCrawler.SecurityProvider.ValidateSession() Method	10
WikiEarlyCrawler.SecurityProvider.ExpandGroup() method.....	11
Wiki Late	11
WikiLateAPI.....	11
Crawler.cs	11
WikiLateCrawler.Crawler.Run() Method.....	12
SecurityProvider.cs	12
WikiLateCrawler.SecurityProvider.Initialize() Method	13
WikiLateCrawler.SecurityProvider.Authorize() Method.....	13
Setup Procedures	14
Prerequisite	14
Directories and URLs	14
Define a New Security Provider	15

Define a New Connector	15
Create a Source and Index the Documents.....	16
Configure the Search Interface and Execute Search Queries.....	17
Advanced Topics	17
What is an External Security Provider	17
Error Objects and Interfaces	17
Configuration Values.....	19
Reserved Parameters	19
IConfigValues Interface.....	19
Multi-Threading	19

Overview

Coveo Enterprise Search (CES) natively supports security for several well-known systems such as *Windows, SharePoint, Microsoft Exchange, Lotus Notes, Novell*, etc. However, the security model of CES can also be extended by developing additional connectors that contain security logic. The security logic contained in a class is called External Security Provider. Usually, the security provider class is packaged in the same DLL as the connector.

Developing a security provider is done using Microsoft Visual Studio (2005 or higher). The following briefly describes the procedure to follow to do so:


1. If a project and a solution have not yet been created for the connector (for more information, refer to the [Getting Started](#) in the Open Connector API section), create a project of Class Library type and add a reference to the base library *Coveo.CES.CustomCrawlers.dll*.
2. Create a class that inherits from the *CrawlerSecurityProvider* Class.
3. Override the appropriate methods:
 - A connector does not necessarily need to override all the inherited methods. For instance, a connector that works in Early-Binding does not need to override the *Authorize* Method.
 - When overriding a method, the base version of that method must not be called, as it does nothing except return a “not implemented” error.

For the list of methods that can be overridden and that specifies when to override them, refer to *CrawlerSecurityProvider* Class.

Retrieving Values from the Configuration

In the security provider, it is possible to retrieve a parameter's value by calling one of the *GetConfigValue* methods in the base class.

To create an external security provider, perform the following procedure:

1. Open the Administration Tool and access the **Administrators** page (Configuration > Security).
2. In the left navigation pane, click **Security Providers**. The **Security Providers** page is displayed.
3. Click . The **Modify Security Provider** page is displayed.
4. In the **Parameters** text box, enter the parameters to pass to the security provider. The string entered must respect the following syntax: `Name1=Value1;Name2=Value2;Name3=Value3`; etc.

Important: The syntax allows users to specify as many parameters as necessary. Parameters are separated by semi-colons (;). For each parameter, there are two parts separated by the equal sign (=)–the first part being the name of the parameter and the second one the value.

5. Click  **Save**.

Important: Two special parameters–*Username* and *Password*–are not specified in the **Parameters** string but can be retrieved by calling the *GetConfigValue* methods. Their values correspond to the **User Identity** entered in the **Modify Security Provider** page.

Abstract

Security in CES

Group Expansion

In most systems, security is defined in terms of groups and users. A group contains users, and users are members of one or more groups. For each object to secure (a file for example), access is only granted to members of specific groups or certain users directly. The security on the object is defined by setting a list of groups and/or users.

In CES, security on documents is also defined by the concept of groups and users. Expanding a group means determining the list of users who are members of that group.

Security on Each Document

When documents are indexed using CES, the list of allowed and denied users/groups is saved in order to determine who can access the documents.

Hence, when a user performs a search query, CES returns only the documents that specific user has access to. If the security for a given document is set using a group, CES validates if the user is a member of that group.

The denied list always has precedence over the allowed list. For example, if a user is a member of group C, and group C is in the denied list for a given document, the user cannot see the said document even if he is part of the allowed list for that document. Another example, if a user is a member of groups A and B, and group B is in the denied list for a given document, the user cannot see the said document even if group A is in the allowed list.

The Security Cache

In order to quickly determine the users who are member of a group and vice versa, CES keeps the results of group expansions in a local cache, which is entitled *File Security Cache*. The group expansion process is done asynchronously, meaning the cache is refreshed every night to reflect changes that are made occasionally in *Active Directory* (e.g., users added to and removed from groups).

For certain external security providers, a mapping between each external user identity and a *Windows (Active Directory)* identity is specified. The *File Security Cache* also keeps that mapping. When such a mapping exists, users do not have to login manually using the login page of the **Search** page. The authentication process between the browser and Web server (e.g. *NTLM* or *Kerberos*) automatically identifies the user and, by using the mapping between external and *Windows* logins, CES grants access to the documents protected by the External Security Provider. However, if no mapping exists, users have to login manually using the login page.

Early-Binding vs. Late-Binding

Evaluating if a user has the rights to access a document can either be done during crawling (early-binding) or querying (late-binding).

Early-Binding

When a connector adds a document to the index, it attaches an Access Control List (ACL) to it. Hence, the ACL is saved in the index along with the rest of the document's information. Afterwards, when a user executes a search query, it is easy to determine in which list of documents to search for the expression queried—time is not lost searching for documents to which the user does not have access.

The advantage of early-binding over late-binding resides in the execution speed of search queries—early-binding is always faster and the difference can vary considerably. When certain queries are performed in certain environments, late-binding can practically be the same speed. However, most of the time, running a search query in late-binding is two to 10 times slower (sometimes even more). To better realize the impact on usability, a response time of less than a second can climb up to 15 seconds when using late-binding.

Important: If possible, the user should develop connectors that work in early-binding.

Late-Binding

Certain external applications require the impersonation of a user in order to determine the rights granted to that user. *Impersonating* a user means logging in as a specific user and trying to access the resource. With such a security model, attaching ACLs to documents when crawling can become difficult, even impossible.

In late-binding, the connector does not attach ACLs to documents. Instead, the access check is performed when performing a query:

1. The query engine gathers a first batch of documents containing the search expression.
2. For each document, the query engine checks in its internal Authorization Cache if the Authorize Method has recently been called. This determines if the querying user can access the document.
3. For the documents not present in the Authorization Cache, the query engine calls the Authorize Method in the External Security Provider. For each document, the method returns a Boolean indicating if the user can access the document or not. Booleans are saved in the Authorization Cache.
4. The documents that can be accessed are added in the query's results.
5. When only a few documents are kept—not enough to fill the result page—the query engine loops back at step 1 with the next batch of documents until there are enough documents to fill the page.

To support late-binding, an External Security Provider implements the Authorize Method.

Samples

The following security provider sample is a fictive Wiki system. The Wiki is a simple repository of documents, which can be of any type—an HTML page, image (GIF, PNG, etc.), simple text file, spreadsheet, etc.—and are stored directly in the file system. A parameter of the connector is the directory containing the documents.

Each document can have attachments. To mark a document as an attachment, it must have a name encoded in a particular way. For example, for the document *doc1.txt*, if the *!AttachmentOf!doc1.txt!Memo.txt* file is created, the document is marked as having an attachment named Memo.txt. Also, the *!AttachmentOf!doc1.txt!PieChart.png* file also marks the document as having an attachment.

Note: The sample does not include a user interface to edit the Wiki pages.

In order to demonstrate the difference between an early-binding and late-binding security provider, the Wiki connector exists in two modes: WikiEarly and WikiLate (two fictive Wiki applications). Both have their respective APIs to give access to their documents: WikiEarlyApi and WikiLateApi. It is easy to see that there is a common piece of code that must be shared. All the shared code is contained in `._Lib\Repository.cs`. This C# file is referenced as a link from both `.\WikiEarly\WikiEarlyApi\WikiEarlyApi.csproj` and `.\WikiLate\WikiLateApi\WikiLateApi.csproj`.

Wiki Early

WikiEarlyAPI

The following C# project represents the API giving access to documents contained in WikiEarly. The connector (security provider and connector) uses this API, which is made up of 5 classes:

- **WikiSession:** This class is the entry-point of the API. To access the documents, it is important to first login. The constructor of this class receives a username and password. If the login information is invalid, the constructor throws an exception. Once successfully authenticated, methods in this class can be called to access the documents.
- **WikiDocument:** This class represents a Wiki document.
- **WikiDocAttachment:** This class represents an attachment to a Wiki document.
- **WikiGroup:** This class represents a group of users. Used in WikiDocument's ACLs.
- **WikiUser:** This class represents a user. Used in WikiDocument's ACLs.

Crawler.cs

This class contains the crawling logic. It is instantiated in the `CESCustomCrawlers*.exe` process. It inherits from the CustomCrawler class from the base project `Coveo.CES.CustomCrawlers.dll`. The most interesting method in this class is `Run()`. The code to add and refresh in CES is standard. Note that some features are not supported by this connection for brevity reasons: it always *rebuilds*, it does not support live indexing, pause/resume, etc.

The `PerformInitialize()` and `Update()` methods simply create a new instance of the Repository class, which contains the code shared by WikiEarly and WikiLate. The path to the repository is obtained by calling `Context.GetConfigValue("RepositoryPath")`.

WikiEarlyCrawler.Crawler.PerformInitialize() Method

```
protected override void PerformInitialize()
{
    m_Repository = new Repository(Context.GetConfigValue("RepositoryPath"));
}
```

WikiEarlyCrawler.Crawler.Update() Method

```
protected override void Update()
{
    m_Repository = new Repository(Context.GetConfigValue("RepositoryPath"));
}
```

The main point of interest in this class is the management of ACLs. This connector works in early-binding, meaning the ACLs are added to each document during crawling. The `AddACLs()` method is called at two places: once for the document itself and another for each of its attachments.

WikiEarlyCrawler.Crawler.Run() Method

```
protected override void Run()
{
    Context.LogCrawlerMessage("Crawling WikiEarly documents...");

    // Validate the value of the parameter 'Uri'.
    string uriParam = Context.GetConfigValue(URI_PARAM_NAME);
    int markerPos = uriParam.IndexOf(PATHNAME_MARKER_IN_URI);
    if (markerPos == -1 || uriParam.IndexOf(PATHNAME_MARKER_IN_URI, markerPos + 1) != -
1) {
        throw new Exception(INVALID_URI_PARAM);
    }

    // Index the documents.
    WikiSession wikiSession = new WikiSession(m_Repository, Context.UserName,
Context.UserPassword);
    foreach (WikiDocument wikiDoc in wikiSession.ListDocuments()) {
        using (Document document = new Document(Context)) {
            try {
                // Fill the document's info.
                document.Uri =
Context.GetConfigValue(URI_PARAM_NAME).Replace(PATHNAME_MARKER_IN_URI,
wikiDoc.PathName);
                document.Data = wikiDoc.Content;
                AddACLs(document, wikiDoc);

                // Transfer the document's attachments.
                foreach (WikiDocAttachment attachment in wikiDoc.DocAttachments) {
                    Document attachmentDoc = new Document(Context);
                    try {
                        attachmentDoc.Uri = Context.GetConfigValue(URI_PARAM_NAME).Replace
(PATHNAME_MARKER_IN_URI, attachment.PathName);
                        AddACLs(attachmentDoc, wikiDoc);
                        attachmentDoc.Data = attachment.Content;
                        document.AddAttachment(attachmentDoc);
                        attachmentDoc = null;
                    } finally {
                        if (attachmentDoc != null) {
                            attachmentDoc.Dispose();
                            attachmentDoc = null;
                        }
                    }
                }
            }
        }

        // Give the document to the indexer.
        IndexDocument(document);
    } catch (ThreadAbortException) {
        // Let go.
    } catch (Exception e) {
        Context.LogCrawlerErrorMessage(e.Message, document.Uri);
        CheckPoint();
    }
}
}
```

The AddACLs() method repeatedly calls the AddACLEntry method on the CES document (Document Class).

WikiEarlyCrawler.Crawler.AddACLs() Method

```
private void AddACLs(Document p_Doc,
                    WikiDocument p_WikiDoc)
{
    foreach (WikiUser wikiUser in p_WikiDoc.AllowedUsers) {
        p_Doc.AddACLEntry(new ACEEntry(SecurityType.ExternalUser, wikiUser.Name,
Context.ExternalSecurityProviderName, true));
    }
    foreach (WikiGroup wikiGroup in p_WikiDoc.AllowedGroups) {
        p_Doc.AddACLEntry(new ACEEntry(SecurityType.ExternalGroup, wikiGroup.Name,
Context.ExternalSecurityProviderName, true));
    }
    foreach (WikiUser wikiUser in p_WikiDoc.DeniedUsers) {
        p_Doc.AddACLEntry(new ACEEntry(SecurityType.ExternalUser, wikiUser.Name,
Context.ExternalSecurityProviderName, false));
    }
    foreach (WikiGroup wikiGroup in p_WikiDoc.DeniedGroups) {
        p_Doc.AddACLEntry(new ACEEntry(SecurityType.ExternalGroup, wikiGroup.Name,
Context.ExternalSecurityProviderName, false));
    }
}
```

SecurityProvider.cs

This class contains the security logic. Usually, it is instantiated in *CESService*.exe* process. It inherits from the *CrawlerSecurityProvider* class from the base project *Coveo.CES.CustomCrawlers.dll*. Note that the *Authorize* Method is not implemented in this sample, since it is specific to early-binding.

The *Initialize()* method simply creates a new instance of the *Repository* class, which contains the code shared by *WikiEarly* and *WikiLate*. The path to the repository is obtained by calling *GetConfigValue("RepositoryPath")*, then a *WikiSession* is created. The username and password used for this session are obtained by calling *GetConfigValue("Username")* and *GetConfigValue("Password")*. This “static” method is an optimization which avoids repeatedly creating new login information at each group expansion. The login used must be granted enough rights to perform group expansion.

WikiEarlyCrawler.SecurityProvider.Initialize() Method

```
public override InitializeError Initialize()
{
    m_Repository = new Repository(GetConfigValue("RepositoryPath"));
    m_WikiSession = new WikiSession(m_Repository, GetConfigValue("Username"),
GetConfigValue("Password"));
    return ErrorOk;
}
```

The two following methods—*Login()* and *ValidateSession()*—which work hand in hand, are required as this sample assumes there is no mapping between Wiki and Windows users. The Search Interface must be configured to use the *WikiEarly* security provider. When the user first accesses the Search page, he is prompted to login. When he clicks the **Login** button to confirm his username and password, *Login()* gets called. The method validates the login and, if valid, returns a BLOB that is saved in the browser’s cookies. Then, each time the user executes a search query, the BLOB is passed back to *Validate()* to make sure the login is still valid. If still valid, the user can perform his query without having to login again; if not, the login page is displayed.

To easily read and write BLOBs, these methods use the *PropTree* Class. However, any other means of manipulating BLOBs could have been used.

WikiEarlyCrawler.SecurityProvider.Login() Method

```
public override LoginError Login(string p_UserName,
                                string p_Password,
                                out byte[] p_OutSessionData)
{
    LoginError retErr;
    try {
        new WikiSession(m_Repository, p_UserName, p_Password);
        // No exception thrown, then login is valid.
        PropTree sessionData = new PropTree();
        sessionData.SetStringValue("u", p_UserName);
        sessionData.SetStringValue("p", p_Password);
        p_OutSessionData = sessionData.StreamAsBinary();
        retErr = ErrorOk;
    } catch (Exception) {
        // Login NOT valid.
        p_OutSessionData = null;
        retErr = ErrorBadUserOrPassword;
    }
    return retErr;
}
```

WikiEarlyCrawler.SecurityProvider.ValidateSession() Method

```
public override ValidateSessionError ValidateSession(byte[] p_SessionData)
{
    ValidateSessionError retErr;
    PropTree sessionData = (p_SessionData != null ? PropTree.Parse(p_SessionData) : new
PropTree());
    try {
        new WikiSession(m_Repository, sessionData.GetStringValue("u"),
sessionData.GetStringValue("p"));
        // No exception thrown, then login is valid.
        retErr = ErrorOk;
    } catch (Exception) {
        // Login NOT valid.
        retErr = ErrorInvalidSession;
    }
    return retErr;
}
```

Finally, there is the `ExpandGroup()` method. This method receives the name of a Wiki group and returns the list of Wiki users it contains. `ExpandGroup()` is called asynchronously to populate/refresh the File Security Cache.

WikiEarlyCrawler.SecurityProvider.ExpandGroup() method

```
public override ExpandGroupError ExpandGroup(string p_GroupName,
                                             ref bool p_Quit,
                                             out string[] p_OutUserNames)
{
    ExpandGroupError retErr;
    WikiGroup wikiGroup = m_WikiSession.GetGroup(p_GroupName);
    if (wikiGroup != null) {
        ReadOnlyCollection<WikiUser> wikiUsers = wikiGroup.Users;
        int seqNo = 0;
        p_OutUserNames = new string[wikiUsers.Count];
        foreach (WikiUser wikiUser in wikiUsers) {
            p_OutUserNames[seqNo++] = wikiUser.Name;
            if (p_Quit) {
                break;
            }
        }
        retErr = ErrorOk;
    } else {
        p_OutUserNames = null;
        retErr = ErrorInvalidGroup;
    }
    return retErr;
}
```

Wiki Late

WikiLateAPI

This C# project represents the API that allows the users to access documents contained in WikiLate. The connector (security provider and connector) uses this API. This API is the same as WikiEarlyAPI, except that the classes WikiGroup and WikiUser are not used. This API is made up of three classes:

- **WikiSession Class:** This class is the entry-point of the API. To access the documents, it is important to first login. The constructor of this class receives a username and password. If the login information entered is not valid, the constructor throws an exception. Once successfully authenticated, methods in this class can be called to access the documents.
- **WikiDocument Class:** This class represents a Wiki document.
- **WikiDocAttachment Class:** This class represents an attachment to a Wiki document.

Crawler.cs

As for the WikiEarly class, it contains the crawling logic. In this section, we focus only on the differences. All the explanations are not repeated here (for more information, refer to [Wiki Early](#)).

The PerformInitialize() and Update() methods are identical.

In the Run() method, there is only one difference: the two calls to AddACLs() are absent because in late-binding, ACLs are not added to the documents during crawling. The rights to access a document are checked at query-time. That is the purpose of the Authorize() method in *SecurityProvider.cs*.

WikiLateCrawler.Crawler.Run() Method

```
protected override void Run()
{
    Context.LogCrawlerMessage("Crawling WikiLate documents...");

    // Validate the value of the parameter 'Uri'.
    string uriParam = Context.GetConfigValue(URI_PARAM_NAME);
    int markerPos = uriParam.IndexOf(PATHNAME_MARKER_IN_URI);
    if (markerPos == -1 || uriParam.IndexOf(PATHNAME_MARKER_IN_URI, markerPos + 1) != -
1) {
        throw new Exception(INVALID_URI_PARAM);
    }

    // Index the documents.
    WikiSession wikiSession = new WikiSession(m_Repository, Context.UserName,
Context.UserPassword);
    foreach (WikiDocument wikiDoc in wikiSession.ListDocuments()) {
        using (Document document = new Document(Context)) {
            try {
                // Fill the document's info.
                document.Uri =
                    Context.GetConfigValue(URI_PARAM_NAME).Replace(PATHNAME_MARKER_IN_URI,
wikiDoc.PathName);
                document.Data = wikiDoc.Content;

                // Transfer the document's attachments.
                foreach (WikiDocAttachment attachment in wikiDoc.DocAttachments) {
                    Document attachmentDoc = new Document(Context);
                    try {
                        attachmentDoc.Uri = Context.GetConfigValue(URI_PARAM_NAME).Replace
(PATHNAME_MARKER_IN_URI, attachment.PathName);
                        attachmentDoc.Data = attachment.Content;
                        document.AddAttachment(attachmentDoc);
                        attachmentDoc = null;
                    } finally {
                        if (attachmentDoc != null) {
                            attachmentDoc.Dispose();
                            attachmentDoc = null;
                        }
                    }
                }

                // Give the document to the indexer.
                IndexDocument(document);
            } catch (ThreadAbortException) {
                // Let go.
            } catch (Exception e) {
                Context.LogCrawlerErrorMessage(e.Message, document.Uri);
                CheckPoint();
            }
        }
    }
}
```

SecurityProvider.cs

As for the WikiEarly class, it contains the security logic. This section only focuses on the differences. All the explanations are not repeated here (for more information, refer to [Wiki Early](#)).

The Initialize() method creates a new instance of Repository, the class that contains the code shared by WikiEarly and WikiLate. The path to the repository is obtained by calling

GetConfigValue("RepositoryPath"). Contrary to WikiEarly, there is not a member WikiSession in this class because ACLs are not added during crawling. Because no groups or users are passed to CES, there is no need to expand Wiki groups.

WikiLateCrawler.SecurityProvider.Initialize() Method

```
public override InitializeError Initialize()
{
    m_Repository = new Repository(GetConfigValue("RepositoryPath"));
    return ErrorOk;
}
```

The Login() and ValidateSession() methods are identical and required for WikiEarly as well as WikiLate. For WikiEarly, these two methods are required as this sample assumes there is no mapping between Wiki and Windows users.

The most interesting part of this class is the Authorize() method. It is the center piece of late-binding: p_UserName and p_SessionData identify the user; p_SessionsData contains the same BLOB returned by Login() and passed to ValidateSession(); p_Uris contains the documents which the method has to determine if the user can access. The result (i.e. the array of Booleans) is returned in p_OutAuthorizations.

At the beginning of the method, a WikiSession is created using the user's login. Then in the loop, for each document, the code tries to access the document by calling GetDocument() on the newly created user session. The document is returned only if the user has the appropriate permissions to access it. That is how impersonation is accomplished with WikiLateAPI. The success/failure is stored as a Boolean in p_OutAuthorizations.

WikiLateCrawler.SecurityProvider.Authorize() Method

```
public override AuthorizeError Authorize(string p_UserName,
                                        byte[] p_SessionData,
                                        string[] p_Uris,
                                        out bool[] p_OutAuthorizations)
{
    AuthorizeError retErr;
    PropTree sessionData = (p_SessionData != null ? PropTree.Parse(p_SessionData) : new
    PropTree());
    CNLAssert.Check(p_UserName == sessionData.GetStringValue("u"));
    try {
        WikiSession wikiSession = new WikiSession(m_Repository, p_UserName,
        sessionData.GetStringValue("p"));

        // No exception thrown, then login is valid.
        string templateUri = GetConfigValue(Crawler.URI_PARAM_NAME);
        int markerPos = templateUri.IndexOf(Crawler.PATHNAME_MARKER_IN_URI);
        p_OutAuthorizations = new bool[p_Uris.Length];
        if (markerPos != -1) {
            int nbCharsAfter = templateUri.Length - (markerPos +
            Crawler.PATHNAME_MARKER_IN_URI.Length);
            for (int uriNo = 0; uriNo < p_Uris.Length; ++uriNo) {
                // Extract the document's pathname from the URI.
                string docName = p_Uris[uriNo].Substring(markerPos, p_Uris[uriNo].Length -
                markerPos - nbCharsAfter);
                // Check if the user can access the document and store the result in
                'p_OutAuthorizations[]'.
                p_OutAuthorizations[uriNo] = (wikiSession.GetDocument(docName) != null);
            }
        }
    }
}
```

```

    }
    retErr = ErrorOk;
  } catch (Exception e) {
    // Login NOT valid.
    p_OutAuthorizations = null;
    retErr = ErrorUnknownFailure(e.Message);
  }
  return retErr;
}

```

Setup Procedures

This section describes the list of actions to perform in the Administration Tool and Interface Editor in order for the security provider to work with your installation of CES. This procedure refers to the WikiEarly and WikiLate samples; however, the procedure is the same for all security providers. To setup both WikiEarly and WikiLate samples, this procedure must be executed twice.

Prerequisite

- CES is already installed;
- CES is functional, meaning the initial setup is done, the license key is entered and an index has been created. The Administration Tool and Search Web pages can be accessed using a Web browser.;
- Both connectors have been compiled and the *WikiEarlyCrawler.dll* and *WikiLateCrawler.dll* DLLs exist;
- An IIS virtual path is pointing on the file system directory containing the document repository (refer to **\$REPOSITORYURL\$** below).

Directories and URLs

- **\$INSTALLDIR\$**: Directory where CES is installed (e.g. *C:\Program Files\Coveo Enterprise Search 5*). This directory normally contains three subdirectories: *Bin*, *Instance* and *Web*.

Note: In the following setup procedure, each time **\$INSTALLDIR\$** is specified, it is important to substitute it with the real path, as it is not done automatically.

- **\$CONNECTORDLL\$**: Complete path and name of the connector DLL (i.e. either *WikiEarlyCrawler.dll* or *WikiLateCrawler.dll*).

Note: In the following setup procedure, each time **\$CONNECTORDLL\$** is specified, it is important to substitute it with the real path, as it is not done automatically.



- **\$REPOSITORYDIR\$**: Root directory of the document repository.

Note: In the following setup procedure, each time **\$REPOSITORYDIR\$** is specified, it is important to substitute it with the real path, as it is not done automatically.

- **\$REPOSITORYURL\$**: IIS virtual path pointing on the document repository.


Note: In the following setup procedure, each time **\$REPOSITORYURL\$** is specified, it is important to substitute it with the real path, as it is not done automatically.

Define a New Security Provider

1. In the Administration Tool, access the **Administrators** page (Configuration > Security – <http://localhost:8080/Admin/Configuration/Security/Security.aspx>).
2. In the left navigation pane, select **Security Providers**. The **Security Providers** page is displayed.
3. Click . The **Modify Security Provider** page is displayed.
4. Click  in the **User Identity** section.
5. Enter the following information in the appropriate fields:

Name	WikiCrawlerSuperUser
User	Crawlersuperuser
Password	p1o2i3u4y5t6r7e8w9q0


This login is defined in the *Security.txt* file which is located in the repository's root directory.

6. Click . The **Security Provider** page is displayed.
7. Enter the following information in the appropriate fields:

Name	WikiEarly Security Provider or WikiLate Security Provider
DLL Path	\$INSTALLDIR\$\Bin\Coveo.CES.CustomCrawlersSecurityProvider.dll
User Identity	Select WikiCrawlerSuperUser from the drop-down list.
Parameters	AssemblyPath="\$CONNECTORDLL\$";RepositoryPath="\$REPOSITORYDIR\$";Uri="\$REPOSITORYURL\$[%PATHNAME]"
Options	<p>For WikiEarly, only select the following checkboxes:</p> <ul style="list-style-type: none"> - Support access list (required to add ACLs on document during crawling); - Support expand group (required in order for ExpandGroup() to be called). <p>For WikiLate, only select the following checkbox:</p> <ul style="list-style-type: none"> - Require authorization (required to enable late-binding and in order for Authorize() to be called).

8. Click .


Define a New Connector

1. In the Administration Tool, access the **Web Connector** page (Configuration > Connectors - <http://localhost:8080/Admin/Configuration/Crawlers/Crawlers.aspx>).
2. In the left navigation pane, select **Additional Connector**. The **Additional Connectors** page is displayed.
3. Click . The **Modify Additional Connector** page is displayed.
4. Enter the following information in the appropriate fields:




Name	WikiEarly Connector or WikiLate Connector
Description	WikiEarly Connector or WikiLate Connector
Assembly Path	\$CONNECTORDLL\$
Type Name	WikiEarlyCrawler.Crawler or WikiLateCrawler.Crawler
Add Parameter	Type: String Name: RepositoryPath

5. Click  **Save** . The **Additional Connectors** page is displayed.
6. Click  **Apply Changes** .

Create a Source and Index the Documents

1. In the Administration Tool, access the **Sources and Collections** page (Index > Sources and Collections). To use the new security provider, a new source must be created. If this source is to be created in a new collection, create it now (for more information, refer to [How to Add a Collection](#)). If not, select the collection in which the source will be added.
2. In the **Sources** section, click  **Add** . The **Add Source** page is displayed.
3. Enter the following information in the appropriate fields:

Name	WikiEarly Source or WikiLate Source
Source Type	WikiEarly Connector or WikiLate Connector
Addresses	abc (the field must not be left empty)
RepositoryPath	\$REPOSITORYDIR\$
Uri	\$REPOSITORYURL\$[%PATHNAME]
User Identity	Select WikiCrawlerSuperUser .

4. Click  **Save** (make sure not to click  **Save and Start**). The **Status** page is displayed.
5. In the left navigation pane, click **Permissions**.
6. Select the **Use a security provider** option.
7. In the **Security Provider** drop-down list, select **WikiEarly Security Provider** or **WikiKate Security Provider**.
8. Click  **Apply Changes** . The source is now ready to index Wiki documents.

Important: Before starting the indexing, it is suggested to open the CES Console in order to view possible error messages.

9. Index the documents.

Configure the Search Interface and Execute Search Queries

1. Open a Web browser and point it to the CES search page (<http://localhost:8080/>).
2. Click **Edit This Interface**.
3. In the Interface Editor, click **Security Providers** at the top of the page.
4. Click **Add New** to add a new security provider.
5. In the **Title** field, enter **WikiEarly** or **WikiLate**.
6. In the **Security Provider** drop-down list, select **WikiEarly Security Provider** or **WikiLate Security Provider**.
7. Select **Automatically Ask To Login**. Selecting this checkbox displays the login page when a user first accesses the Search Interface.
8. Click **OK**, then **Close**.

The browser is now redirected to the Search page; however, because a security provider is now defined, the login page is automatically displayed. For security reasons, the login page only works over a secured connection (HTTPS). Chances are that your Web site is not configured to automatically switch in HTTPS mode. If so, a red message will be displayed in the login page asking you to switch in HTTPS mode. Try changing *http* to *https* in the browser's address bar.

Important: If your IIS supports HTTPS, the red message in the login page should disappear. It is now possible to login and execute search queries. To find a valid login, access the *Security.txt* file which is located in the repository's root directory. If your IIS does not support HTTPS, it is mandatory to install an additional package in IIS. For more information, search *SelfSSL* on the Web.

Advanced Topics

What is an External Security Provider

An External Security Provider is a DLL loaded in the CES kernel (typically in the *CESService*.exe* process, contrary to a connector DLL that is loaded in *CESCustomCrawlers*.exe*). The DLL must export the *InitializeCESSecurityProvider()* function, which returns an instance of a C struct named CGLSPI. That struct contains pointers to functions. It is a low-level API that requires mastering the C language.

Usually, it is required to develop an External Security Provider every time a connector is developed. In order for CES to index documents from an external system, a connector is developed, containing both the crawling and security logics to manage access to those documents. Also, code needs to be shared between the connector and external security provider; therefore, Coveo has developed a wrapper DLL entitled *Coveo.CES.CustomCrawlersSecurityProvider.dll*. That DLL takes care of implementing the low-level C API and redirects the calls to a *.NET* managed DLL. The latter DLL must contain a class that inherits from the abstract class *CrawlerSecurityProvider*. The developer implements the security logic by overriding methods declared in that base class.

This documentation assumes that the External Security Provider is implemented as a *.NET* class that inherits from *CrawlerSecurityProvider*. The *C#* syntax is mainly used in this documentation, but any *.NET* managed language can be used to develop an External Security Provider.

Error Objects and Interfaces

The return value of most of the methods that can be overridden is a return code, or more accurately a reference to an error object. In *CrawlerSecurityProvider*, there are a couple of nested interfaces and classes related to error handling. There is an error interface for each method:

- The InitializeMethod returns a reference to the InitializeError Interface.
- The Login Method returns a reference to the LoginError Interface.
- The ValidateSession Method returns a reference to the ValidateSessionError Interface.
- The Authorize Method returns a reference to the AuthorizeError Interface.
- The ExpandGroup Method returns a reference to the ExpandGroupError Interface.
- The ExpandUser Method returns a reference to the ExpandUserError Interface.

And there are error classes that implement the above interfaces:

- The OkError Class implements the InitializeError, LoginError, ValidateSessionError, AuthorizeError, ExpandGroupError and ExpandUserError interfaces.
- The InvalidateSessionError Class implements the ValidateSessionError interface.
- The BadUserOrPasswordError Class implements the LoginError interface.
- The InvalidGroupError Class implements the ExpandGroupError interface.
- The InvalidUserError Class implements the ExpandUserError interface.
- The NoUserMappingError Class implements the ExpandUserError interface.
- The UnknownFailureError Class implements the InitializeError, LoginError, ValidateSessionError, AuthorizeError, ExpandGroupError and ExpandUserError interfaces.

The same error can be returned from different methods, but not all methods return all the errors. This little class hierarchy is to make sure that only valid errors are returned from each method. If a non-allowed error is returned from a method, it will generate a compile error.

In the base class `CrawlerSecurityProvider`, there is a static property for each possible error. It is a little optimization intended to avoid having to create an instance each time a method is called. For instance, a method should do:

```

    return ErrorOk;
instead of
    return new OkError(); .
  
```

The following displays the list:

- ErrorOk Property
- ErrorInvalidSession Property
- ErrorBadUserOrPassword Property
- ErrorInvalidGroup Property
- ErrorInvalidUser Property
- ErrorNoUserMapping Property
- ErrorUnknownFailure Method

Configuration Values

Reserved Parameters

There are two reserved parameter names: `AssemblyPath` and `EntryPointTypeName`. Both are interpreted by *Coveo.CES.CustomCrawlersSecurityProvider.dll* itself and cannot be retrieved by calling the `GetConfigValue` methods.

- `AssemblyPath` is mandatory and specifies the path name of the combined security provider / connector DLL.
- `EntryPointTypeName` is optional and useful only if more than one class in the DLL inherits from `CrawlerSecurityProvider`. It specifies which one to use.

IConfigValues Interface

`IConfigValues` is an internal interface implemented by the generic layer of external security providers and additional connectors. The main purpose of this interface is to give access to the parameter values that have been set in the Administration Tool when configuring either an external security provider or additional connector. The way to define parameters in the Administration Tool is different depending on whether a security provider or a connector is being configured. This interface gives a polymorphic access to them when implementing a connector.

Multi-Threading

There is only one instance of the derived class that is created by process (usually *CESService*.exe*); however, methods on that instance can be called simultaneously in different threads. If the class contains member and/or static variables, care must be taken to prevent modifications made in one thread to cause processing in other threads to fail. In multi-thread programming, it is common to use locks to serialize access to shared resources and prevent data corruption.