

Coveo Enterprise Search 6.0

Database Connector

Coveo's *Database* connector allows users to crawl and index database content directly through *ODBC* connection and SQL queries. Since all major database vendors provide an *ODBC* connection interface (*Microsoft SQL Server*, *Oracle*, *DB2*, etc.), the *Database* connector will index data from almost any database type.

Features

The following details the features available in the *Database* connector:

- Fully customizable database indexing, either by specifying SQL queries or item types (table names);
- Fully customizable field mapping, from your database fields to *Coveo Enterprise Search* (CES) custom and system fields;
- **Security:** Indexes the custom permissions defined by the user for each object indexed;
- **Live indexing:** Allows the connector to periodically query the database for the latest edits, keeping the index content up to date;
- **Pause/resume:** Allows the connector to be paused and resumed while indexing a database;
- Support of standard authentication to databases (username and password);
- Capability to redirect the documents' hyperlink to an existing Web interface or their respective HTML cached version;
- Support of the most common database drivers (*ODBC* and *OLE DB*).

Requirements

The *Database* connector requires the following in order to work properly:

- A database with *Database* connection capability;
- CES 5.2 and later.

Configuration

▶ How to Index a Database using a Configuration File

To index data from a database, it is possible to use a configuration file. This file is used to instruct the *Database* connector on the way the data must be retrieved and copied from the record fields to the CES system and custom fields.

One of the most important CES system fields is the required URI field, as it is used as the underlying link of a search result. It is important to determine where a user is redirected when a result is opened:

- In the best case scenario, a Web interface is displayed publishing the content of your database; hence, it can be handy to recreate the address of a record's page with some fields of your database.



- In the worst case scenario, it is impossible to redirect the users when they click on a search result. However, if the Open results with cached version option is selected, the document is automatically opened in the *Quick View* mode. For more information, refer to the [How to Index a Database Source](#) section.

The following steps are required to setup a configuration file:

1. Create an XML-formatted configuration file and save it anywhere on the CES server. A best practice is to save it in the configuration folder of CES (default location: *C:\CESx\Config*). It is possible to create it from scratch or from the sample provided in [Appendix C](#).
2. Create a complete *Database* source and specify the path of the configuration file with the **Configuration File Path** source parameter. For more information, refer to the [How to Index a Database Source](#) section.

▶ How to Index a Database using a View

It is possible to index data from a database using embedded views (queries). These views are defined directly in your database. The following steps are required to use views:

1. Create an *Database* source as described in the [How to Index a Database Source](#) section.
2. In your database, create a view (query) that returns at least the *URI* field. The value of the *URI* field must be a valid URI, meaning it must have any scheme prefix followed by a unique identifier (e.g. *odbc://id=4*). It does not have to be browsable or have any special signification. The fields that can be mapped to a document are: *Uri*, *UriCaption*, *ClickableUri*, *PrintableUri*, *Title*, *FileName*, *ContentType*, *BinaryBody*, *Body*, *ModifiedDate* and all security-related fields. For more information, refer to **Appendix D**.
3. Define the required security permissions on your documents using one of the following methods:
 - In the Administration Tool:
 - a. Access the **Sources and Collections** page (Index > Sources and Collections).
 - b. In the **Source** section, expand the drop-down list of the appropriate source.
 - c. Select **Edit Permissions**. The **Permissions** page corresponding to the source is displayed.
 - d. In the **Permissions** section, select **Specify the security permissions to index**.
 - e. Specify the necessary users in the **Allowed Users** and **Denied Users** lists. To add a new user, enter the username in the corresponding text box and click  **Add**. By default, the **Allowed Users** list contains everyone, meaning that everybody has access to the indexed documents. Note that no configuration file is required with this method.
 - f. Click  **Apply Changes**.
 - From the created View :
 - a. In the created View, add new fields containing the required permissions for every document. Note that no configuration file is required with this method.

| Security type | Required Field Names | |
|---------------------|----------------------|----------------------|
| | User-related | Server-related |
| Windows | | |
| -allowed users: | AllowedWindowsName | AllowedWindowsServer |
| -denied users: | DeniedWindowsName | DeniedWindowsServer |
| Custom group | | |
| -allowed users: | AllowedCGName | AllowedCGServer |
| -denied users: | DeniedCGName | DeniedCGServer |
| Custom user | | |
| -allowed users: | AllowedCUName | AllowedCUServer |
| -denied users: | DeniedCUName | DeniedCUServer |

- b. In the Administration Tool, access the **Sources and Collections** page (Index > Sources and Collections).
 - c. In the **Sources** section, expand the drop-down list of the appropriate source.
 - d. Select **Edit Permissions**. The **Permissions** page corresponding to the source is displayed.
 - e. In the **Permissions** section, select Index security permissions.
- o From a configuration file:
 - a. Create a configuration file as described in the [How to Index a Database using a Configuration File](#) section. By doing so, you are able to add an *AllowedUsers* node in which it is possible to define allowed and denied users for every document. This also implies that query expressions have to select the entire view or the required fields from it.

▶ How to Index a Database Source

1. In the Administration Tool, access the **Sources and Collections** page (Index > Sources and Collections).
2. In the **Sources** section, click  **Add**. The **Add Source** page is displayed.
3. Enter the following values in the appropriate fields in order to index your database:

| Field | Description | Example |
|--------------------------|--|--|
| Name | Any descriptive name. | Employees DB Source |
| Source Type | The connector used by this source. | Database |
| Addresses | The connection string used to connect to the database. | Each type of database has a different syntax for the connection string. To learn more about the several connection string syntaxes, refer to http://www.connectionstrings.com/ . |
| Configuration File Path* | The full path to the configuration file used by the connector. | C:\CESx\Config\odbc_config.xml. |
| Items to Crawl | The database tables/views to index. | A comma-separated list of object names. An object is a table or view. Unless the table to index has a field named URI, it is recommended to design a database View to output field names that match CES required fields. Note that this parameter acts as a filter on objects defined in the configuration file, as the connector filters out unlisted items from the query. |

| | | |
|-----------------------------------|--|---|
| Driver Type | The software driver (or provider) that provides access to the database. | The default value is Odbc, meaning that the connector connects to the database using the default <i>ODBC</i> driver, given that it provides one. Other possible driver types are: OleDb (to use an OLEDB driver) and SqlClient (to use a MSSQL driver). |
| Command Timeout | The maximum time allowed to perform a query on the database. | The default value is 60 seconds. Once elapsed, the query times out. |
| Open results with cached version* | Whether opened search results are displayed using their cached version in the Quick View panel or not. | This option is not selected by default. Select it if you do not have a specific URI to redirect users to when a search result is clicked. |

* These parameters are optional.

4. Click  [Save and Start](#).

| | |
|--------------------------------|--|
| Name | Employees DB Source ? |
| Source Type | Database |
| Addresses | <pre>Driver={Microsoft Access Driver (*.mdb)};Dbq=C:\odbc\northwind.mdb;Uid=Admin;Pwd=;</pre> <p>Depends on the additional connector used. One entry per line.</p> |
| Rating | Normal ? |
| Document Types | Default ? |
| Fields | Default Scheme ? |
| Refresh Schedule | Every day ? |
| Items to crawl | Employees ? |
| Configuration File Path | C:\CES6\config\odbc_config.xml ? |
| Command Timeout | 60000 ? |
| Driver Type | Odbc ? |
| Option | <input checked="" type="checkbox"/> Index subfolders ? <input checked="" type="checkbox"/> Index the document's metadata ? <input type="checkbox"/> Document's addresses are case-sensitive ? <input checked="" type="checkbox"/> Generate a cached HTML version of indexed documents ? <input type="checkbox"/> Open results with cached version ? |
| Authentication | User Identity (none) ? |

▶ How to Enable LiveIndexing on a Database Source

Live indexing keeps documents up-to-date by scanning repositories and re-indexing modified documents at short intervals. To enable this feature on a source, database items that have to be indexed, such as tables or views, must contain a Date type field to indicate their latest modification date. Fields can be given any name as long as it has the right data type. Furthermore, whenever a record is modified, it is important to update its latest modification date field. For more information concerning configuration files, refer to [Appendix A](#).

To enable Live Indexing, perform the following step:

- In the SQL query, the SELECT statement must have a WHERE clause with a criterion on the last modification date field.

```
WHERE ITEM.LAST_MODIFICATION_DATE_FIELD >= ?
```

The question mark operator (?) in the previous example is used to pass a parameter to the SQL query. The *Database* connector automatically replaces it with the last refresh date of the current database item.

However, specifically and only for MSSQL/SQLServer databases (SqlClient driver type), it is important to replace the question mark sign by the @LastRefresh expression.

```
WHERE ITEM.LAST_MODIFICATION_DATE_FIELD >= @LastRefresh
```

▶ How to Enable Pause/Resume on a Database Source

In order for a source to support Pause/Resume, the following additions are necessary in your configuration file:

- In the SQL query, the SELECT statement must have an ORDER BY on the chronological field.
 - **If the source supports Live Indexing ():** The same chronological field must also be used in the ORDER BY (OrderDate in the example below).
 - **If the source does not support Live Indexing:** A field must be selected. Ideally, the field should identify uniquely each record. If the primary key contains only one field, that field should be selected. However, if there is a column that contains a sequential number incremented by the DBMS each time a new record is added, that column is also a good choice.

If no field exists that uniquely identifies each record, any field can be used. However, during a Resume, some records that have been indexed before pausing can probably be reindexed. For example, if we have a table containing an integer column, which contains values shared by many records. Also, 1,000 records have the value 123 for that column and we pause the crawling after 500 records. If crawling is resumed, the 500 records are reindexed, which is not necessary. However, records with values less than 123 are not reindexed.
- Once the chronological field is selected, some XML attributes must be added on the Accessor element in the configuration file:
 - *OrderByFieldName* specifies the name of the column on which the *ORDER BY* is applied. This attribute must be present to enable *Pause/Resume*.
 - *OrderByFieldType* specifies the .NET data type of that column. This attribute is not normally required. The *Database* connector automatically tries to determine the data type by preparing the SQL query without executing it and looking at the schema of the results. However, if a specific DBMS does not handle that process correctly, one can manually specify the data type with this attribute. The following lists the allowed types:

- short (16-bit signed integer)
 - ushort (16-bit unsigned integer)
 - int (32-bit signed integer)
 - uint (32-bit unsigned integer)
 - long (64-bit signed integer)
 - ulong (64-bit unsigned integer)
 - float (single-precision floating point number)
 - double (double-precision floating point number)
 - string (string)
 - DateTime (temporal value corresponding to a day and the time in that day)
- To support both Live Indexing and Pause/Resume, *LiveIndexingFieldName* must contain the same name as *OrderByFieldName*.

For a configuration file excerpt for a source with Pause/Resume and LiveIndexing enabled, refer to:

```
<Mapping type="Orders">
  <Accessor type="query"
    OrderByFieldName="OrderDate"
    OrderByFieldType="DateTime"
    LiveIndexingFieldName="OrderDate">
    SELECT Shippers.CompanyName AS ShipperName,
    Orders.OrderID AS ID,
    Orders.CustomerID,
    Orders.OrderDate,
    Orders.RequiredDate,
    Orders.ShippedDate,
    Customers.CompanyName,
    Employees.LastName,
    Employees.FirstName
  FROM Orders, Shippers, Customers, Employees
  WHERE Orders.ShipVia = Shippers.ShipperID AND
    Orders.CustomerID = Customers.CustomerID AND
    Orders.EmployeeID = Employees.EmployeeID AND
    Orders.OrderDate >= ?
  ORDER BY Orders.OrderDate
  </Accessor>
</Mapping>
```

Appendix A

► Configuration File Reference

This section explains how to write an XML configuration file in order to index data from a database. Basically, a single mandatory section must be included in a configuration file: the *Mapping* node, which must be placed under a parent *Database* node. Note that the *CommonMapping* node is optional. The following displays this basic structure:

```
<?xml version="1.0" encoding="utf-8" ?>
<ODBC>
  <CommonMapping>
  </CommonMapping>
  <Mapping>
  </Mapping>
</ODBC>
```

<CommonMapping> section

This section defines fields that should be mapped for more than one mapping. If you plan to index more than one object from your source, this section allows to define identical fields for all mappings. The following displays an overview of the <Commonfields> section:

```
<CommonMappings excludedItems="ItemType">
</CommonMappings>
```

<CommonFields> section

This node represents a Fields node, as in the mapping section, that is added to all mappings except the ones specified in the excludedItems list. This list is made of comma separated values:

```
<CommonFields excludedItems="ItemType">
</CommonFields>
```

<Mapping> section

This section defines the mapping between the fields of the ODBC object to index and the fields in CES that hold this information. It can be duplicated as many times as necessary, each occurrence means a new type of item to index. A mapping of an accessor and collection of fields is made, as well as a collection of allowed users. The code below displays an overview of the mapping section.

Note: All node values can be defined using this syntax: *%[odbcField]*. When the indexing process starts, this value is replaced by the actual value from the *Database* source (e.g., in *<FileName>%[CustomerID].txt</FileName>*, *%[CustomerID]* is dynamic and replaced by the source value; however, *.txt* is static and identical for each document found in the source.

```
<Mapping type="ItemType">
  <Accessor type="AccessorType"></Accessor>
  <Fields>
    <Uri></Uri>
    <UriCaption></UriCaption>
    <ClickableUri></ClickableUri>
    <PrintableUri></PrintableUri>
    <OriginalUri></OriginalUri>
    <FileName></FileName>
    <Title></Title>
    <Body></Body>
    <ModifiedDate></ModifiedDate>
    <ContentType></ContentType>
    <CustomFields>
      <CustomField name="FieldName"></CustomField>
    </CustomFields>
  </Fields>
  <AllowedUsers>
    <AllowedUser type="SecurityType" allowed="isAllowed">
      <Name></Name>
      <Server></Server>
    </AllowedUser>
  </AllowedUsers>
</Mapping>
```

<Mapping type="ItemType">

Each *Mapping* node must have a *type* attribute. This attribute represents the name of this mapping and is the value used to reference a mapping.

<Accessor type="AccessorType">

The *accessor* of a mapping represents how to retrieve the required information to be indexed. This node is mandatory in order to have a valid mapping. The *type* attribute of an accessor is mandatory and can have two possible values: object or query. Depending on the value of the *type* attribute, the value of the accessor node is different:

- **Object:** It is recommended to provide the name of an object from your source, a table or a view in the case of a SQL source.
- **Query:** It is recommended to provide a SQL query that returns every field to index in CES.

<BinaryBody>

This node is used to map a BLOB field from the database to the document's body. Exceptionally, the value specified for this node must not use the standard syntax `%[odbcField]`. Instead, use the syntax `odbcField`. Once the mapping is resolved, an appropriate converter is called depending on the type of file inside the BLOB field of the database. Note that you cannot use both *BinaryBody* and *Body* (below) nodes at the same time.

<Fields>

This node represents a collection of fields to be mapped when a document is indexed. The following lists the nodes it can contain: Uri, UriCaption, ClickableUri, PrintableUri, OriginalUri, FileName, Title, Body, ModifiedDate, ContentType and CustomFields.

<Uri>

This node represents the URI of a document. This node is mandatory in order to index documents. The mapping syntax should be used to define the mapping of this field.

<ClickableUri>

This node represents the URI opened when trying to open documents from the user interface of CES.

<PrintableUri>

This node represents the URI displayed in the UI of CES.

<FileName>

This node represents the name of the file indexed. The extension part of the filename is used to select the appropriate converter. Either this node or *<ContentType>* must be specified, as it is used to find the appropriate converter. The mapping syntax should be used to define the mapping of this field.

<ContentType>

This node represents the type of data in the document's body. Either this node or *<FileName>* must be specified, as it is used to find the appropriate converter. If you are not sure of the type of data being indexed, use the *binarydata* value and CES will find the right converter for your document data. The mapping syntax should be used to define the mapping of this field.

Recommended: For database records without an actual filename, do not specify the *<FileName>* node; however, do specify the *<ContentType>text/html</ContentType>* node.

<Title>

This node represents the title of the document indexed. The mapping syntax should be used to define the mapping of this field.

<Body>

This node represents the body of the document indexed; the body can be textual or binary data (e.g. it is possible to specify a BLOB field name). The mapping syntax should be used to define the mapping of this field. Notice that you cannot use both *Body* and *BinaryBody* nodes at the same time.

<ModifiedDate>

This node represents the last modification date on a document; hence, the value of this field must be a date. The mapping syntax should be used to define the mapping of this field.

<CustomFields>

This node allows the mapping of *ODBC* fields to CES custom fields. This node is made of a collection of *<CustomField>* nodes.

<CustomField name="FieldName">

This node represents a mapping of an *ODBC* field to a CES custom field. The name attribute is mandatory and represents the name of the CES custom field to bind data to. The value of this node should use the mapping syntax.

System fields can be overridden by creating a custom field named after the system field's name (e.g. *<CustomField name="sysauthor">%[Author]<CustomField>* would override the document's author with data from the source).

<AllowedUsers>

This node allows the mapping of *ODBC* fields to CES security. This node is made of a collection of *<AllowedUser>* nodes. This section is optional; however, note that if this section is missing, everybody has access to all the documents indexed.

<AllowedUser type="SecurityType" allowed="isAllowed">

This node represents the rights of a user or group on indexed documents. The two attributes—*type* and *allowed*—of this node are mandatory. The **type** attribute can be one of the following values: *Windows*, *CustomGroup* and *CustomUser*. The **allowed** attribute value can be *True* or *False*.

Two nodes can be defined inside of an *AllowedUser* node—*Name* and *Server*. These nodes should contain the value to use to create the security. The mapping syntax should be used if you need to use data from the *Database* source as security rights on indexed documents. Note that if these nodes are missing, everybody has access to all the documents indexed.

Notice that it is possible to define multiple allowed users within a single *AllowedUser* node. To do so, separate the values by a semicolon (;).

Appendix B

▶ NorthWind – Advanced Sample Tour

This section is a detailed explanation of the configuration file found in **Appendix C**. This file can be used by the *Database* connector to index data from the *NorthWind* database, which is a sample database part of *Microsoft Access* and *Microsoft SQL Server* products.

<CommonMapping>

In *<CommonMapping>*, it is possible to specify various settings common to all or several of the mappings in the configuration file. Among these settings, the *<CustomFields>* and *<AllowedUsers>* nodes used for all tables can be added, unless specified otherwise. For more information concerning these fields, refer to the *<Mapping>* section. The following is an example of *<CommonMapping>*:

```
<CommonMapping excludedItems="Customers">
```

The *excludedItems* attribute lists all the objects from the *ItemType* source parameter that do not use the settings specified in *<CommonMapping>*. In the current example, the table *Customers* is excluded. The following is a listing of the entire *<CommonMapping>* nodes:

```
<CommonMapping excludedItems="Customers">
  <Fields>
    <CustomFields>
      <CustomField name="ID">%[ID]</CustomField>
    </CustomFields>
  </Fields>
  <AllowedUsers>
    <AllowedUser type="Windows" allowed="true">
      <Name>everyone</Name>
      <Server></Server>
    </AllowedUser>
  </AllowedUsers>
</CommonMapping>
```

<Mapping>

The *<Mapping>* node defines how the connector retrieves data from tables, how this indexed data is stored and who has access to it. It has an attribute called *type*. In this node, it is important to add the name of the table used for mapping. For this example, the mapping for the table *Orders* is displayed. Two tables are indexed, however this one contains the most complex configuration:

```
<Mapping type="Orders">
  <Accessor type="query">
    SELECT Shippers.CompanyName AS ShipperName, Orders.OrderID AS ID,
    Orders.CustomerID, Orders.OrderDate, Orders.RequiredDate, Orders.ShippedDate,
    Customers.CompanyName, Employees.LastName, Employees.FirstName
    FROM Orders INNER JOIN
    Shippers ON Orders.ShipVia = Shippers.ShipperID INNER JOIN
    Customers ON Orders.CustomerID = Customers.CustomerID INNER JOIN
    Employees ON Orders.EmployeeID = Employees.EmployeeID
  </Accessor>
  <Fields>
    <Uri>http://www.coveo.com/Orders/details.aspx?Id=%[ID]</Uri>
    <ClickableUri>http://www.coveo.com</ClickableUri>
    <FileName>%[ID].txt</FileName>
    <Title>%[ID]</Title>
    <Body>OrderDate: %[OrderDate] \n RequiredDate: %[RequiredDate] \n ShippedDate:
    %[ShippedDate] \nShipped via: %[ShipperName]
  </Body>
    <CustomFields>
      <CustomField name="Type">Order</CustomField>
      <CustomField name="OrderDate">%[OrderDate]</CustomField>
      <CustomField name="RequiredDate">%[RequiredDate]</CustomField>
      <CustomField name="ShippedDate">%[ShippedDate]</CustomField>
      <CustomField name="Shipper">%[ShipperName]</CustomField>
      <CustomField name="sysAuthor">%[FirstName] %[LastName]</CustomField>
    </CustomFields>
  </Fields>
  <AllowedUsers>
```

```

    <AllowedUser type="CustomGroup" allowed="true">
      <Name>%[FirstName] %[LastName]</Name>
      <Server></Server>
    </AllowedUser>
  </AllowedUsers>
</Mapping>

```

<Accessor>

This is where the way to access the table is specified. It has an attribute called *type* that can either be equal to *object* or *query*. In our example, the *<Accessor>* looks like this:

```
<Accessor type="query">
```

For this example, the query type is used, meaning that the information stored in the database is accessed using an SQL query and can be stored in multiple tables and views. The *<Accessor>* node contains the query string used to extract the data from the table. The query string in this example is:

```

SELECT Shippers.CompanyName AS ShipperName, Orders.OrderID AS ID, Orders.CustomerID,
Orders.OrderDate, Orders.RequiredDate, Orders.ShippedDate, Customers.CompanyName,
Employees.LastName, Employees.FirstName
FROM Orders INNER JOIN
Shippers ON Orders.ShipVia = Shippers.ShipperID INNER JOIN
Customers ON Orders.CustomerID = Customers.CustomerID INNER JOIN
Employees ON Orders.EmployeeID = Employees.EmployeeID

```

<Fields>

This is where the *Database* connector indexing the content read from the database is configured.

<Uri>

```
<Uri>http://www.coveo.com/Orders/details.aspx?Id=%[ID]</Uri>
```

When a user clicks on the title of a search result, the address to which he is redirected is called a URI. It is also used by the index to identify documents.

Note: Even though the database records indexed do not have an actual Web address where they can be viewed by users, it is important to provide one. It is possible to generate an URI based on any dummy address, as long as it is unique. It is suggested to create one formed with a primary key provided from your database.

<ClickableUri>

<ClickableUri> specifies how to access the data from the search results page.

```
<ClickableUri>http://www.coveo.com</ClickableUri>
```

In the previous example, the clickable URI points to <http://www.coveo.com>, but in a production environment it may point to a Web page that displays the content of the appropriate results.

<FileName>

<FileName> is the name of the file to create holding the SQL query information. This way, the information can be indexed as a document:

```
<FileName>%[ID].txt</FileName>
```

<Title>

<Title> is the name displayed on the search result page to represent the entry in the table.

```
<Title>%[ID]</Title>
```

Once again, *%[ID]* is used to give each entry its own name based on the ID field in the *Orders* table.

<Body>

<Body> is a way to represent the content of the entry. It can be a mix of static text and dynamic content taken from the table.

```
<Body>OrderDate: %[OrderDate] \n RequiredDate: %[RequiredDate] \n ShippedDate:
%[ShippedDate] \nShipped via: %[ShipperName]
</Body>
```

In the previous example, we have:

```
OrderDate: %[OrderDate] \n RequiredDate: %[RequiredDate] \n ShippedDate: %[ShippedDate]
\n Shipped via: %[ShipperName]
```

%[OrderDate], *%[RequiredDate]*, *%[ShippedDate]* and *%[ShipperName]* fields are all dynamic content types that change with each entry taken from the table. When \n is displayed, it means that what comes after is printed on a new line.

<CustomFields>

This is where we link the entries taken from the table to the custom fields of CES.

<CustomField>

Each <CustomField> node represents a custom field in CES and the data contained therein. In the following example, several <CustomField> nodes are displayed:

```
<CustomField name="OrderDate">%[OrderDate]</CustomField>
```

The *name* attribute is the name of the custom field that CES uses to store the data. The *%[OrderDate]* expression instructs the connector to copy the information from the *OrderDate* database field to the *OrderDate* custom field in CES.

<AllowedUsers>

Since the *Database* connector does not index the securities of the database automatically, this field can be used to protect the data taken from the database.

<AllowedUser>

In this node, rights are given to users to access the current table. Refer to the *AllowedUser* in the following example:

```
<AllowedUser type="CustomGroup" allowed="true">
```

There are two attributes used in *AllowedUser*. The first one is *type*—it specifies the type of user to which rights are given—*Windows*, *CustomGroup* or *CustomUser*. In the previous example, it is *CustomGroup*. The second attribute is *allowed*—it can be set to *True* or *False*. In the previous example, it is set to *True*, meaning access is granted.

<Name>

In the <AllowedUser> node, it is possible to specify the name of a user or group to which you want to grant permissions. In the following example:

```
<Name>%[FirstName] %[LastName]</Name>
```

The *FirstName* and *LastName* values are used and stored in the table. This type of setup is useful only if these fields correspond to users that have access to the database. A best practice would be to have one or more tables that list the users as well as their respective permissions and to use this information. Another solution is to provide the name of a *Windows* group that contains all the users that should have access. Otherwise, this field can be set to *Everyone*, meaning everyone could have access to the information taken from the table.

<Server>

<Server> specifies the domain name for the current group or user. In the current example, no server name is specified. However, in an everyday context, this field needs to be set a specific value.

Appendix C

► Configuration Files – A Working Example (NorthWind)

```
<?xml version="1.0" encoding="utf-8" ?>
<ODBC>
  <CommonMapping excludedItems="Customers">
    <AllowedUsers>
      <AllowedUser type="Windows" allowed="true">
        <Name>everyone</Name>
        <Server></Server>
      </AllowedUser>
    </AllowedUsers>
  </CommonMapping>
  <Mapping type="Orders">
    <Accessor type="query">
      SELECT Shippers.CompanyName AS ShipperName,
             Orders.OrderID AS ID,
             Orders.CustomerID,
             Orders.OrderDate,
             Orders.RequiredDate,
             Orders.ShippedDate,
             Customers.CompanyName,
             Employees.LastName,
             Employees.FirstName
      FROM   Orders, Shippers, Customers, Employees
      WHERE  Orders.ShipVia = Shippers.ShipperID AND
             Orders.CustomerID = Customers.CustomerID AND
             Orders.EmployeeID = Employees.EmployeeID AND
             Orders.OrderDate >= ?
    </Accessor>
    <Fields>
      <Uri>http://www.coveo.com/Orders/details.aspx?Id=%[ID]</Uri>
      <ClickableUri>http://www.coveo.com</ClickableUri>
      <FileName>%[ID].txt</FileName>
      <Title>%[ID]</Title>
      <ModifiedDate>%[OrderDate]</ModifiedDate>
      <Body>
        OrderDate: %[OrderDate]<br/>
        RequiredDate: %[RequiredDate]<br/>
        ShippedDate: %[ShippedDate]<br/>
        Shipped via: %[ShipperName]
      </Body>
      <CustomFields>
        <CustomField name="Type">Order</CustomField>
        <CustomField name="OrderDate">%[OrderDate]</CustomField>
        <CustomField name="RequiredDate">%[RequiredDate]</CustomField>
        <CustomField name="Shipper">%[ShipperName]</CustomField>
        <CustomField name="sysAuthor">%[FirstName] %[LastName]</CustomField>
      </CustomFields>
    </Fields>
    <AllowedUsers>
      <AllowedUser type="CustomGroup" allowed="true">
        <Name>%[FirstName] %[LastName]</Name>
        <Server></Server>
      </AllowedUser>
    </AllowedUsers>
  </Mapping>
</ODBC>
```

```

    </AllowedUser>
  </AllowedUsers>
</Mapping>
<Mapping type="Customers">
  <Accessor type="object">Customers</Accessor>
  <Fields>
    <Uri> http://www.coveo.com/Customers/details.aspx?Id= %[CustomerID]</Uri>
    <ClickableUri>http://www.coveo.com</ClickableUri>
    <ContentType>text/html</ContentType>
    <Title>%[CompanyName] ([CustomerID])</Title>
    <Body>%[CompanyName] \n %[ContactTitle] %[ContactName] %[Address]</Body>
    <CustomFields>
      <CustomField name="Type">Customer</CustomField>
      <CustomField name="ID">%[CustomerID]</CustomField>
    </CustomFields>
  </Fields>
</Mapping>
</ODBC>

```

Appendix D

▶ How to Create a View in a Database

It is possible to exploit the principle of embedded views (queries) in a database. By doing so, you can avoid rewriting long, complex queries in a configuration file. However, if you have to use a configuration file, your query expression can be used to filter out the objects you do not want from an existing View.

To create a view in a database:

Log on to your database with a database editor/client. Make sure you have the appropriate read/write rights or administrator privileges to the database.

On the client's command-line, enter the necessary expression to create your View. To do so, refer to one of the samples provided below (based on the *NorthWind* sample database). If you are using an Access-like database editor, create a View using only the *Select* and *From* parts of the queries provided below. In the first example, security permissions for documents should be defined using one of the methods explained in the section *How to Index a Database Using a View*. In the second example, permissions are defined directly as fields inside the view (allowed user for all documents is *coveo\hford*).

CES_Orders

```

CREATE view "CES_Orders" AS
SELECT
  'http://www.coveo.com/Orders/details.aspx?Id=' + convert(varchar, Orders.OrderID) AS
  Uri,
  'http://www.coveo.com' AS ClickableUri,
  Orders.OrderID AS Title,
  'OrderDate: ' + convert(varchar, OrderDate) + CHAR(10) + 'RequiredDate: ' +
  convert(varchar, RequiredDate) + CHAR(10) + 'ShippedDate: ' + convert(varchar,
  ShippedDate) + CHAR(10) + 'Shipped via: ' + Shippers.CompanyName AS Body,
  'Order' AS Type,
  OrderDate,
  RequiredDate,
  ShippedDate,
  Shippers.CompanyName AS Shipper,
  FirstName + ' ' + LastName AS sysAuthor
FROM
  Orders

```

```

INNER JOIN Shippers ON Orders.ShipVia = Shippers.ShipperID
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
  
```

CES_Products

```

CREATE view "CES_Products" AS
SELECT
  'http://www.coveo.com/Products/details.aspx?Id=' + convert(varchar,
Products.ProductID) AS Uri,
  'http://www.coveo.com' AS ClickableUri,
  ProductName + ' (' + convert(varchar, Products.ProductID )+ ')' AS Title,
  'Name: ' + ProductName + CHAR(10) + 'Category: ' + CategoryName + CHAR(10) +
'Supplier: ' + Suppliers.CompanyName AS Body,
  'Product' AS Type,
  ProductName AS Product,
  Discontinued,
  UnitPrice,
  QuantityPerUnit,
  CategoryName AS Category,
  Suppliers.CompanyName AS Supplier
  'hford' AS AllowedWindowsName,
  'coveo' AS AllowedWindowsServer
FROM
  Categories
  INNER JOIN Products ON Categories.CategoryID = Products.CategoryID
  INNER JOIN Suppliers ON Products.SupplierID = Suppliers.SupplierID
  
```