Coveo .NET Front-End 12.0 for Developers

.NET Search UI Home

The Coveo .NET Front-End software comes with the Coveo Platform, and provides out-of-the-box ASP.NET controls to render a search user interface (see Installing Coveo .NET Front-End).

The Coveo .NET search user interface is designed to enable extensive customization, therefore responding to the ever-growing needs of the customers. To customize the search interface, you can either use the Interface Editor to modify options that affect the behavior of the built-in search controls, or you can even create a new skin according to your specific needs (see Search Interfaces and Skins).

The Interface Editor (**Do more** menu > **Edit this Interface**) is certainly the easiest way to customize the search interface (see Interface Editor). However, because it is a GUI tool, there are certain limitations. This approach is strongly recommended to enable or disable standard features, manage facets, change colors, etc.

On the other hand, if more complex modifications are required, such as adding or removing controls, modifying the global layout or the behavior of the search interface using custom code, it is recommended to create a custom skin (see Search Interfaces and Skins) and create custom search controls in order to implement the desired features which are not available out-of-the-box from the .NET Search UI.

This section of the documentation describes how to customize the search interface using the second approach.

Even if it is possible to customize the search interface using the two approaches simultaneously, it is recommended to use only one of them at a time, as the two may conflict in some circumstances. Therefore, to perform advanced customizations, use the second approach from the start.

Everything that can be done through the Interface Editor can also be done directly from within a custom skin.

.NET Search UI Concepts

The .NET search UI is composed of several modular ASP.NET controls that interact together to provide the different features available to the user. These controls are referenced in a set of user control files called skins (see Search Interfaces and Skins). Using skins makes it easy to add, extend or remove features from the interface, as this can usually be done by adding or removing a control from the skin. The framework is fully extensible, meaning that it is possible to author your own controls, that integrate in the same manner in a custom search interface.

The following table lists examples of search interface controls.

Control	Description
`Query`	Allows the user to enter the search keywords.
`SearchButton`	Renders the button used to perform new searches.
`ResultTitle`, `ResultExcerpt`, and `ResultPrintableUri`	Output the title, the excerpt and the address (URI) of a query result (those are typically included in a repeated template).
`Pager`	Allows users to browse the various query result pages.
`CustomRefineByFields`	Renders a toolbar that contains the facets configured using the Interface Editor.

All the interactions between the various controls making up a search interface are controlled by the `SearchControl` control (see Search Interface).

The complete .NET search UI can be added to any ASP.NET page by adding a single special control called `SearchHub`. This control loads the appropriate search interfaces, skins and settings into the page (see Loading the .NET Search UI).

Loading the .NET Search UI

The Coveo .NET Search UI is usually added to an ASP.NET page through the `SearchHub` control. This control is responsible of loading the rest of the controls that make up the search UI.

Example:

In the default search page file:

```
[.Net_Front-End_Path]\Web\default.aspx
```

The search controls are added to the web page by the following line:

```
<ces:SearchHub id="h" runat="server" />
```

A search hub is defined as a collection of search interfaces. For example:

- All Content
- My Emails

- My Files
- Intranet
- Etc.

With the out-of-the-box skins, the search interfaces are listed in the black bar at the top of the search page. The user can switch from one to another by clicking on their names in that top black bar. As many search hubs as needed can be created with the Interface Editor. Each search hub can have a list of search interfaces that is different from the others.

At the initial display of the search page in the browser, the selection of the search hub to show by default, and the search interface in it, is done by following certain steps:

1. During the `Init` stage, the `SearchHub` control first determines which search hub to load on the page. The easiest way to configure this is to set its `Name` property to the name of the desired search hub. Otherwise, the `SearchHub` control looks for a name through the query string (`sh=`*`somehubname`* parameter). If none is found, the **Default** search hub is used.
2. When the search hub to use is selected, the `SearchHub` control then determines which search interface to show. It looks for a name specified through the query string (`sk=`*`someinterfacename`* parameter), and attempts to use the **Per Uri Settings** to match a search interface (see Adding a Per URI Setting With the .NET Interface Editor). If none is found, the first search interface is used by default.
3. The `SearchInterface` control then loads the `InterfaceSettings` object for the selected search interface. That object holds all the search interface settings (such as the display styles, the facets, etc.), and can be edited using the Interface Editor.
4. Using the `Skin` property of the `InterfaceSettings` object, the `SearchInterface` then loads the proper `CoveoSearch.ascx` file in the page from the skin sub folder under `[.NET_Front-End_Path]\Web\Coveo\Skins`. That user control is the root control of the skin. It inherits from `SearchControl`. It loads in turn all the other components required by the search interface. See the Structure of a Skin topic for more details.

Coveo Index Server

The Coveo index server to which the search controls connect is specified in the IIS site's `web.config` file. Even if they can be modified by opening the file with a text editor, they are usually configured with the Coveo .NET Front-End First Time Setup page.

Difference between 6.5 and 12.0

In Coveo Enterprise Search 6.5, in addition to loading the search interface controls as described above i.e. by adding a line like this one:

```
<ces:SearchHub id="h" runat="server" />
```

It was also possible to load a specific search interface (e.g. My Emails) by inserting a SearchInterface control instead in an ASPX/ASCX file:

```
<ces:SearchInterface Name="Email" id="si" runat="server" />
```

In Coveo Front-End 12.0, this is no longer supported.

Search Interfaces and Skins

Skin designates a set of user control files that define the structure and the layout of a search interface. By default, the .NET Search UI comes with a few built-in skins: **Default** (a.k.a. **All Content**), **My Emails**, **My Files**, **People**, **Intranet**, etc. SharePoint skins can also be installed if the integration to Microsoft SharePoint is enabled.

Skins must not be confused with Search Interfaces. A search interface acts as a container for settings, such as display styles and configuration settings. One of these settings is the name of the skin used by the search interface. There can be several search interfaces bound to a single skin, much in the same way than many search pages/hubs can include a given search interface. In other words:

- A search page includes a `SearchHub` control.
- A `SearchHub` control instantiates `SearchInterface` controls that are added as child controls.
- A `SearchInterface` control provides settings such as styles, facets, preference defaults, etc. These settings are contained in an instance of the `InterfaceSettings` class.
- A skin provides the structure and the layout of the rendered search interface, and is associated to it via its `Skin` property.

Search Interface

A search interface is a user interface unit. It generally contains a query box, shows the results returned by the query, and displays other user interface elements like facets.

A search hub generally contains a few search interfaces. Using the default skins, the search interfaces are listed in the top bar (e.g. **All Content**, **My Emails**, **Intranet**, etc.). The user can switch from one to another by clicking on the desired search interface in the top bar.

A search interface usually comes with its own skin, and therefore its skin directory under `[.Net_Front-End_Path]\Web\Coveo\Skins`. The main user control of a skin is always a `SearchControl` control, which is defined in the `CoveoSearch.ascx` file of the skin. The `SearchContr`

`ol` control is responsible of synchronizing the interactions between all the controls of the search interface. It also keeps track of the current query and fires the various events that cause other controls to perform various tasks. There is always only one main `SearchControl` instance per search interface. Also note that it is possible to instantiate more than one `SearchHub` control on the same page. Each `SearchHub` control can contain only one or a few search interfaces.

A single search interface can display results from several queries, therefore making it possible to search several kinds of items from the same UI (for example, combining a people search feature with a more general document search). Every query is performed by the `QueryControl` control. When the user performs a search, each `QueryControl` uses the global query (provided by `SearchControl`), adds additional information (for example, restricting to people results), and sends the query to the server in order to retrieve the results, which are then displayed.

Search Interface Settings

Search interface settings are a set of options that specifies:

- Basic information about a search interface, such as its name and its skin directory.
- Which facets (e.g. Author, Type, Year, etc.) are included in the search interface.
- Whether the facets toolbar is displayed to the left or to the right of the search results.
- Which information are displayed for each search result (e.g. address, modified date, thumbnail, result number, etc.).
- The display styles (CSS) of the elements.
- Etc.

Most of the search interface settings can be edited using the Interface Editor, which can be opened by selecting **Edit this interface** in the **Do more** menu when a search page is already opened in a browser, or via the Windows Start menu. Click here for more details.

The search interface settings are saved in XML files on disk, in the directory `[.Net_Front-End_Path]\Web\Coveo\Skins`. Each search interface has its separate XML file, whose name starts with "INTERFACE_", followed by its name, and has the ".xml" extension (e.g. "INTERFACE_Default.xml"). Those XML files contain only the changes compared to the default settings. For example, if you modify the name of the **All Content** search interface and nothing else, its XML file will be almost empty and will only contain its new name. If a search interface does not have a search interface settings file, it uses all the default options.

In the code, the search interface settings are available through an instance of the `InterfaceSettings` class.

What Is the Structure of a Skin?

A skin is a set of user control (`.ascx`) files that define the appearance and behavior of a search interface.

The following lists the most important files in a skin:

- **`CoveoSearch.ascx`**
  This is the root file of the skin. It defines the overall HTML layout and includes the other skin files at the appropriate places. This file is a good place to add custom code that handles events triggered by the `SearchControl` object (see Query Execution Process Overview).
- **`InitialPanel.ascx`**
  This file defines the content of the panel displayed when the user loads the Search page without passing a query through the query string.
- **`SearchPanel.ascx`**
  This file defines the content of the panel above the search results. It usually contains a `Query` control and a search button.
- **`ResultsPanel.ascx`**
  This file defines the content of the panel containing the current query results and facets.
- **`ResultTemplate.ascx`**
  This file defines the content that is repeated for each query result. Modify this file to alter the result template.
- **`LeftToolbar.ascx`** and **`RightToolbar.ascx`**
  These files define the content of the panel, to the left and to the right of the results, that usually contains search facets.
- **`AdvancedSearchPanel.ascx`**
  This file defines the content of the Advanced Search panel (shown when clicking on the **Advanced Search** link usually below the query text box).
- **`PreferencesPanel.ascx`**
  This file defines the content of the **Preferences** page.

Obviously, there are several other files inside a typical skin. Feel free to explore them.

Controls Hierarchy

A Coveo .NET Search UI is composed of a series of controls loaded into an ASP.NET page

See the Loading the .NET Search UI topic to see how to load a Coveo .NET Search UI into an ASP.NET page.

The `SearchHub` Control vs. `BoundToHub` Controls

The root control of a Coveo .NET Search UI is the `SearchHub` object. That control is one of the rare search controls that is not included inside a skin. It is rather generally included in an ASPX page.

Example: SearchHub Control Included in an ASPX Page

```
<form id="f" runat="server">
  <ces:SearchHub id="h" runat="server" />
</form>
```

The `SearchHub` control is the root of the Coveo .NET search controls hiearchy. All the controls enumerated in the following sections are loaded (not necessarily directly) under the `SearchHub`control.

Through a `BoundToHub` object, inner controls can have access to the `SearchHub` control which they are embedded into by calling its `HubObject` property. This is also true for custom code declared in skin files.

The `BoundToSearch` class that is covered below inherits from the `BoundToHub` class. So any code that has access to a `BoundToSearch` object can also have access to the `SearchHub` object.

The `SearchControl` Control vs. `BoundToSearch` Controls

The root control of a search interface skin is always a `SearchControl` object. The `CoveoSearch.ascx` user control inherits from it, most of the time indirectly though. Each skin is usually bound to a specialized version of the `SearchControl` class i.e. a class that inherits from it and which adapts some behaviors to the search interface needs. For example: `PeopleSearch`, `EmailSearch`, `IntranetSearch`, etc

Example: CoveoSearch.ascx of the Intranet Skin

```
<%@ Control Language="c#" AutoEventWireup="false"
Inherits="Coveo.CES.Web.Search.Controls.IntranetSearch" %>

<%--************************************************************************
* This user control defines the look and structure of the CES search interface.
************************************************************************--%>

. . .
```

All the controls enumerated in the following sections are loaded (not necessarily directly) under the `SearchControl`control.

Through a `BoundToSearch` object, inner controls can have access to the `SearchControl` control which they are embedded into by calling its `SearchObject` property. This is also true for custom code declared in skin files.

The `BoundToQuery` class that is covered below inherits from the `BoundToSearch` class. A wide variety of other controls also inherit from the `BoundToSearch` class. Here are a few examples:

- `SortByFieldLinks`
- `DateParams`
- `QueryDuration`
- `SearchInterfaceLink`
- `MiniResultsPlaceHolder`
- `LoadIfActive`
- Etc.

The `QueryControl` Control vs. `BoundToQuery` Controls

The `ResultsPanel.ascx` user control of a skin usually inherits from the `ResultsPanel` class which itself inherits from the `QueryControl` class.

Example: ResultsPanel.ascx

```
<%@ Control Language="c#" AutoEventWireup="false"
Inherits="Coveo.CES.Web.Search.Controls.ResultsPanel" %>


. . .
```

All the controls that display information related to query results are instantiated (not necessarily directly) under the `QueryControl` control.

Through a `BoundToQuery` object, inner controls can have access to the `QueryControl` control which they are embedded into by calling its `QueryObject` property. This is also true for custom code declared in skin files.

Several controls inherit from the `QueryControl` class. Here are a few examples:

- `ResultsPanel`
- `MiniResultsPanel`
- `OutlookContextualSection`
- Etc.

The `BoundToResult` class that is covered below also inherits from the `BoundToQuery` class.

Result List Controls vs. `BoundToResult` controls

A skin generally includes at least one result list.

Example: MainResultList Control Included in ResultsPanel.ascx

```
. . .
        <%-- Results --%>
        <ces:MainResultList id="r" FetchParents="true" runat="server">
          <ResultTemplate>
            <ces:DefaultResultTemplate TemplatePath="ResultTemplate.ascx"
runat="server"/>
          </ResultTemplate>
          <ChildTemplate>
            <ces:DefaultResultTemplate TemplatePath="ChildTemplate.ascx"
runat="server"/>
          </ChildTemplate>
        </ces:MainResultList>
. . .
```

There are a few result list classes. Here are a few (some inherit from others):

- `MainResultList`
- `ResultList`
- `BaseResultList`
- `TableResultList`

All the controls that display information about a query result are instantiated (not necessarily directly) under a result list control. Here are a few result control examples:

- `ResultTitle`
- `ResultOpenLink`
- `ResultAuthor`
- Etc.

Through a `BoundToResult` object, inner result controls can have access to the `ResultInfo` object which they are attached to by calling its `Result` property.

Search Settings

The `SearchControl` object keeps track of the current query content using a collection of objects called Search Settings. Each of these objects is responsible for storing a particular aspect of the query, for example the keywords entered by the user, or the type of document to restrict to. All these objects derive from the `SearchSetting` class.

Usually, each `SearchSetting` usually has a strong relationship to a control that allows the user to control a specific aspect of the query. For example:

- The `Query` control (the text box in which the user enters keywords) is associated with a `QuerySetting` object.
- The `FieldValueFacet` control (which represents a facet, usually created as a child control by the `CustomRefineByFields` control) is associated with a `FieldValueFacetSetting` object.
- The `Format` control (the drop-down used to restrict to a specific format) is associated with a `FormatSetting` object.

When the user changes the content of these controls (called parameter controls), the associated search setting is updated to reflect the new value. Then, any instance of the same control located elsewhere on the page is notified to update its content.

Search Setting Persistence

The `SearchControl` object is responsible for persisting the search settings across requests. Two sets of settings are persisted:

- The `Staging` settings hold the search settings currently in the various parameter controls.
- The `Effective` settings hold the search settings that make up the current query.

Having two distinct sets of settings allows the user to edit queries without it having an immediate effect on the results displayed. The changes made by the user are only effective when the search button is clicked. At this point, the current effective settings are discarded and replaced by a copy of the staging settings. Most parameter controls thus only need to care about the staging settings (i.e. they never access the effective settings). However, this is not true for all controls. For example, the facet controls directly alter the effective settings.

Both sets of search settings are persisted by the `SearchControl` object in the page view state. When a postback occurs, the settings are read from the saved state, and an event is triggered to inform all parameter controls that they should initialize themselves using the content of the setting collections. The framework then proceeds through the normal page life cycle and at the end of it, the settings are saved back to the view state.

Query Execution Process Overview

In this topic:

Whenever a search interface needs to execute a query on the Coveo index server, the `SearchControl` and `QueryControl` objects run several steps in order to build the query, send it to the server, transform the results in ASP.NET controls, which are then inserted in the page and rendered to the browser.

(Click here for a more thorough description of the whole life cycle of a search page.)

When Are Queries Executed?

The `SearchControl` control can execute a query at two moments in the lifetime of an ASP.NET request:

- **`PreLoad`**
  During a postback (e.g. click in a facet), if a query was executed during the previous postback, it is executed again before the `Load` stage . The result controls of the previous HTTP request are re-created, they load their previous values from the view state, they can process their postback events. This makes possible to use controls such as buttons in the result templates, along with event handlers that perform processing when the user clicks the button.

  > Note that if the previous postback was performed not a long time ago (most of the time), the query results are retrieved from a cache, rather than having the previous query to be really executed again. So the burden on the Coveo index server is negligible.

- **`PreRender`**
  When a new search is performed following an action from the user such as clicking the search button, a query is executed during the `PreRender` stage.

Events Fired by the SearchControl

Events	Description	
BeforeQueries	The BeforeQueries event is fired by the SearchControl object. Therefore other components can execute certain tasks before the other query events are fired.	
PerformQueries	The PerformQueries event is fired by the SearchControl object. This makes all its QueryControl objects to perform all the steps required to execute their query on the Coveo index server.	
AfterQueries	The AfterQueries event is fired by the SearchControl object. Therefore other components can execute certain tasks after the other query events are fired.	

Populating the SearchBuilder Object

A SearchBuilder object accumulates all the information about a query that is going to executed by the Coveo index server. When the query is being prepared, various components of the search interface add their own bits of information (such as query expressions, sort, etc.). A SearchBuilder object is where all the information is gathered. The SearchControl and QueryControl objects work together in the preparation of the SearchBuilder object.

The SearchControl object fills the information that are common to all the QueryControl objects. This includes the Search Interface Settings, the Search Settings, along with various other settings. The SearchControl object also triggers the SetupSearchBuilder event to allow external code to tweak the query that is going to be executed.

The QueryControl object then adds the specific information that are related it, and triggers its SetupSearchBuilder event.

Arrived at this point, the SearchBuilder object is complete and ready to be sent to the Coveo index server.

From a SearchBuilder to a QueryWrapper

A SearchBuilder object describes every aspect of a search query. It doesn't have any knowledge about how the connection to the Coveo index server will be made (search provider), so it is not specific to a given search provider implementation.

To have the query executed by the Coveo index server, the SearchBuilder object first needs to be converted into a QueryWrapper object, that contains all the same exact information, but bound to a given search provider (e.g. CMF). The search provider API consists mainly of a set of interfaces (Coveo.CES.Web.Search.Providers namespace) that are implemented differently for every "kind of back-end servers" that are supported. The Coveo .NET Front-End comes with a few built-in search providers. The most important ones are:

- Coveo Messaging Framework (CMF)
- Coveo Messaging Framework for CES 6.5 (CMF465)

(Theorically, one could even develop its own search provider implementation that would cover his specific needs.)

Contrary to the SearchBuilder, a QueryWrapper also contains the query results returned by the Coveo index server after the query execution.

Before a new QueryWrapper object is actually created from a SearchBuilder object, the QueryWrapperFactory class first checks in its QueryWrapper cache if the same query has recently been executed (usually in the last minute). If so the existing QueryWrapper object is re-used, along with its query results. This helps reducing the burden on the Coveo index server. If a suitable QueryWrapper object is not present in the cache, a new one is created, and the query is sent to the Coveo index server to be executed and its results returned.

Creating the Result Controls

After retrieving its QueryWrapper, each QueryControl fires the CreateResultControls event in order to instruct the ResultList controls to instantiate the proper result template for every query result. The resulting ASP.NET controls are inserted in the page control tree, their appropriate events are triggered, and they render themselves in HTML to be finally sent to the browser.

What Is the Life Cycle of a Search Page Request?

The processing of an HTTP request to a search page involves the following steps.

Init

- The SearchHub control triggers the OverrideUser event to allow additional identities to be specified for the connection to the Coveo index server (and therefore for executing the search queries). This is useful when the search controls are integrated to a page that is

configured with a special authentication mechanism, and to potentially prevent the user from having to login twice to the same system.
- The `SearchHub` control establishes a connection to the Coveo index server. If the connection attempt fails, no other search controls are instantiated and an error page is displayed.
- The `SearchHub` control then loads all the search interfaces that it is configured with. During the load process, the `ShouldDisplayInterface` event is triggered for each search interface. This allows creating custom rules that control which search interfaces are made visible. When all the search interfaces are loaded, the `InterfacesLoaded` event is triggered.
- The `SearchHub` control then instantiates the `SearchInterface` child controls, which each in turn loads its respective `InterfaceSettings`, as well as its associated skin.
- The `SearchControl` object of the active `SearchInterface` then initializes itself, loading user's preferences, saved queries and filters, etc. Then it triggers the `ConfigIsLoaded` event.

LoadControlState (when performing a postback)

- The parameter controls restore their states.

LoadViewState (when performing a postback)

- The `SearchControl` object loads its persisted `SearchState` object containing, among other things, the staging and the effective settings.
- It then triggers the `LoadSettings` event, so that all parameter controls can initialize themselves with the value they had during the previous HTTP request (normally controls load their previous value directly from the view state, but parameter controls are using the search settings instead).

LoadPostData (when performing a postback)

- The parameter controls are updated with the values posted by the browser. (This is done automatically by the underlying ASP.NET Framework.) If a posted value is different than the one loaded using the search settings, the framework raises a flag to remember triggering, after the `Load` phase, a change or click event that the control supports.

PreLoad

- If a query was executed during the previous postback, it is executed again in order to re-create the result controls of the previous HTTP request. This allows them to use the view state and handle postback events.
  (See the `PreRender` stage below for a more thorough description of the query execution steps.)

> Note that if the previous postback was performed not a long time ago, the query results are retrieved from a cache, rather than having the previous query to be executed again. So the burden on the index is negligible.

Load

If this is the first load of the page:

- The query string is analyzed, and any search parameters that are present are loaded.
- The `InitializeSettings` event is raised to give an opportunity to initialize search settings based on custom rules.

RaisePostDataChangedEvent+RaisePostbackEvent (when performing a postback)

- The ASP.NET Framework triggers the change (or equivalent) event on controls flagged as having changed since the last postback (modification detection performed at the `Load` stage). When a parameter control receives a change event, it updates the search staging settings accordingly.
- If the current search interface has changed, the `SearchHub` control triggers the `ActiveInterfaceChanged` event.
- When the search staging settings become effective, the `SearchControl` object triggers the `TweakSettings` event.

LoadComplete

If the HTTP request is to restore a previous search state (browser bookmark, or back/forward navigation):

- The `AjaxManager` control (AJAX Framework) triggers the `RestoreState` event.
- The `SearchHub` control redirects it to the `SearchControl` object of the active search interface.
- The `SearchControl` object then triggers its `RestoreState` and `RestoreStateComplete` events.

PreRender

- The `SearchControl` object triggers the `LoadSettings` event again in order, for all the parameter controls, to update themselves using the settings that may have been modified by a change event.

- The `SearchControl` object then has the queries executed by the Coveo index server. (Click here to see the Query Execution Process Overview.) The `SearchControl` object triggers the following events, whose main goal is to have the QueryControl objects to execute their queries:
  - The SearchControl.BeforeQueries event is triggered. This allows external code to execute some custom logic.
  - The `SearchControl.PerformQueries` event is triggered. This also allows external code to execute some custom logic. And this also triggers the following sub-events:
    - A `SearchBuilder` object is created, and filled with all the effective search settings (query expression, sort, etc.). Search interface settings and various settings specific to every `QueryControl` object are also taken into account to fill the `SearchBuilder` object (duplicate filtering, etc.).
    - The `QueryControl.SetupSearchBuilder` and `SearchControl.SetupSearchBuilder` events are triggered to allow external code to override and customize the query that is going to be executed.
    - The final `SearchBuilder` object is then passed to the QueryWrapperFactory to retrieve a QueryWrapper that encapsulates the query (the following steps assume that the query is not in the cache).
    - The query is sent to the index for execution.
    - The `QueryControl.CheckResultAccess` and `SearchControl.CheckResultAccess` events are triggered for every query result.
    - The `QueryControl.TweakResult` and `SearchControl.TweakResult` events are triggered for every query results.
    - The `QueryControl.CreateResultControls` event is triggered. This causes the `ResultList` control to instantiate the result template for every query result that is to be displayed.
  - The SearchControl.AfterQueries event is triggered. This allows external code to execute some custom logic. This also triggers the following sub-events:
    - `SearchControl.GenerateComments`
      - `SearchHub.BeforeGenerateHubComments`
      - `SearchHub.GenerateHubComments`
  - Finally, the `SearchControl.MainQueryCompleted` event is triggered.

PreRenderComplete

- The `SearchControl` object triggers the `ReportAnalyticsData` event. If either the on-premises or cloud analytics module is enabled (or both), this will trigger the asynchronous push of the action that was performed during the current HTTP request.

SaveControlState+SaveViewState

- All the controls (`SearchControl`, `QueryControl`, the parameter controls, and all the others) in the page persist their states.

Render

- When this stage of the page processing is reached, all controls render themselves to produce the output HTML.
- On every query result, the `QueryControl.TweakUri` and `SearchControl.TweakUri` events are triggered.
- On every query result, the `ResultPrintableUri` control (if present in the skin result template) triggers the `QueryControl.TweakBreadcrumb` and `SearchControl.TweakBreadcrumb` events to allow external code to customize the breadcrumbs displayed.

Unload

If the `PreRenderComplete` stage was not triggered (can happen for example if the `Page.Redirect` or `Page.Response.End` methods were called), then the `SearchControl` object triggers the `ReportAnalyticsData` event.

AJAX Requests and SearchUpdatePanel Control

The .NET Search UI uses AJAX (Asynchronous JAvascript and XML) requests to provide a more streamlined user experience. Most of the time, when customizing the search interface, the developer does not need to care about it. However, there are situations where additional knowledge is required.

The .NET Search UI uses a model that is very close to Microsoft AJAX `UpdatePanel`'s in order to update the affected parts of the search page without fully reloading it (referred to as postback) when the user performs an action. It rather uses JavaScript code to perform a partial postback to the server that executes the required actions, and then updates only the affected HTML portions of the page.

The zones that can be updated individually are defined by the instances of the `SearchUpdatePanel` control. Whenever a control inside a `SearchUpdatePanel` control must be updated, the HTML of the whole nearest `SearchUpdatePanel` is rendered, sent to the browser and then updated. A `SearchUpdatePanel` control can contain other `SearchUpdatePanel` controls. When the ASP.NET request reaches the Render phase, the AJAX Framework uses different criteria to build the list of panels to update, and only the HTML of these specific panels is sent to the browser.

How the Search Interface Determines which Panels to Update

During a request processing, the `SearchControl` object keeps track of several events that can cause certain controls to require an update.

The following lists these events:

- A new query has been performed
- The staging settings have been modified
- The effective settings have been modified
- The `Mode` of the search interface has been modified
- The current result page has been modified
- The currently selected result has been modified
- A saved query or saved filter has been modified

When the request processing reaches the Render phase, each `SearchUpdatePanel` recursively scans its children, looking for controls implementing special marker interfaces associated to these events.

The available marker interfaces are:

- `IUpdateOnNewSearch`
- `IUpdateOnStagingChange`
- `IUpdateOnEffectiveChange`
- `IUpdateOnModeChange`
- `IUpdateOnPageChange`
- `IUpdateOnSelectedChange`
- `IUpdateOnQueryOrFilterChange`

When a control implementing one of these marker interfaces is found, and the associated event has been raised during the processing, the nearest `SearchUpdatePanel` automatically schedules itself for updating. Its parent `SearchUpdatePanel` controls (if any) are not triggered.

Making Updates More Granular

In order for updates to be more granular, it is possible to add `SearchUpdatePanel` controls to the custom skins that wrap around the updatable zones. No further logic is required, the search interface automatically uses these new panels when possible.

Writing New Controls Managed by SearchUpdatePanel's

The update logic of the search controls is managed by the `SearchUpdatePanel` controls. Most of the time, when modifying a skin, the developer does not need to take care about it. For example, a result control (typically in a result template) is updated automatically when the bound result changes. However, if a developer develops his own search control, he must determine which events should cause its rendered HTML to be updated, and implement the associated marker interfaces. Then, the control will be updated automatically whenever it is required.

Forcing the Update of a Control in Code

It is possible to force a specific control to be updated (through its nearest `SearchUpdatePanel`) by calling the `UpdatePanel.UpdateContainingPanel` method, and passing a reference to the control to update.

Modifying a Skin

A skin is a set of user control files (`.ascx`) that define the appearance and behavior of a search interface (see Search Interfaces and Skins). The Coveo .NET Front-End comes with a set of out-of-the-box skins that should not be tampered with to ensure that they are properly updated when you update the Coveo .NET Front-End. You can however duplicate an original skin file, and then freely modify the copy to achieve the appropriate result. You can then use the modified copy of the skin to obtain the desired search interface.

To modify a skin file:

1. Locate the `.ascx` skin file that you want to modify in the `[.Net_Front-End_Path]\Web\Coveo\Skins` folder (typically `C:\Program Files\Coveo .NET Front-End 12\Web\Coveo\Skins`).
2. Duplicate the `.ascx` skin file.

   > **Note:**
   > A best practice is to always duplicate/copy an out-of-the-box skin file before you modify it. Then you can apply your modifications on the duplicated skin file. Not doing so will most likely cause the out-of-the-box skin files to not be automatically updated when you upgrade the Coveo .NET Front-End to its latest version.

3. Using a text editor, open the duplicated or custom skin file to modify.
4. Make appropriate modifications (see What Is the Structure of a Skin?).
5. Save your modifications.
6. Ensure your search interface uses the skin you modified. You can assign a skin to an existing or new skin using the Interface Editor (see Creating a Search Interface With the .NET Interface Editor).
7. Reload the Search page in your web browser. The modifications should immediately be taken into consideration.

Customization Examples

The topics in this section contain examples of custom changes that can be done in a search page or in a search interface skin.

Adding Query Ranking Expressions to a Search Interface

Out-of-the-box, the ranking of the Coveo search results is optimized to return relevant documents for most cases. You can however use query ranking expressions (QRE) to modify the ranking behaviour of the search results. A QRE uses a positive or negative modifier value to respectively boost or reduce the ranking score of documents matching a given query expression.

Example:
If you want documents from a website to be ranked better (appear first in the search interface when results are sorted by relevancy) compared to equivalent documents from other indexed repositories, you can use a query ranking expression to boost the ranking of the website documents. The following QRE adds `50` to the ranking score of documents from the `MyWebSite` source.

```
expr.Expression = "@syssource='MyWebSite'";
expr.Modifier = 50;
```

Notes:

- A ranking modifier actually increases or decreases the relevance of search results, depending if its value is positive or negative respectively. Technically, its value can be in the range -2^31..2^31, but it is typical - and a good practice - to set it somewhere between -100..100, as this will not completely override the different ranking weights of search results (except for results that have the same QRE weight).
- The QRE `modifier` value is correlated to the document ranking score by a 1 to 10 ratio.

Example:
A QRE `modifier` value is `100`. Consequently this QRE adds `1000` to the ranking score of matching documents.

It is possible to add more than one ranking expression to the same search query. Ranking expressions can be specified by adding code to a search interface skin, either at the `SearchControl` or at the `QueryControl` level. The code can be added for example in the `CoveoSearch.ascx` file in the skin directory (`[.Net_Front-End_Path]\Web\Coveo\Skins\[SkinName]`).

The following code sample is a complete example of a query ranking expression implementation, and it is explaned below.

CoveoSearch.ascx

```
<script language="c#" runat="server">
protected override void OnInit(EventArgs p_Args)
{
 SetupSearchBuilder += this.Search_SetupSearchBuilder;
 base.OnInit(p_Args);
}
private void Search_SetupSearchBuilder(object sender, SetupSearchBuilderEventArgs
args)
{
 // Retrieve current query expression from the Effective query state.
 QuerySetting settings = ((QuerySetting)State.Effective[QuerySetting.NAME]);
 if (settings != null)
 {
  string QueryExpression = settings.Expression;
  // Use the SearchQuery object to parse the query and retrieve mandatory terms
  // which we'll use to build the query ranking expressions.
  ICESQuery query = Provider.CreateQuery();
  query.Expression = QueryExpression;
  IParsedQuery parsedQuery = query.GetParsedQuery();
  StringBuilder rankingQueryBuilder = new StringBuilder();
  foreach (object term in parsedQuery.MandatoryTerms)
  {
   if (rankingQueryBuilder.Length > 0 ) {
    rankingQueryBuilder.Append(",");
   }
   rankingQueryBuilder.Append(term.ToString());
  }

  // Apply various ranking expressions based on query expression typed by the user.

  // Field 1 expression
  RankingExpression expr = new RankingExpression();
  expr.Expression = "@syssource='CoveoSourceExample'";
  expr.Modifier = 50; // -100 to 100
  args.Builder.RankingExpressions.Add(expr);

  // Field 2 expression
  expr = new RankingExpression();
  expr.Expression = String.Format("{0}=({1})", "@examplefield",
rankingQueryBuilder.ToString());
  expr.Modifier = -50; // -100 to 100
  args.Builder.RankingExpressions.Add(expr);
 }
}
</script>
```

Script Block to Handle the `SetupSearchBuilder` Event

The sample is implemented as a script block added directly in the `CoveoSearch.ascx` file of the skin. The `SetupSearchBuilder` event is generally registered at the `Init` stage.

```
<script language="c#" runat="server">
protected override void OnInit(EventArgs p_Args)
{
 SetupSearchBuilder += this.Search_SetupSearchBuilder;
 base.OnInit(p_Args);
}
private void Search_SetupSearchBuilder(object sender, SetupSearchBuilderEventArgs
args)
{
 // Retrieve the current query and add new query ranking expressions.
 // ...
}
</script>
```

Parsing the Current Query Expression

The sample first checks whether the user has typed query terms. Adding QRE rules if no query has been typed cannot be done here, because one of the ranking expressions is based on the search query that the user has specified. You can retrieve the current query by looking for the `QuerySetting` object.

```
// Retrieve current query expression from the Effective query state.
 QuerySetting settings = ((QuerySetting)State.Effective[QuerySetting.NAME]);
 if (settings != null)
 {
  string QueryExpression = settings.Expression;
  // Use the SearchQuery object to parse the query and retrieve mandatory terms
  // which we'll use to build the query ranking expressions.
  ICESQuery query = Provider.CreateQuery();
  query.Expression = QueryExpression;
  IParsedQuery parsedQuery = query.GetParsedQuery();
  StringBuilder rankingQueryBuilder = new StringBuilder();
  foreach (object term in parsedQuery.MandatoryTerms)
  {
   if (rankingQueryBuilder.Length > 0 ) {
    rankingQueryBuilder.Append(",");
   }
   rankingQueryBuilder.Append(term.ToString());
  }

        // ...
 }
```

Adding Query Ranking Expressions

Now, here is the most relevant part: 2 custom ranking expressions are added to the `SearchBuilder` object:

- A first ranking expression is added to boost the ranking of the documents from a source named "CoveoSourceExample".
- A second ranking expression is added to boost the ranking of the documents whose `@examplefield` field contains query terms typed by the user.

```
// Apply various ranking expressions based on query expression typed by the user.

  // Field 1 expression
  RankingExpression expr = new RankingExpression();
  expr.Expression = "@syssource='CoveoSourceExample'";
  expr.Modifier = 50; // -100 to 100
  args.Builder.RankingExpressions.Add(expr);

  // Field 2 expression
  expr = new RankingExpression();
  expr.Expression = String.Format("{0}=({1})", "@examplefield",
rankingQueryBuilder.ToString());
  expr.Modifier = -50; // -100 to 100
  args.Builder.RankingExpressions.Add(expr);
```

AjaxManagerScript Partial Postback Error Handler

When performing Ajax partial postback requests, it is possible that the HTTP request returns a status different than 200 (OK). Errors are automatically handled by the `PartialPostBack` object. However, you may want to handle known errors by yourself. To do so, you can register a custom error handler with the `AjaxManagerScript` object. Then, when executing a partial postback, the custom error handler tries to handle the error before the default error handler is executed.

Note:
This feature is available starting with the Coveo .Net Front-End 12.0.252 July 2013 monthly release.

ErrorHandler Prototype

The error handler receives one parameter, which is the `XmlHttpRequest` object that produced the error. That object contains all the information needed to handle the error.

The handler must return a Boolean value that indicates whether the error has been handled or not. If nothing is returned, the error will be considered as not handled.

Usage

Registering an Error Handler in JavaScript

To specify an error handler in JavaScript, you must set the `PartialPostBackErrorHandler` property to your function as shown in the following example.

```
_coveoJQuery(document).ready(function() {

Coveo.CNL.Web.Scripts.Ajax.AjaxManagerScript.get_current().set_partialPostBackErrorHan
dler( function (request) {
        //Handle the error
        return true;
    });
});
```

Registering an Error Handler in C#

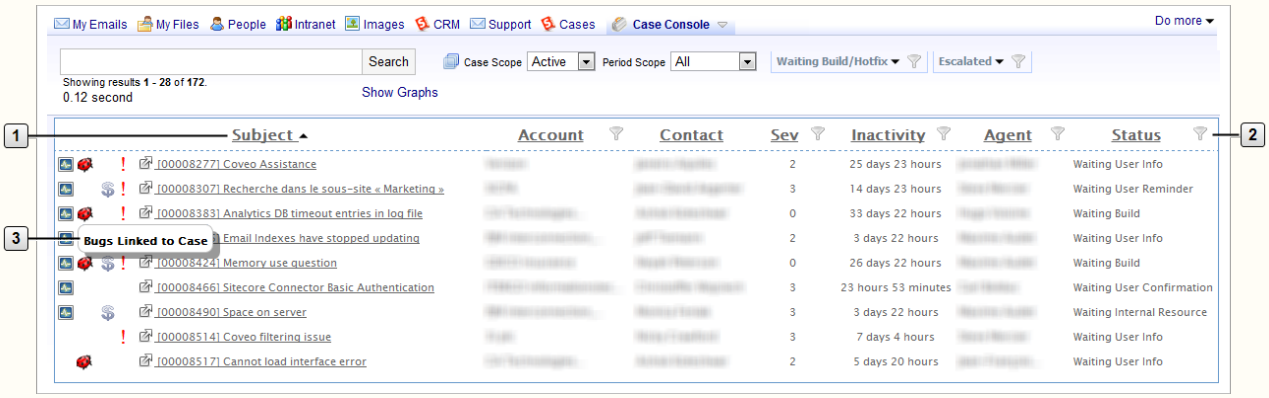
To specify an error handler in C#, you must register an inline script that sets the `PartialPostBackErrorHandler` property to your function as shown in the following example.

```
AjaxManager.Current.RegisterInlineScript(theControl,
   @"_coveoJQuery(document).ready(function() {

Coveo.CNL.Web.Scripts.Ajax.AjaxManagerScript.get_current().set_partialPostBackErrorHan
dler( function (request) {
      //Handle the error
      return true;
     });
    });"
  );
```

Configuring a TableResultList

You can customize a search interface to present search results in a tabular form using the `TableResultList` control. This search interface type is convenient for homogeneous search results containing short values. The table can feature columns on which the results can be sorted or filtered, acting as a facet. The table can also include elements rendered by specific ASCX controls.

Example:
You can use a table result list for a CRM case console as shown below. The end-user can use table result list features to navigate within the case information.



1	Clicking the title of a column sorts results in an ascending or descending order according to the content of the column.
2	Clicking the funnel icon ( ) allows the user to select values on which to filter results.
3	Bringing the pointer over dynamic elements shows tool tips.

You can use the `TableResultList` control in an existing search interface or in a new custom interface by instantiating it in the `ResultsPanel.ascx` file of the corresponding skin.

With the `TableResultList` control, you can:

- Use various column definition types with a variety of properties to build your table (see TableResultList Reference).
- Add code to register and handle table events to dynamically modify the table (see TableResultList Events).

To configure a table result list:

1. On the Coveo Front-End server, go to the skin folder corresponding to the search interface in which you wish to use the table result list. You can find the skin used by a search interface from the Interface Editor (**Search Interfaces** tab > **Features** menu > **General** page). The default folder for the skin files is: `[.Net_Front-End_Path]\Web\Coveo\Skins\[Search_Interface_Skin]`.

2. Using a text editor:
    a. Open the `ResultsPanel.ascx` file from the skin folder.
    b. In the file, find the location where you want to insert the table result list, and then add the `TableResultList` control lines (see TableResultList Reference).
    c. Add the code to register and handle the events from the table result list to allow modifying the table dynamically (see TableResultList Events).
    d. Save the file.

TableResultList Reference

This topic presents a `TableResultList` sample and describes the available elements and properties that you can include in the `ResultsPanel.ascx` file (see Configuring a TableResultList).

The following defines a table result list for a CRM case console with eight columns:

```
<ces:TableResultList HeaderWhenEmpty="true" MaxLines="2" runat="server" id="TRL">
  <ces:GenericColumnDefinition Width="200px" Align="left"
    HeaderControl="header.ascx"
    CellControl="TableResultListTemplate.ascx"
    FooterControl="footer.ascx" />
  <ces:StringColumnDefinition Width="100px" Title="Account" Field="@syssfaccount"
    GroupByField="@syssfaccountid" />
  <ces:StringColumnDefinition Width="100px" Title="Contact" Field="@syssfcontact" />
  <ces:StringColumnDefinition Width="100px" Title="Sev" Field="@syssfpriority"
    IsSortable
    IsGroupBy />
  <ces:DateTimeColumnDefinition Width="100px" Title="Inactivity" Field="@sysdate"
Align="right"
    IsSortable
    TimeSpan />
  <ces:StringColumnDefinition Width="100px" Title="Agent" Field="@syssfowner"
    IsGroupBy />
  <ces:StringColumnDefinition Width="100px" Title="Status"  Field="@syssfstatus"
    IsGroupBy
    GroupByAlignment="right" />
  <ces:StringColumnDefinition Width="100px" Title="Last Action"
Field="@sflastactiondetail" />
</ces:TableResultList>
```

`TableResultList` Properties

The `TableResultList` element is the main element and has the following properties.

- **`HeaderWhenEmpty:`**

  When set to `true`, specifies to display a header section even when no results are returned. The default is `false`.

- **`FooterWhenEmpty:`**

  When set to `true`, specifies to display a footer section even when no results are returned. The default is `false`.

- **`OutputStyles:`**

  When set to `false`, the stylesheet associated with the `TableResultList` elements is not included. This is useful for providing your own custom styling. The CSS classes on the elements are still present. The default is `true`.

- **`HeaderRowCssClass:`**

  Specifies the CSS class to output for the header row of the table.

- **`FooterRowCssClass:`**

  Specifies the CSS class to output for the footer row of the table.

- **`MaxLines:`**

  Specifies the maximum number of lines to display within a cell. This property is useful to allow result rows with variable number of lines, while limiting to the number of lines.

- **`FixedLines:`**

  Specifies the exact number of lines to display within a cell. This property is useful to force a number of lines per result row and consequently obtain a uniform look for all the result rows.

Column Definition Types

In the `TableResultList` element, you must declare every column that you want to display in the table, each column being of a given type. The available column definition types are:

- `GenericColumnDefinition`:

  Use this column type to define a column whose content comes directly from the ASCX file.

  > **Example:**
  > A generic column definition may populate a cell with the content from an ASCX file. It may include several elements such as icons, text, and tool tips.

- `StringColumnDefinition`:

  Use this column type to create a column that will show a result document string field value.

- `DateTimeColumnDefinition`:

  Use this column type to create a column that will show a result document date/time field value.

> All the above column definition types inherit directly or indirectly from the `AbstractColumnDefinition` class. You could derive your own class from it to implement some features not supported by the built-in column definition types.

Column Definition Properties

Each column definition type supports a set of properties. Several properties are inherited from the `AbstractColumnDefinition` and `FieldColumnDefinition` abstract classes.

The following table indicates which properties are available on each column definition type.

	Column Definition Types			Abstract Column Definition Types	
Properties	**Generic**	**String**	**DateTime**	**Field**	**Abstract**
Table					
HeaderControl					
Header					
FooterControl					
Footer					
CellControl					
Cell					
Width					
Align					
IsGroupBy					
GroupByField					
GroupByAlignment					
HeaderCssClass					
FooterCssClass					

FunelPopUpType					
FacetSortBy					
FacetName					
LookupField					
Title					
LocalizedTitle					
Field					
IsSortable					
SortByField					
RenderIfEmpty					
Format					
TimeSpan					
Long					
Options					
ShortenMaxLength					

The available column definition properties are described below:

- `Table`:

  Gets the reference to the `TableResultList` control. Used only when implementing a custom column definition type.

- `HeaderControl`:

  Gets or sets the path to the ASCX file whose content is used to populate the header cells.

- `Header`:

  Gets the control displayed in the header. Used only when implementing a custom column definition type.

- `CellControl`:

  Gets or sets the path to the ASCX file whose content is used to populate the result row cells.

- `Cell`:

  Gets the control template for the cells. Used only when implementing a custom column definition type.

- `FooterControl`:

  Gets or sets the path to the ASCX file whose content is used to populate the footer cells.

- `Footer`:

  Gets the control displayed in the footer. Used only when implementing a custom column type.

- `ShortenMaxLength`:

  Gets or sets the maximum length of the content, in number of characters. When the string length exceeds this value, the ending part of the string is truncated and replaced by an ellipsis (…).

- `Width`:

  Gets or sets the width of the column, in pixel (e.g. `Width="100px"`).

- `Align`:

  Gets or sets the column text alignment (`left` or `right`). The default is `left`.

- `IsGroupBy`:

  When set to `true`, specifies that the column can act like a facet. A funnel icon ( ) appears in the header to allow users to select

from a dialog box the values on which to filter the results. The default is `false`.

- `GroupByField`:

  Gets or sets the name of the field used when `IsGroupBy` is `true`.

- `GroupByAlignment`:

  Gets or sets the column side (`left` or `right`) on which the group-by dialog box appears. This property is useful to ensure that the dialog box is not displayed outside of the visible area for columns to the far right.

- `HeaderCssClass`:

  Gets or sets the CSS class specified on the header cells (`th` tag).

- `FooterCssClass`:

  Gets or sets the CSS class specified on the footer cells (`td` tag).

- `FunelPopUpType`:

  Gets or sets the type of control used to render the funnel popup control. Used only when implementing a custom column definition type.

- `FacetSortBy`:

  Gets or sets the facet sort type.

- `FacetName`:

  Gets or sets the facet name (only for data recorded by the Analytics module). If not set, the localized title is used.

- `LookupField`:

  Gets or sets the field to display in the facet (e.g. client name instead of the client ID).

- `Title`:

  Gets or sets the text displayed in the column header.

- `LocalizedTitle`:

  Gets or sets the multilingual string unique name. This property allows to support the standard Coveo multilingual mechanism to dynamically show messages in the current UI language. When it is not set, the value of the `Title` property is used.

Note:
The multilingual string must be defined in the `[.Net_Front-End_Path]\Web\Strings\SearchStrings.xml` file. When custom strings are added to the `SearchStrings.xml` file, that XML file must be referenced in the `Web.config` file.

- `Field`:

  Gets or sets the name of the field to display.

- `IsSortable`:

  When set to `true`, the label of the header becomes clickable to allow users to sort the results in an ascending or descending order of the column content. The default is `false`.

- `SortByField`:

  Gets or sets the name of the field used for sorting, if different than the value of `Field`.

- `RenderIfEmpty`:

  When set to `true`, the "non-applicable" string ("`n/a`" in English) is rendered in the cell when the field value is empty.

Note:
The "non-applicable" string ("`n/a`") is defined for all languages in the `[.Net_Front-End_Path]\Web\Strings\SearchStrings.xml` file.

- `Format`:

  Gets or sets the optional string containing a specific .NET date/time format in which to display the date/time values.

Note:

Refer to the Microsoft Date and Time Format Strings document for details on standard and custom date/time format strings.

- `TimeSpan`:

  When set to `true`, displays the date/time value in the form of a time span relative to now. The default is `false`.

Example:
`2 hours 7 minutes`

- `Long`:

  When set to `true`, displays the date/time in the standard long format. The default is `false`.

- `Options`:

  Gets or sets the `DateTimeToStringOptions` option to use to convert the date/time values to strings.

Example:

```
Options="AlwaysIncludeTime"
```

TableResultList Events

You can register events on the `TableResultList` control to dynamically modify elements of the table. The events can registered in a script block in the ASCX file.

The following sample illustrates how to register all the available events on a `TableResultList` control that would have been declared with `id="TRL"`.

```
protected override void OnLoad(EventArgs p_Args) {
    TRL.PreRenderRow += new
EventHandler<TableResultListRowEventArgs>(EventHandler_PreRenderRow);
    TRL.PostRenderRow += new
EventHandler<TableResultListRowEventArgs>(EventHandler_PostRenderRow);
    TRL.PreRenderCell += new
EventHandler<TableResultListCellEventArgs>(EventHandler_PreRenderCell);
    TRL.PostRenderCell += new
EventHandler<TableResultListCellEventArgs>(EventHandler_PostRenderCell);
    TRL.PreRenderHeader += new
EventHandler<TableResultListHeaderEventArgs>(EventHandler_PreRenderHeader);
    TRL.PostRenderHeader += new
EventHandler<TableResultListHeaderEventArgs>(EventHandler_PostRenderHeader);
    TRL.PreRenderFooter += new
EventHandler<TableResultListFooterEventArgs>(EventHandler_PreRenderFooter);
    TRL.PostRenderFooter += new
EventHandler<TableResultListFooterEventArgs>(EventHandler_PostRenderFooter);
    base.OnLoad(p_Args);
}
```

You do not need to register all the events, only the ones you need to use.

The following sample shows how the `PreRenderRow` event could be implemented to:

- Change the way that a query result is rendered by displaying other data in the row.
- Change the background color for the query result.
- Display a special row before the query result.

```
private void EventHandler_PreRenderRow(object p_Sender, TableResultListRowEventArgs
p_Args)
{
  if (p_Args.Result.ResultObject.Date <= DateTime.Now.AddYears(-11)) {
    // Change the way that the result is rendered by displaying other data in the row.
    p_Args.RenderRow = false;
    p_Args.HtmlWriter.RenderBeginTag("tr");
      p_Args.HtmlWriter.RenderBeginTag("td");
        p_Args.HtmlWriter.Write("SFCase");
      p_Args.HtmlWriter.RenderEndTag();
      p_Args.HtmlWriter.RenderBeginTag("td");
        p_Args.HtmlWriter.Write("Some more content");
      p_Args.HtmlWriter.RenderEndTag();
      p_Args.HtmlWriter.RenderBeginTag("td");
      p_Args.HtmlWriter.RenderEndTag();
      p_Args.HtmlWriter.RenderBeginTag("td");
      p_Args.HtmlWriter.RenderEndTag();
      p_Args.HtmlWriter.RenderBeginTag("td");
      p_Args.HtmlWriter.RenderEndTag();
    p_Args.HtmlWriter.RenderEndTag();
  } else if (p_Args.Result.ResultObject.Date <= DateTime.Now.AddYears(-10)) {
    // Change the background color for this result.
    p_Args.HtmlWriter.AddStyleAttribute("background-color", "Red");
  } else if (p_Args.Result.ResultObject.Date <= DateTime.Now.AddYears(-9)) {
    // Display a special row before the result.
    p_Args.HtmlWriter.RenderBeginTag("tr");
      p_Args.HtmlWriter.AddAttribute("colspan", "5");
      p_Args.HtmlWriter.RenderBeginTag("td");
        p_Args.HtmlWriter.Write("This will be displayed Before the result");
      p_Args.HtmlWriter.RenderEndTag();
    p_Args.HtmlWriter.RenderEndTag();
  }
}
```

The following sample shows how the `PostRenderRow` event could be implemented to display a special row after a query result.

```
private void EventHandler_PostRenderRow(object p_Sender, TableResultListRowEventArgs
p_Args)
{
  if (p_Args.Result.ResultObject.GetField("@sysfiletype") == "SFTask") {
    // Display a special row after the result
    p_Args.HtmlWriter.RenderBeginTag("tr");
      p_Args.HtmlWriter.AddAttribute("colspan", "5");
      p_Args.HtmlWriter.RenderBeginTag("td");
        p_Args.HtmlWriter.Write("This will be displayed After the result");
      p_Args.HtmlWriter.RenderEndTag();
    p_Args.HtmlWriter.RenderEndTag();
  }
}
```

`TableResultList` Event Arguments

The available event argument classes are:

- `TableResultListRowEventArgs`
- `TableResultListCellEventArgs`
- `TableResultListHeaderEventArgs`
- `TableResultListFooterEventArgs`

Each event argument classes has a set of properties. The following table indicates which properties are available on each event argument class.

	Event Argument			
Attribute	**Row**	**Cell**	**Header**	**Footer**
Table				
HTMLWriter				
Result				
RowCssClass				
RenderRow				
ColumnDefinition				
RenderCell				

The available properties for table result list events elements are described below.

- `Table`:

  Gets or sets the `TableResultList` control.

- `HtmlWriter`:

  Gets or sets the `HtmlWriter` object used to output the query result HTML markup.

- `Result`:

  Gets or sets the current query result being displayed.

- `RenderRow`:

  Gets or sets whether to render the current query result row. Only applies to `PreRenderRow` events.

- `RowCssClass`:

  Gets or sets the CSS class that will be specified for the current query result row. Only applies to `PreRenderRow` events.

- `ColumnDefinition`:

  Gets or sets the `AbstractColumnDefinition` object that defines this column.

- `RenderCell`:

  Gets or sets whether to render the current cell. Only applies to `PreRenderRow` events.

Creating or Customizing Related Results Display Template Skin Files

In a Coveo .NET search interface, a Related Results panel can present search results from another search interface (see About Related Results in .NET Search Interfaces). A Related Results panel uses a display template ASCX file to define what is rendered inside the panel. The location and default name for that file is `[.NET_Front-End_Path]\Web\Coveo\Skins\[Interface_Name]\RelatedResultsDisplayTemplate.ascx`. You can duplicate and customize this file in each search interface. Once custom Related Results display template files are available in a skin folder, a Coveo administrator can add a Related Results panel to a search interface (see Adding or Customizing a Related Results Panel With the .NET Interface Editor).

Display template files must inherit from the `RelatedResultsWidget` class (directly or indirectly). These template ASCX files can be modified using a text editor. You can add controls to them. You can also add script code block that is run on the server to modify their behavior and configuration.

The following sections provide some samples.

> **Note:**
> The Related Results feature is available starting with the Coveo .NET Front-End 12.0.404+ (October 2013 monthly release).

To create or customize a Related Results display template file:

1. Using an administrator account, connect to the Coveo Front-End server (where the Coveo .NET Front-End is installed).

2. In the `[.NET_Front-End_Path]\Web\Coveo\Skins\[Interface_Name]` folder of the interface for which you want to add a template file, make a copy of the `RelatedResultsDisplayTemplate.ascx` file.
3. Using a text editor, open the copied file and modify the file content as desired. Refer to the following sections for examples.

Changing the Number of Displayed Results

The default number of results that appear in a Related Results panel is 3. You can change this value with a script block like the following.

Note:
You can also change the number of results from the Interface Editor.

```
<script runat="server">
  protected override void OnInit(EventArgs args) {
    QueryObject.ResultsPerPage = 10;
    base.OnInit(args);
  }
</script>
```

Filtering Related Results by a Facet from the Parent Search Interface

By default, a Related Results panel only imports the query from the parent search interface. You can use a script block like the following to also take into account values selected in a facet from the parent search interface.

```
<%@ Import Namespace="Coveo.CES.Web.Search.Settings" %>
<script runat="server">
  protected override void OnImportSettings(EventArgs args) {
    // When importing settings for a new Related Results query, also import the
desired facet setting from the parent search interface.
    string settingName = FieldValueFacetSetting.ComputeName(CustomFields.SYSYEAR, "");
    FieldValueFacetSetting parentSetting = (FieldValueFacetSetting)
ParentSearchObject.State.Staging[settingName];
    if (parentSetting != null) {
      SearchObject.State.Staging.Add(parentSetting);
    } else {
      // Remove the setting in our state if the parent search interface doesn't have
it anymore.
      FieldValueFacetSetting currentSetting = (FieldValueFacetSetting)
SearchObject.State.Staging[settingName];
      if (currentSetting != null) {
        SearchObject.State.Staging.Remove(settingName);
      }
    }
    base.OnImportSettings(args);
  }

  protected override bool ShouldUpdateResults() {
    // By default, Related Results panels update their results only when a new search
is performed in the parent search interface.
    // Tell this one to also update its results when a facet state changes in the
parent search interface.
    return base.ShouldUpdateResults() || ParentSearchObject.StagingHasChanged;
  }
</script>
```

Filtering Related Results by All the Parent Search Interface Facets

By default, a Related Results panel only imports the query from the parent search interface. You can use a script block like the following to also take into account values selected in all the facets from the parent search interface.

```
<%@ Import Namespace="System.Collections.Generic" %>
<%@ Import Namespace="Coveo.CES.Web.Search.Settings" %>
<script runat="server">
  protected override void OnImportSettings(EventArgs args) {
    // Clear all the FieldValueFacetSetting for the Related Results panel before
adding new ones.
    // Warning: This also removes FieldValueFacetSetting automatically initialized by
the Related Results panel SearchControl.
    List<FieldValueFacetSetting> settingsToRemove = new
List<FieldValueFacetSetting>();
    foreach (KeyValuePair<string, SearchSetting> pair in
SearchObject.State.Staging.Dictionary) {
      if (pair.Value is FieldValueFacetSetting) {
        settingsToRemove.Add((FieldValueFacetSetting) pair.Value);
      }
    }
    foreach (FieldValueFacetSetting settingToRemove in settingsToRemove) {

SearchObject.State.Staging.Remove(FieldValueFacetSetting.ComputeName(settingToRemove.F
ield, ""));
    }

    // When importing settings for a new Related Results panel query, also import all
the facet settings from the parent search interface.
    foreach (KeyValuePair<string, SearchSetting> pair in
ParentSearchObject.State.Staging.Dictionary) {
      if (pair.Value is FieldValueFacetSetting) {
        SearchObject.State.Staging.Add(pair.Value);
      }
    }
    base.OnImportSettings(args);
  }

  protected override bool ShouldUpdateResults() {
    // By default, Related Results panels update their results only when a new search
is performed in the parent search interface.
    // Tell this one to also update its results when a facet state changes in the
parent search interface.
    return base.ShouldUpdateResults() || ParentSearchObject.StagingHasChanged;
  }
</script>
```

Changing the Target Frame of Result Open Links in a Related Results Panel

By default, when a user clicks a query result in a Related Results panel, the document opens in the current browser tab. You can force the document to open in another browser tab by setting the link's frame to `"_blank"`.

```
<script runat="server">
  protected override void OnInit(EventArgs args) {
    SearchObject.TweakUri += new TweakUriEventHandler(SearchObject_TweakUri);
    base.OnInit(args);
  }

  private void SearchObject_TweakUri(object sender, TweakUriEventArgs args) {
    args.Frame = "_blank";
  }
</script>
```

Adding a Constant Query Expression to a Related Results Panel

You can add a constant query expression to a Related Results panel to be applied on top of the query performed by end-users in the parent search interface.

Example:
If you want the Related Results panel to only show results for document created or modified in 2013. You can add the constant query expression `@sysyear==2013`.

```
<script runat="server">
  protected override void OnInit(EventArgs args) {
    SearchObject.SetupSearchBuilder += new
SetupSearchBuilderEventHandler(SearchObject_SetupSearchBuilder);
    base.OnInit(args);
  }

  private void SearchObject_SetupSearchBuilder(object sender,
SetupSearchBuilderEventArgs args) {
    args.Builder.AddConstantExpression("My Constant Query");
  }
</script>
```

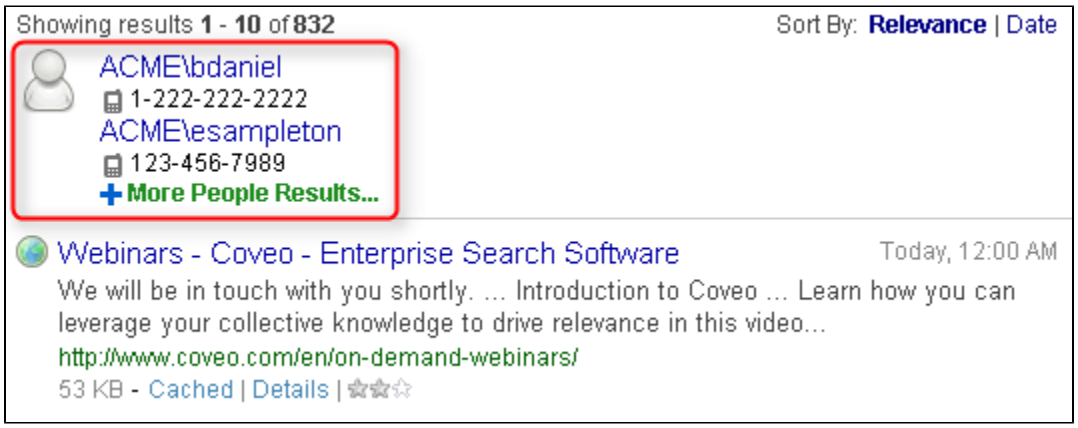
Adding a Query Ranking Expression to a Related Results Panel

When the Related Results panel sort order is **Relevance**, you can customize the ranking of the results appearing in a Related Results panel using query ranking expressions (QRE) (see What Are Query Ranking Expressions?).

```
<%@ Import Namespace="Coveo.CES.Web.Search.Providers" %>
<script runat="server">
  protected override void OnInit(EventArgs args) {
    SearchObject.SetupSearchBuilder += new
SetupSearchBuilderEventHandler(SearchObject_SetupSearchBuilder);
    base.OnInit(args);
  }

  private void SearchObject_SetupSearchBuilder(object sender,
SetupSearchBuilderEventArgs args) {
    RankingExpression qre = new RankingExpression();
    qre.Expression = "query";
    qre.Modifier = 100;
    qre.IsConstant = true; // false if qre.Expression is dynamic
    args.Builder.AddRankingExpression(qre);
  }
</script>
```

Customizing Mini-Results Displayed in a Search Interface

Mini-results are used in the .NET Search UI to display information that is complementary to search results. They actually are search results themselves i.e. search results that come from another search interface. They are inserted just before standard search results.



Sometimes it can be convenient to control which mini-results are displayed in which search interface, instead of relying on default settings. For example, you could display contacts that are related to images searched under the Images search interface.

For this purpose, the `CheckMiniResultsVisibility` event is exposed on the `SearchHub` control. You need to attach a handler function that specifies which search interfaces are used as a source that provides mini-results. This handler function must have the following signature:

```
public delegate void CheckMiniResultsVisibilityEventHandler(object sender,
CheckMiniResultsVisibilityEventArgs args);
```

The `CheckMiniResultsVisibilityEventArgs` argument object exposes two properties:

- `ActiveInterface` which is a `SearchInterface` object. It refers to the currently selected search interface in the search hub for which mini-results are being created.
- `InterfacesToCreateMiniResultsFrom` which is a list of `SearchInterface` objects. These are the search interfaces from which mini-results can come from.

> **Note:**
> The `CheckMiniResultsVisibility` event is available starting with the Coveo .NET Front-End 12.0.467+ December 2013 monthly release.

The following section provides a sample.

Adding Mini-Results to a Specific Search Interface

This sample consists of an ASPX page that contains a `SearchHub` control, and a script block that handles the `CheckMiniResultsVisibility` event. The event handler disables all the mini-results, except the People mini-results that are added only to the Images search interface.

1. On the Coveo Front-End server, using a text editor:
   a. Paste the following code in a new text file:

```
<%@ Page language="c#"%>
<cnlm:Doctype runat="server" />

<%-- Force IE to display in the highest compatibility mode possible that we
tested... --%>
<cnlm:XUACompatibleMetaElement Value="IE10" runat="server" />

<html style="overflow-y: scroll">
    <head runat="server"></head>
    <body style="margin: 0px;">
        <form id="f" runat="server">
            <ces:SearchHub id="h" runat="server" />
        </form>
    </body>
</html>

<script runat="server">
    protected override void OnInit(EventArgs e)
    {
        h.CheckMiniResultsVisibility +=
SearchHub_CheckMiniResultsVisibility;
        base.OnInit(e);
    }
    private void SearchHub_CheckMiniResultsVisibility(object sender,

CheckMiniResultsVisibilityEventArgs args)
    {
        // Remove mini-results from all interfaces.
        args.InterfacesToCreateMiniResultsFrom.Clear();
        // Display People-related mini-results only in the Images search
interface.
        if (String.Equals(args.ActiveInterface.Name, "images",
StringComparison.InvariantCultureIgnoreCase)) {

args.InterfacesToCreateMiniResultsFrom.Add(h.GetSearchInterfaceByName("Peop
le"));
        }
    }
</script>
```

   b. Adapt the pasted code to the desired behavior.
   c. Save it as an `aspx` file under the `web` folder (`[.NET_Front-End_Path]\Web\`). You can give it an arbitrary name, such as `MySearchPage.aspx`.
2. Load the newly created search page in a web browser (typical URL is http://localhost:8080/MySearchPage.aspx).
3. Validate that mini-results are displayed in the search interface as specified in the code.

Customizing the Default Sort Order of Search Results

Whenever you perform a query in a search interface, search results are sorted and displayed according to a specific order, which is by **Relevance** by default.

You may want to modify that default sort order, for example to sort search results by **Date** or by any other custom field.

There are essentially two classes that make it possible:

- The `SearchControl` control which exposes the `InitializeSettings` event. This event is usually where the code for customizing the sort order is added. The `InitializeSettings` event is called at the load of the search page to configure the initial search settings. This event is not called when the user executes further actions in the search page like clicking in facets. An `InitializeSettings` even t handler is a good place to add a new search setting that customizes the sort order.
- The `SortByFieldSetting` class exposes two particularly relevant properties:
  - The `Field` property, which is a string, is used to specify a custom field by which search results should be sorted.
  - The `Descending` property, which is a boolean, is used to specify whether search results should be sorted in descending or ascending order.

Note:
The `Field` property accepts only field names preceded by the @ character. For example, to sort search results by a field named `MyTestFi eld`, you should set the value of the `Field` property to `@MyTestField`. Alternatively, you can also use the `CustomFields` class, which defines constants for the most commonly used custom fields: SYSAUTHOR, SYSDATE, SYSFILETYPE, SYSCOLLECTION, SYSSOURCE, etc.

The following section provides a sample.

Sorting by Modification Date by Default

To sort search results by modification date by default:

1. On the Coveo Front-End server, go to the skin folder that corresponds to the search interface in which you want to customize the default sort order.
   You can find the skin used by a search interface from the Interface Editor (**Search Interfaces** tab -> **Features** menu -> **General** page).
   The default folder for skin files is: `[.Net_Front-End_Path]\Web\Coveo\Skins\[Search_Interface_Skin]`.

   > You are strongly advised to make a copy of a built-in skin before modifying it, the reason being that installing a more recent .NET Front-End will not update it properly.

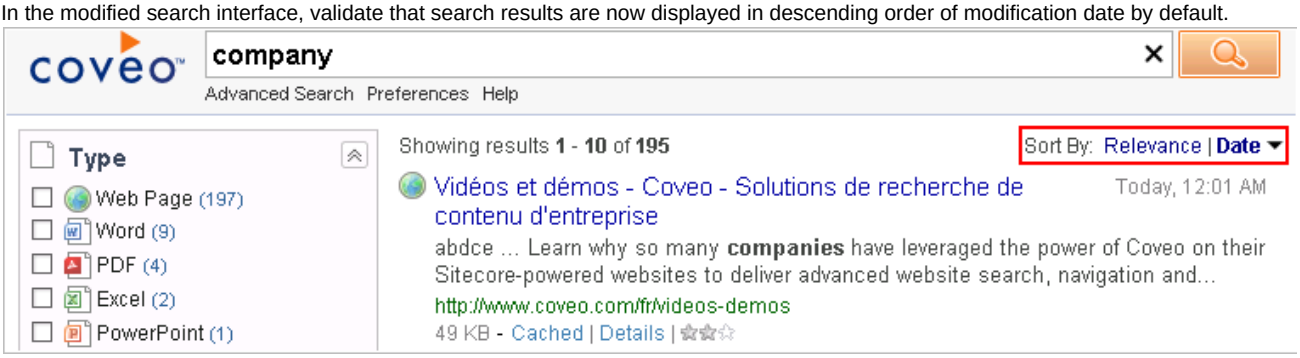
2. Using a text editor:
   a. In the skin folder, open the `CoveoSearch.ascx` file.
   b. At the end of the file, paste the following code block sample that modifies the default sort order of search results by displaying the most recent ones first.

Code: modifying the default sort order.

```
<%@ Import Namespace="Coveo.CES.Web.Search.Settings" %>
<script runat="server">
    protected override void OnInitializeSettings(EventArgs p_Args)
    {
        // Sort search results by date by default, in descending order.
        // We first create a custom SortByFieldSetting instance and then
add
        // it to the search interface settings.
        SortByFieldSetting setting = new SortByFieldSetting();
        setting.Field = CustomFields.SYSDATE;
        setting.Descending = true;
        State.Staging.Add(setting);

        // Call the original method to apply all the other search
        // interface settings.
        base.OnInitializeSettings(p_Args);
    }
</script>
```

   c. Save the file.

3. In the modified search interface, validate that search results are now displayed in descending order of modification date by default.



Enabling Fuzzy or Phonetic Search in the People Search Interface

The Coveo Enterprise Search (CES) index can expand searched terms with fuzzy or phonetic matches to help users find what they are looking for (see Advanced Field Queries). Fuzzy or phonetic matches are particularly useful when searching for people names for which users do not remember or know the exact spelling.

Starting with Coveo .NET Front-End 12.0.856 (July 2014 monthly release), when the `@sysfullnameexpanded` field exists in the index, you can configure the **People** search interface to either use the fuzzy or phonetic match for searched names.

When fuzzy matches are enabled, the quey is automatically changed to: `@sysfullnameexpanded~=keywords`

Similarly, when phonetic matches are enabled, the query is automatically changed to: `@sysfullnameexpanded%=keywords`

To enable the fuzzy or phonetic search in the People search interface:

1. Locate the `[.Net_Front-End_Path]\Web\Coveo\Skins\People\CoveoSearch.ascx` skin file for the People search interface (typically `C:\Program Files\Coveo .NET Front-End 12\Web\Coveo\Skins\People\CoveoSearch.ascx`).
2. Duplicate the `.ascx` skin file.

   > **Note:**
   > A best practice is to always duplicate/copy an out-of-the-box skin file before modifying it. Then you can apply your modifications on the duplicated skin file. Not doing so will most likely cause the out-of-the-box skin files to not be automatically updated when you upgrade the Coveo .NET Front-End to a more recent version.
3. Using a text editor, open the duplicated or custom skin file to modify.
4. At the end of the first line, before the closing symbols (%>), add either `QueryMode="Phonetic"` or `QueryMode="Fuzzy"` depending on the type of match that you want to enable.

   Example
   ```
   <%@ Control Language="c#" AutoEventWireup="false"
   Inherits="Coveo.CES.Web.Search.Controls.PeopleSearch" QueryMode="Phonetic" %>
   ```
5. Save your modifications.
6. Ensure your search interface uses the skin you modified. You can assign a skin to an existing or new skin using the Interface Editor (see Creating a Search Interface With the .NET Interface Editor).
7. Reload the Search page in your web browser. The modifications should immediately take effect.

Injecting Identities

Your Coveo index may contain sources for which permissions are indexed with identities defined in one or more user directories. When a user accesses the Coveo .NET search interface, the queries that the user performs are made with a "default" identity, which mainly depends on how the web site authentication is configured in IIS. In search results, the user will only see the documents to which that "default" identity has access.

Example:
On a Windows system by default:

- When the anonymous authentication is enabled on the web site, the query is made with the Windows well-known `NT AUTHORITY\ANONYMOUS LOGON` (`S-1-5-7`).
- When both the anonymous authentication and the ASP.NET impersonation are disabled on the web site, the query is made with the Windows account which the application pool is configured to run as.
- Otherwise and when the Windows authentication is enabled on the web site, the query is made with the Active Directory identity of the connected user that was authenticated by IIS and the browser.

For various repositories, the identity of an authenticated user is not automatically passed to the query. The consequence is that documents that the user should legitimately see with this identity will not appear in the search results.

Example:
A user is logged in to a Confluence site with his Confluence identity. By default, the Coveo .NET search interface integrated in that Confluence site does not automatically take that Confluence identity into account to run search queries.

The solution is to add one or more user and/or group identities to every query executed through a search hub. By injecting these identities, the index will retrieve the appropriate results from the sources crawled with a corresponding security provider.

The following code sample is a complete example that injects additional identities, and it is explained below.

```
<%@ Page language="c#" %>
<cnlm:Doctype runat="server" />
<cnlm:XUACompatibleMetaElement Value="IE10" runat="server" />

<html style="overflow-y: scroll">

<body style="margin: 0px;">
 <script runat="server">
 void h_OverrideUser(object p_Sender,
Coveo.CES.Web.Search.Controls.OverrideUserEventArgs p_Args)
    {
  string userOrGroupName = "UserName";
  string securityProviderName = "SecurityProvider";
  bool isGroup = false; // or true

  Coveo.CES.Web.Search.Providers.ICESUserIdentityFactory factory =
Coveo.CES.Web.Search.Providers.SearchProviderFactory.CreateDefaultUserIdentityFactory(
);
  Coveo.CES.Web.Search.Providers.IUserIdentity userOrGroup =
factory.CreateSecurityProviderUser(userOrGroupName, securityProviderName , isGroup,
null);
  p_Args.AdditionalIdentities.Add(userOrGroup);
 }
 </script>

 <form id="f" runat="server">
  <ces:searchhub name="Confluence" id="h" runat="server"
OnOverrideUser="h_OverrideUser" />
 </form>
</body>
</html>
```

Handling the `OverrideUser` Event

You must implement a handler for the `OverrideUser` event on the `SearchHub` control. In the code below, this event handler is named `h_OverrideUser` .

```
<%@ Page language="c#" %>
<cnlm:Doctype runat="server" />
<cnlm:XUACompatibleMetaElement Value="IE10" runat="server" />

<html style="overflow-y: scroll">

<body style="margin: 0px;">
 <script runat="server">
 void h_OverrideUser(object p_Sender,
Coveo.CES.Web.Search.Controls.OverrideUserEventArgs p_Args)
    {
  // The code to inject the user or group identity goes here.
 }
 </script>

 <form id="f" runat="server">
  <ces:searchhub name="Confluence" id="h" runat="server"
OnOverrideUser="h_OverrideUser" />
 </form>
</body>
</html>
```

Injecting a User or Group Identity

The following steps describe how to inject a user or group identity:

1. Get an `ICESUserIdentityFactory` object to create the identities with:

   ```
   Coveo.CES.Web.Search.Providers.ICESUserIdentityFactory factory =
   Coveo.CES.Web.Search.Providers.SearchProviderFactory.CreateDefaultUserIdentityFac
   tory();
   ```

2. Create the user or group identity to inject by calling the `CreateSecurityProviderUser` method:

   ```
   Coveo.CES.Web.Search.Providers.IUserIdentity userOrGroup =
   factory.CreateSecurityProviderUser(string p_UserOrGroupName, string
   p_SecurityProviderName, bool p_IsGroup, byte[] p_Data);
   ```

   Parameters:
   - `p_Name` (string): The name of the user or group.
   - `p_Provider` (string): The name of the security provider as defined in the Administration Tool.
   - `p_IsGroup` (bool): If set to true, a group identity is created. If set to false, a user identity is created.
   - `p_Data` (byte[]) : Additional data to pass to the security provider.

3. Register the additional identity:

   ```
   p_Args.AdditionalIdentities.Add(userOrGroup);
   ```

Integrating a Coveo .NET Search UI in Confluence

You can integrate a Coveo .NET search interface in a non-ASP.NET web site such as a Java application like Confluence by bootstraping the

search interface. A reverse proxy web application is used to manage the queries sent to and the results received from the Coveo index server where the Confluence content is indexed.

On the Coveo .NET Front-End Server

Website Setup

1. In IIS, create a new website, for example `WebsiteName` with a domain name (`www.examplesite.com`).
2. Right click on your new website (e.g. `WebsiteName`) and click **Add Virtual Directory**.
3. Under **Alias**, specify an alias name (e.g. `CESVirtualDirectoryName`). The physical path should be `[CES-installation-path]/Web`.
4. Right-click on the virtual directory you just created and click **Convert to Application**.

Custom Search Page

In order to be able to search with Confluence permissions, you will need to create a new search page in the following directory:

```
[Coveo Front-End installation path]\Web
```

For example, the new search page could be called `ConfluenceSearch.aspx`.

To retrieve the username from a query string (this is the how the identity is passed by the Confluence `search.vmd` decorator below) and add it to the query, add the following code to your search page:

ConfluenceSearch.aspx

```
<%@ Page language="c#" %>
<cnlm:Doctype runat="server" />
<%-- Force IE to display in the highest compatibility mode possible that we tested...
--%>
<cnlm:XUACompatibleMetaElement Value="IE10" runat="server" />
<html style="overflow-y: scroll">
<head runat="server">

</head>
<body style="margin: 0px;">
 <script runat="server">
  void h_OverrideUser(object p_Sender,
Coveo.CES.Web.Search.Controls.OverrideUserEventArgs p_Args)
  {
    Coveo.CES.Web.Search.Providers.ICESUserIdentityFactory factory =
Coveo.CES.Web.Search.Providers.SearchProviderFactory.CreateDefaultUserIdentityFactory(
);

    // Check if the user has logged in.
    if (Request.QueryString["username"] != null) {
   if (String.Compare(Request.QueryString["username"].ToString(), "$remoteUser.name",
true) != 0) {
    // Get the username from the query string.
    string userName = Request.QueryString["username"].ToString();
    // The name of the security provider defined under the Admin Tool.
      string securityProviderName = "Confluence Developers.coveo.com";
    Coveo.CES.Web.Search.Providers.IUserIdentity user =
factory.CreateSecurityProviderUser(userName, securityProviderName, false, null);
    p_Args.AdditionalIdentities.Add(user);
   }
    }

    // Always add Everyone (S-1-1-0) to always show anonymous results
    Coveo.CES.Web.Search.Providers.IUserIdentity everyoneGroup =
factory.CreateSecurityProviderUser("S-1-1-0", "Active Directory", true, null);
    p_Args.AdditionalIdentities.Add(everyoneGroup);
  }
 </script>
<form id="f" runat="server">
 <ces:searchhub name="Confluence" id="h" runat="server"
OnOverrideUser="h_OverrideUser" />
</form>
</body>
</html>
```

Reverse Proxy

Because Confluence runs under Apache Tomcat, a reverse proxy on the Confluence server is necessary to bootstrap a CES instance (that is hosted on another server).

2014-02-11 NBernier reports that he is now using this proxy http://j2ep.sourceforge.net/ so we do not have to support it. He will supply details when he is back.

The sources for the reverse proxy web application are available here.

1. Unzip the `.war` file to edit its content. In `web-inf\web.xml`, the URLs in `param-value` of `CESHosts` needs to be the URL of the CES web application on the IIS server.

```
<context-param>
  <param-name>CESHosts</param-name>

<param-value>http://www.examplesite.com/CESVirtualDirectoryName/;http://www.examp
lesite.com/CESVirtualDirectoryName/</param-value>
 </context-param>
```

2. In `SampleEmbedding.html`, adjust the virtual directory path passed in the first argument to the `Coveo.CNL.Web.Scripts.Ajax.Bootstrap.insertAjaxAspNetPage` method.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
</head>
<body>
  <iframe id="_historyFrame" src="empty.htm" style="display:none" ></iframe>
  <script type="text/javascript"
src="http://www.examplesite.com/CESVirtualDirectoryName/Coveo/default.aspx?k=embe
dding"></script>
  <div id="here" style="border: 2px solid gray; padding: 10px">Loading... </div>
  <script>

Coveo.CNL.Web.Scripts.Ajax.Bootstrap.insertAjaxAspNetPage("/CESVirtualDirectoryNa
me/", document.getElementById('here'));
  </script>
</body>
</html>
```

3. When the configuration of the web application is completed, compress the reverse proxy web application by executing this command from inside the `CESVirtualDirectoryName` folder:

```
jar -cvf CESVirtualDirectoryName.war *
```

Important
The IIS virtual directory for CES must have the same name as the `.war` file (`CESVirtualDirectoryName` in this example).

On the Confluence Server

1. Copy the webapp archive (e.g. `CESVirtualDirectoryName.war`) to the following folder (Confluence installation path may differ):
   `[Confluence-installation-path]\webapps\`
   (For example `C:\Program Files\Atlassian\Confluence\webapps`)

2. Restart confluence by executing the following commands:
   `[Confluence-installation-path]\bin\shutdown.bat`
   `[Confluence-installation-path]\bin\startup.bat`

3. The following URL should now lead to the Coveo search default page `http://youconfluencesite/CESVirtualDirectoryName/`

4. The next step is to modify the Confluence search page to use the Coveo .NET UI bootstraper. Replace the content of the file:
   `C:\Program Files\Atlassian\Confluence\confluence\decorators\search.vmd`
   with the following:

```
<html>
<head>
  <title>$action.getText("title.search.all")</title>
</head>
<body>
  <iframe id="_historyFrame" src="empty.htm" style="display:none" ></iframe>
  <script type="text/javascript"
src"=http://www.examplesite.com/CESVirtualDirectoryName/Coveo/?k=embedding"></scr
ipt>
  <div id="here" style="padding: 10px">
 <div style="text-align:center;">
  <img src="http://api.developers.coveo.com/images/wait.gif">
 </div>
  </div>

  <script type="text/javascript">
 var username = "$remoteUser.directoryId";
 var queryString = "$generalUtil.escapeXml($!queryString)";

 if ( queryString !== "" ) {

Coveo.CNL.Web.Scripts.Ajax.Bootstrap.insertAjaxAspNetPage("/CESVirtualDirectoryNa
me/ConfluenceSearch.aspx?q=$generalUtil.escapeXml($!queryString)&username="+usern
ame,document.getElementById('here'), true);
 } else {

Coveo.CNL.Web.Scripts.Ajax.Bootstrap.insertAjaxAspNetPage("/CESVirtualDirectoryNa
me/ConfluenceSearch.aspx?q=%40uri&username="+username,document.getElementById('he
re'), true);
 }
  </script>
</body>
</html>
```

The Confluence search page should now be replaced with the the Coveo search page.

Important Considerations

- To use another interface than the default one, specify another file than `default.aspx` in the bootstrap call.
- In the title of the HTML file, `$action.getText(....)` retrieves the query string (the syntax in Confluence is "`queryString=`" and not "q="). This query string is sent in the bootstrap call.

Maintaining Facet Selections Between Search Interfaces

Whenever you switch to a different search interface, your previous facet selections are lost.

You may want to change this behavior, for example to be able to search within the same collection for files of type PDF, no matter what search interface is selected (All Content, Intranet, etc.). One way to do it is to force a search interface to re-use the facet selections of the previously selected search interface.

An `ImportSettings` event is exposed on the `SearchControl` object. You must define a handler function which applies the facet settings of the previously selected search interface to the current one. That handler function must have the following signature:

```
public delegate void ImportSettingsEventHandler(object p_Sender, ImportSettingsEventArgs p_Args)
```

The `ImportSettingsEventArgs` argument contains 2 properties:

- The `From` property, which is a `SearchControl` object. It refers to the previously active (selected) search interface. This is where previous facet settings come from.

- The `Mini` property, which is a `bool`. It indicates whether the current search interface is going to execute a mini query. But it is not particularly relevant in our context.

To maintain the facet selections of the previously selected search interface:

1. On the Coveo Front-End server, go to the skin folder that corresponds to the search interface in which you want to apply the same facet selections.
   You can find the skin used by a search interface from the Interface Editor (**Search Interfaces** tab -> **Features** menu -> **General** page).
   The default folder for the skin files is: `[.Net_Front-End_Path]\Web\Coveo\Skins\[Search_Interface_Skin]`.

   > You are strongly advised to make a copy of a skin before modifying it, the reason being that installing a more recent .NET Front-End will not update it properly.

2. Using a text editor:
   a. In the skin folder, open the `CoveoSearch.ascx` file.
   b. At the end of the file, paste the following code excerpt that allows to keep the same facet selections as the previously selected search interface:

```
<script runat="server">
protected override void OnInit(EventArgs p_Args)
{
    ImportSettings += new
ImportSettingsEventHandler(SearchControl_ImportSettings);
    base.OnInit(p_Args);
}
private void SearchControl_ImportSettings(object sender,
ImportSettingsEventArgs p_Args)
{
    // Get the facet settings of the previously selected search interface,
then apply it to the current
    // search interface.
    IDictionary<string, SearchSetting> settings =
p_Args.From.State.Effective.Dictionary;
    foreach (SearchSetting setting in settings.Values) {
        // Make sure that we only apply facet-related settings.
        if (setting is FieldValueFacetSetting) {
            State.Staging.Add(setting);
        }
    }
}
</script>
```

   c. Save the file.
3. In the modified search interface, validate that facet selections are kept from any previously selected search interface.
4. Repeat the steps above for any other search interface that must behave in the same way.

Opening the Original Document Associated with a Mobile Search Result

Whenever you perform a query in a mobile search interface, every search result points to a cached copy of the original document. That copy is stored in the Coveo index.

On most mobile devices, search results appear like this:

On iPad devices, search results appear like this:

Since a cached copy is displayed when you click on a search result, it is not guaranteed to be an up-to-date representation of the original document. One way to circumvent it is to make search results pointing to the original documents instead.

A `TweakUri` event is exposed on the `QueryControl` object. To open the original document, all you have to do is implement a handler which overrides the URI pointed to by the search result. That handler must have the following signature:

```
public delegate void TweakUriEventHandler(object p_Sender, TweakUriEventArgs p_Args)
```

The `TweakUriEventArgs` argument contains three properties:

- The `Result` property, which is a `ResultInfo` object. It describes an individual search result.
- The `Uri` property, which is a `string`. It is used to customize the URI pointed to by a search result.
- The `Frame` property, which is a `string`. It is used to specify a window or frame where the document should be opened. Set this value to `"_blank"` to open the document in a new browser tab.

For obvious reasons, the proposed method will not work for search results whose URIs are unreachable. Additionally, it may also not work for content that is not web-based, the reason being that a native reader for that content may not exist on the mobile device that is used.

Note:
The `TweakUri` event handler is available starting with CES 6.0.3551+.

To make search results pointing to the original documents on mobile devices:

1. On the Coveo Front-End server, go to the skin folder that corresponds to the mobile search interface.
   You can find the skin used by a search interface from the Interface Editor (**Search Interfaces** tab -> **Features** menu -> **General** page).
   The default folder for the mobile skin files is: `[.Net_Front-End_Path]\Web\Coveo\Skins\[Mobile_Search_Interface_Skin]`.

   You are strongly advised to make a copy of a skin before modifying it, the reason being that installing a more recent .NET Front-End will not update it properly.
2. Using a text editor:
   a. In the skin folder, open the `ResultsPanel.ascx` file.
   b. At the end of the file, paste the following content that makes search results pointing to the original documents.

Code:

```
<script runat="server">
protected override void OnInit(EventArgs p_Args)
{
    // Customize the URIs pointed to by search results.
    TweakUri += this.Query_TweakUri_DoNotOpenMobileQuickView_UseDefaultUri;
    base.OnInit(p_Args);
}

private void Query_TweakUri_DoNotOpenMobileQuickView_UseDefaultUri(object
p_Sender, TweakUriEventArgs p_Args)
{
    p_Args.Uri = p_Args.Result.ResultObject.ClickUri;
}
</script>
```

c. Save the file.

3. In the modified search interface, validate that search results now point to the original documents.

When non-mobile search interfaces are used on iPad devices, to make search results pointing to the original documents:

1. On the Coveo Front-End server, go to the skin folder that corresponds to the non-mobile search interface.
   You can find the skin used by a search interface from the Interface Editor (**Search Interfaces** tab -> **Features** menu -> **General** page).
   The default folder for the skin files is: `[.Net_Front-End_Path]\Web\Coveo\Skins\[Search_Interface_Skin]`.

   > You are strongly advised to make a copy of a skin before modifying it, the reason being that installing a more recent .NET Front-End will not update it properly.

2. Using a text editor:
   a. In the skin folder, open the `ResultsPanel.ascx` file.
   b. At the end of the file, paste the following content that makes search results pointing to the original documents (on iPad devices only).

Code:

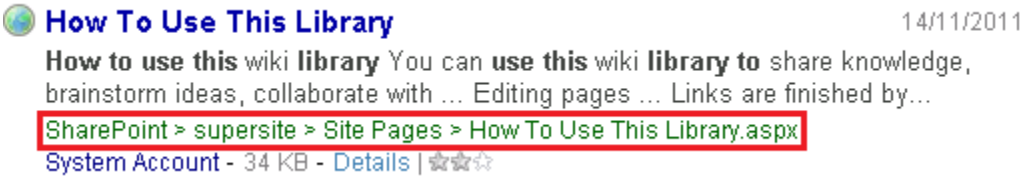
```
<script runat="server">
protected override void OnInit(EventArgs p_Args)
{
    // Customize the URIs pointed to by search results (specifically for
iPad devices).
    if (Browser.IsIpad) {
        TweakUri +=
this.Query_TweakUri_DoNotOpenMobileQuickView_UseDefaultUri;
    }
    base.OnInit(p_Args);
}

private void Query_TweakUri_DoNotOpenMobileQuickView_UseDefaultUri(object
p_Sender, TweakUriEventArgs p_Args)
{
    p_Args.Uri = p_Args.Result.ResultObject.ClickUri;
}
</script>
```

c. Save the file.

3. In the modified search interface, validate that search results now point to their original document.

Tweaking Breadcrumbs

Breadcrumbs are used in the .NET Search UI to show the hierarchy of a search result within the repository it comes from. They are composed of one or multiple segments. They generally appears below a search result like this:



Sometimes it can be convenient to tweak them:

- To include additional information in the hierarchy of their associated search result.
- To simply customize the caption and URI of their existing segments.

> Breadcrumbs do not automatically reflect the modifications done on a search result URI via an Alternate URI. This is because an Alternate URI has the ability to dynamically modify the URI that is shown in the Search UI, without actually modifying the search result itself. Indeed, this can cause an inconsistency between URIs and breadcrumbs, and one way to circumvent is through the use of the `TweakBreadcrumb` event handler to readjust the URI related to each segment.

A `TweakBreadcrumb` event is exposed on both the `QueryControl` and the `SearchControl` objects. To customize breadcrumbs, you must define a handler which modifies the proper segments. That handler function must have the following signature:

```
public delegate void TweakBreadcrumbEventHandler(object sender, TweakBreadcrumbEventArgs args)
```

The `TweakBreadcrumbEventArgs` argument contains 2 properties:

- The `Result` property, which is a `ResultInfo` object. It refers to the search result associated with the current set of breadcrumbs.
- The `Segments` property, which is a list of `TweakBreadcrumbEventArgs.PathSegment` objects. A `TweakBreadcrumbEventArgs.PathSegment` object describes the appearance of an individual breadcrumb segment. You can customize its caption, its URI, the character used to separate it from other breadcrumbs ('>' by default) and its layout priority.

Note:
The `TweakBreadcrumbEventArgs` event is available starting with the Coveo .NET Front-End 12.0.371+ October 2013 monthly release.

To add a new breadcrumb segment to search results:

1. On the Coveo Front-End server, go to the skin folder that corresponds to the search interface in which you want to customize breadcrumbs.
   You can find the skin used by a search interface from the Interface Editor (**Search Interfaces** tab -> **Features** menu -> **General** page).
   The default folder for the skin files is: `[.Net_Front-End_Path]\Web\Coveo\Skins\[Search_Interface_Skin]`.

   > You are strongly advised to make a copy of a skin before modifying it, the reason being that installing a more recent .NET Front-End will not update it properly.

2. Using a text editor:
   a. In the skin folder, open the `CoveoSearch.ascx` file.
   b. At the end of the file, paste the following content that adds a new **Home** breadcrumb segment pointing to http://www.coveo.com/ for every search result:

Code: adding a new breadcrumb segment

```
<script runat="server">
protected override void OnInit(EventArgs p_Args)
{
    base.OnInit(p_Args);
    TweakBreadcrumb += new
TweakBreadcrumbEventHandler(SearchControl_TweakBreadcrumb);
}
private void SearchControl_TweakBreadcrumb(object sender,
TweakBreadcrumbEventArgs args)
{
    // Insert a brand new breadcrumb segment before all other segments (for
every search result).
    TweakBreadcrumbEventArgs.PathSegment segment = new
TweakBreadcrumbEventArgs.PathSegment();
    segment.Caption = "Home";
    segment.Separator = " > ";
    segment.Uri = "http://www.coveo.com/";
    args.Segments.Insert(0, segment);
}
</script>
```

   c. Adapt the pasted code according to you want to do.
   d. Save the file.
3. In the modified search interface, validate that you now see a new **Home** breadcrumb segment which points to http://www.coveo.com/ for every search result.

The event handler is executed for every search result. To modify the breadcrumb of specific search results, you can filter search results by their URI. Use the `TweakBreadcrumbEventArgs.Result.ResultObject.Uri` property to perform such a filtering operation.

To customize the breadcrumb segments of specific search results:

1. On the Coveo Front-End server, go to the skin folder that corresponds to the search interface in which you want to customize breadcrumbs.
   You can find the skin used by a search interface from the Interface Editor (**Search Interfaces** tab -> **Features** menu -> **General** page).
   The default folder for the skin files is: `[.Net_Front-End_Path]\Web\Coveo\Skins\[Search_Interface_Skin]`.

   You are strongly advised to make a copy of a skin before modifying it, the reason being that installing a more recent .NET Front-End will not update it properly.
2. Using a text editor:
   a. Open the `CoveoSearch.ascx` file from the skin folder.
   b. At the end of the fileTweakBreadcrumbEventArgs.Result.ResultObject.Uri, paste the following content that replaces the **supersite** caption by **Intranet** and make it pointing to http://intranet/ :

Code: customizing a specific breadcrumb segment

```
<script runat="server">
protected override void OnInit(EventArgs p_Args)
{
    base.OnInit(p_Args);
    TweakBreadcrumb += new
TweakBreadcrumbEventHandler(SearchControl_TweakBreadcrumb);
}
private void SearchControl_TweakBreadcrumb(object sender,
TweakBreadcrumbEventArgs args)
{
    // Modify the properties of an existing breadcrumb segment for a search
result whose Uri
    // contains the "supersite" substring.
    if (args.Result.ResultObject.Uri.ToLower().Contains("supersite")) {
        foreach (TweakBreadcrumbEventArgs.PathSegment segment in
args.Segments) {
            if (segment.Caption.Equals("supersite",
StringComparison.InvariantCultureIgnoreCase)) {
                segment.Caption = "Intranet";
                segment.Uri = "http://intranet/";
            }
        }
    }
}
</script>
```

   c. Adapt the pasted code according to what you want to do.
   d. Save the file.
3. In the search interface, validate that the original **supersite** breadcrumb caption is now renamed to **Intranet** and points to http://intranet/.

Using Query Functions in a .NET Search Interface

AVAILABLE IN: JUNE 2015 RELEASE

A query function is an expression that is evaluated at query time on every document. The result is stored in a dynamic numerical field on the documents. A dynamic field exists only for the duration of the query. The field can then be used for any purposes, like any conventional numerical fields: to filter search results, for ranking, for a facet, etc.

Example:
A useful query function would be to filter or rank search results by geolocation, to return the stores that are near the customer current location (longitude and latitude coordinate). Another example would be for currency conversions.

To use query functions in a .NET search interface, follow the following procedure. The example below is for showing only search results (e.g. stores) that are in a 10 km range from a given longitude and latitude coordinate. The example assumes that documents in the index have `@latitude` and `@longitude` fields.

1. Ensure you are using a Coveo .NET Front-End version that supports query functions (started to be supported in the 12.0 June 2015 release).
2. Using a text editor, open the `CoveoSearch.ascx` file of your skin.

Example:
For the `Default` skin: `C:\Program Files\Coveo .NET Front-End 12\Web\Coveo\Skins\Default\CoveoSearch.ascx`

3. At the end of the file, add the following script block:

Code Sample

```
<script runat="server">
protected override void OnInit(EventArgs p_Args)
{
    base.OnInit(p_Args);
    SetupSearchBuilder += new
SetupSearchBuilderEventHandler(Search_SetupSearchBuilder);
}
void Search_SetupSearchBuilder(object p_Sender, SetupSearchBuilderEventArgs
p_Args)
{
    // Use a means of your choice to get the reference latitude and longitude.
    string refLat = "37.77919";
    string refLng = "-122.41914";
    string radius = "10000"; // Value in meters

    // Create a geolocation query function that will store the result in a
dynamic field named "distance".
    Coveo.CES.Web.Search.Providers.QueryFunction queryFct = new
Coveo.CES.Web.Search.Providers.QueryFunction();
    queryFct.Function = "dist(@latitude,@longitude," + refLat + "," + refLng +
")";
    queryFct.FieldName = "distance";
    p_Args.Builder.QueryFunctions.Add(queryFct);

    // Add a query expression that will filter results using the geolocation
function.
    p_Args.Builder.AddAdvancedExpression("@distance<" + radius);
}
</script>
```

Customizing Cloud Usage Analytics Events in a Coveo .NET Front-End Search Page

AVAILABLE IN: JUNE 2015 RELEASE

Since June 2015, a Coveo .NET Front-End search page can be configured to send search usage information to the Coveo Usage Analytics cloud service, as mentioned in the Coveo .NET Front-End First Time Setup online help page.

In some cases, it may be helpful to customize the analytics event data before it is pushed to the cloud service. For example, for a Coveo .NET search page integrated to a certain site, it might be necessary to transform the analytics events in some way to make them anonymous. As another example, it might be useful to add custom data (name and value pairs) to analytics events that could then be used to build cloud analytics dashboards.

Customizing analytics event data can be done by handling the `CustomizeCloudEvent` event on the `SearchHub` control.

The sample below obfuscates the user name by computing a hash on it. This allows preventing the real user names from being pushed to the cloud service and visible in the analytics dashboards, therefore making the usage information anonymous. The sample also adds the hypothetical version number of the site to which the Coveo .NET search page is integrated. The sample script block should be added to the ASPX file where the `SearchHub` control has been added.

Example:
The default search ASPX page is: `C:\Program Files\Coveo .NET Front-End 12\Web\default.aspx`

The script block assumes that the ID of the `SearchHub` control is "h".

```
<script runat="server">
    protected override void OnInit(EventArgs e)
    {
        h.CustomizeCloudEvent += CustomizeEvent;
        base.OnInit(e);
    }
    private static void CustomizeEvent(object p_Sender,
Coveo.CES.Web.Search.Analytics.CustomizeCloudEventArgs p_Args)
    {
        p_Args.CloudEvent.Username = ObfuscateName(p_Args.CloudEvent.Username);
        p_Args.CloudEvent.UserDisplayName =
ObfuscateName(p_Args.CloudEvent.UserDisplayName);
        p_Args.CloudEvent.CustomData.Add("siteVersion", "4.5");
    }
    private static string ObfuscateName(string p_Name)
    {
        if (String.IsNullOrEmpty(p_Name)) {
            return p_Name;
        } else {
            using (var sha1 = new System.Security.Cryptography.SHA1Managed()) {
                byte[] textData = Encoding.UTF8.GetBytes(p_Name);
                byte[] hash = sha1.ComputeHash(textData);
                return BitConverter.ToString(hash).Replace("-", String.Empty);
            }
        }
    }
</script>
```

Note that the `CloudEvent` property in the CustomizeCloudEventArgs class points to a class inheriting from the CloudBaseEventData class, which one depends on the analytics event type.

Developing New Controls in Microsoft Visual Studio

When a customization is needed in a skin, it is often possible to simply modify some `.ascx` files in the skin directory and no further actions are needed. (Click here to see such customization examples.)

However, there are situations where the desired feature has such a high level of complexity that developing a new ASP.NET control is needed. Click here to see some search interface samples. The samples zip file contains some Microsoft Visual Studio custom control projects. They are already configured with the necessary references to the Coveo assemblies. One can look at them as a starting point, to understand quickly the principle and to see the potential.

Search Interface Samples

The search interface samples are packaged in a ZIP archive that you need to extract in the appropriate sub-directory where the Coveo .NET Front-End was installed.

To download and install the .NET UI samples:

1. Download the `CoveoWebSamples.zip` archive file.
2. Extract the content of the archive file in the `Web` sub-directory where the Coveo .NET Front-End was installed (for example: `C:\Program Files\Coveo .NET Front-End 12\Web\`).
   (If the `Samples` directory already exists, rename or delete it first.)

In the `Samples` sub-directory, there is a Visual Studio solution named `CoveoWebSamples.sln`. By opening it, you then have easily access to the source code of every sample. The source code and the accompanying files of every sample is located in its own sub-directory. You can explore and modify them. Most of the samples come with a test page that you can open in a browser. The name of the test page is the name of the sample prefixed by "Test". Let's suppose that you configured the Coveo .NET Front-End default search page on the port 8080. To open the

search page itself, you would type the address `http://localhost:8080/` in the browser address bar. To test the `AddingExpressionsToQuery` sample, you would have to specify the address `http://localhost:8080/Samples/AddingExpressionsToQuery/TestAddingExpressionsToQuery.aspx` in the browser address bar.

Some of the samples require their C# project to be compiled in Visual Studio first. The output directory of those projects is set to the `bin` directory of the Coveo .NET Front-End default search site. So they can be used from the test pages without having to copy the DLL files manually. They also automatically reference the Coveo .NET Front-End assemblies.

> ⚠ When you compile the samples projects, make sure to compile them in the same bittage than your Coveo .NET Front-End installation (e.g. 64-bit).

AddingExpressionsToQuery Sample

This topic describes the AddingExpressionsToQuery sample that was made for the Coveo .NET Front-End.

Viewing the Sample

When the samples are installed (see Search Interface Samples), the search page to test this sample can be opened in a browser using the following path starting from the Coveo .NET Front-End search site:
`[http://localhost:8080/] Samples/AddingExpressionsToQuery/TestAddingExpressionsToQuery.aspx`

The ASPX Page

This sample is made up of a single ASPX test page named `TestAddingExpressionsToQuery.aspx`. It contains a `SearchHub` control to show a Coveo .NET search UI, along with 2 buttons to enable or disable a test query expression.

```
<body style="margin:0px">
  <form id="f" runat="server">
    <asp:Button id="BTNToggleOn" Text="Toggle On" runat="server"/>
    <asp:Button id="BTNToggleOff" Text="Toggle Off" runat="server"/>

    <ces:SearchHub runat="server"/>
  </form>
</body>
```

Registering the Button Click Events

As usually for ASP.NET button controls, for the `Click` event to be called, it must be hooked up at the `Init` stage.

```
protected override void OnInit(EventArgs p_Args)
    {
        BTNToggleOn.Click += new EventHandler(this.BTNToggleOn_Click);
        BTNToggleOff.Click += new EventHandler(this.BTNToggleOff_Click);

        base.OnInit(p_Args);
    }
```

Adding the Filter

When the user clicks on the **Toggle On** button, a query expression on the `@sysfiletype` field is added to restrict the documents returned by the Coveo index server. The `Reset` method on the `SearchControl` object is also called to make the change effective.

```
private void BTNToggleOn_Click(object p_Sender, EventArgs p_Args)
    {
        SearchControl search = SearchBinding.MainSearchObject;
        search.AddQueryExpression("SearchOnlyForWeb", "@sysfiletype=html");

        // This will trigger a new search. Otherwise, the change wouldn't
        // have been effective until the next time the user clicked Search.
        search.Reset();
    }
```

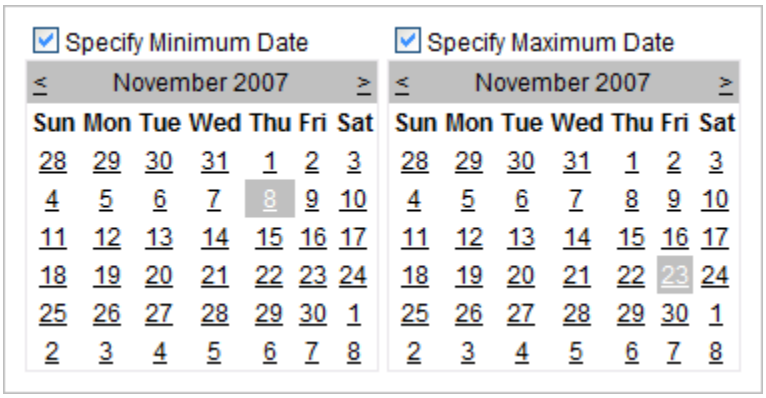
Removing the Filter

When the user clicks on the **Toggle Off** button, the query expression on the `@sysfiletype` field is removed from the staging search settings. The `Reset` method on the `SearchControl` object is also called here to make the change effective.

```
private void BTNToggleOff_Click(object p_Sender, EventArgs p_Args)
    {
        SearchControl search = SearchBinding.MainSearchObject;
        search.RemoveQueryExpression("SearchOnlyForWeb");
        search.Reset();
    }
```

CalendarDateSearch Control Sample

This topic describes the `CalendarDateSearch` sample that was made for the Coveo .NET Front-End.



Viewing the Sample

When the samples are installed (see Search Interface Samples), the search page to test this sample can be opened in a browser using the following path starting from the Coveo .NET Front-End search site:
`[http://localhost:8080/] Samples/CalendarDateSearch/TestCalendarDateSearch.aspx`

C# Project Overview

The `CalendarDateSearch` project is made up of 2 classes:

- `CalendarDateSearch`:
  This class is the control that displays the calendar controls and their associated check-boxes. This class inherits from `BoundToSearch`.
- `CalendarDateSearchSetting`:
  This class represents its associated search setting object (see Search Settings). This class inherits from `SearchSetting`.

The `CalendarDateSearchSetting` class

Like most search controls, the `CalendarDateSearch` control uses a class derived from `SearchSetting` to store its state (see Search Settings). The `CalendarDateSearchSetting` class holds the state of the `CalendarDateSearch` control across postbacks, and of adding the proper expressions to the query when the `SearchControl` is preparing it to be executed by the Coveo index server.

Setting Name

Each search setting class needs to have a unique name that will be used for serializing and deserializing.

```
/// <summary>
/// The name of the <see cref="CalendarDateSearchSetting"/> setting.
/// </summary>
public const string NAME = "CalendarDateSearch";
```

State Properties

The `CalendarDateSearchSetting` uses 2 properties to store the control state: one holds the minimum restricted date, and the other the maximum restricted date. When one is set to `DateTime.MinValue`, it means that the associated check-box is not selected.

```
// The minimum date
private DateTime m_MinimumDate;
// The maximum date
private DateTime m_MaximumDate;

/// <summary>
/// Gets or sets the minimum date on which the results may have been modified.
/// </summary>
public DateTime MinimumDate
{
   get {
       return m_MinimumDate;
   }
   set {
       m_MinimumDate = value;
   }
}
/// <summary>
/// Gets or sets the maximum date on which the results may have been modified.
/// </summary>
public DateTime MaximumDate
{
   get {
       return m_MaximumDate;
   }
   set {
       m_MaximumDate = value;
   }
}
```

The `Empty` Property

Each search setting class must override the `Empty` property. This property should return `True` when the setting object does not contain any significant value.

```
public override bool Empty
{
   get {
       return  m_MinimumDate == DateTime.MinValue && m_MaximumDate ==
DateTime.MinValue;
   }
}
```

Constructors

A search setting typically implements two constructors: one of them is used when creating a new empty setting object, while the other is used when deserializing a setting object that was serialized in a previous a postback.

```
//*************************************************************************
/// <summary>
/// Initializes a new instance of <see cref="CalendarDateSearchSetting"/>.
/// </summary>
//*************************************************************************
public CalendarDateSearchSetting()
:   base(NAME)
{
}

//*************************************************************************
/// <summary>
/// Initializes a new instance of <see cref="CalendarDateSearchSetting"/>.
/// </summary>
/// <param name="p_Deserializer">The <see cref="StringDeserializer"/> from which to
read the data.</param>
//*************************************************************************
public CalendarDateSearchSetting(StringDeserializer p_Deserializer)
:   base(NAME, p_Deserializer)
{
   m_MinimumDate = p_Deserializer.GetDateTime();
   m_MaximumDate = p_Deserializer.GetDateTime();
}
```

The `Serialize` and `Deserialize` Methods

Each search setting class overrides a method called `Serialize` that must properly serialize the content of the setting.

```
public override void Serialize(StringSerializer p_Serializer)
{
   base.Serialize(p_Serializer);
   p_Serializer.Append(m_MinimumDate);
   p_Serializer.Append(m_MaximumDate);
}
```

Each search setting class also implements a `Deserialize` method that basically does the reverse of the `Serialize` method. The implementation here only calls the appropriate constructor overload described earlier.

```
public static SearchSetting Deserialize(StringDeserializer p_Deserializer,
                                        string p_Custom)
{
   return new CalendarDateSearchSetting(p_Deserializer);
}
```

The `Apply` Method

When a `SearchControl` object performs a search, it creates a `SearchBuilder` object that accumulates all the information about a query to execute. As part of this process, all the effective search setting objects have their `Apply` method called to supply their own part of the query (see Query Execution Process Overview).

```
public override void Apply(SearchBuilder p_Builder,
                           SearchControl p_Search)
{
   if (m_MinimumDate != DateTime.MinValue) {
       string expr = CustomFields.SYSDATE + ">=" +
SearchUtilities.DateToString(m_MinimumDate);
       p_Builder.AddAdvancedExpression(expr);
   }
   if (m_MaximumDate != DateTime.MinValue) {
       string expr = CustomFields.SYSDATE + "<=" +
SearchUtilities.DateToString(m_MaximumDate);
       p_Builder.AddAdvancedExpression(expr);
   }
}
```

If either a minimum or maximum date is specified, the code appends a query expression to restrict the search to documents that were modified in the specified time range. Dates need to be converted to string using `SearchUtilities.DateToString` in order to ensure that the correct format is used.

Registering the Setting Class in the `SearchSettingFactory`

Each search setting class must be registered with the `SearchSettingFactory` when the Web application initializes, for a mapping to be established between its search setting name and the method to deserialize it. In our case, the registration of the `CalendarDateSearchSettin g` is done in the static constructor of the `CalendarDateSearch` class.

```
static CalendarDateSearch()
{
   // Register our setting with the factory
   SearchSettingFactory.RegisterSetting(CalendarDateSearchSetting.NAME, new
SearchSettingFactory.DeserializeDelegate(CalendarDateSearchSetting.Deserialize));
}
```

The `CalendarDateSearch` Control

The `CalendarDateSearch` class inherits from `BoundToSearch`**.** This base class makes it possible to interact with the `SearchControl` object. The class also implements the `IUpdateOnStagingChange` marker interface, so that its rendered HTML is properly updated when the staging settings change (see AJAX Requests and SearchUpdatePanel Control).

```
public class CalendarDateSearch : BoundToSearch,
                                  IUpdateOnStagingChange,
                                  INamingContainer
```

The LoadSettings Event

Whenever the CalendarDateSearch control has to load its state from its associated CalendarDateSearchSetting object, the SearchControl object fires the LoadSettings event. For the LoadSettings event to be called, it must first be registered at the Init stage.

```
protected override void OnInit(EventArgs p_Args)
    {
        SearchObject.LoadSettings += this.Search_LoadSettings;
        // . . .
        base.OnInit(p_Args);
    }
```

The LoadSettings event handler grabs its CalendarDateSearchSetting object from the Staging settings collection, and modifies the check-boxes and the calendar controls to correctly reflect the state. Note that the setting may not be present in the collection.

```
private void Search_LoadSettings(object  p_Sender,
                                 EventArgs p_Args)
{
   CalendarDateSearchSetting setting = (CalendarDateSearchSetting)
SearchObject.State.Staging[CalendarDateSearchSetting.NAME];
   // Update the controls depending of the current setting, if any
   if (setting != null) {
       if (setting.MinimumDate != DateTime.MinValue) {
           CHKUseMinimumDate.Checked = true;
           CALMinimumDate.Enabled = true;
           CALMinimumDate.SelectedDate = setting.MinimumDate;
       } else {
           CHKUseMinimumDate.Checked = false;
           CALMinimumDate.Enabled = false;
       }
       if (setting.MaximumDate != DateTime.MinValue) {
           CHKUseMaximumDate.Checked = true;
           CALMaximumDate.Enabled = true;
           CALMaximumDate.SelectedDate = setting.MaximumDate;
       } else {
           CHKUseMaximumDate.Checked = false;
           CALMaximumDate.Enabled = false;
       }
   } else {
       CHKUseMinimumDate.Checked = false;
       CALMinimumDate.Enabled = false;
       CHKUseMaximumDate.Checked = false;
       CALMaximumDate.Enabled = false;
   }
}
```

Creation of the Child Controls

The control uses 2 check-boxes as child controls, as well as 2 `Calendar` controls. Those are declared as private members. One of the calendars is for the minimum modified date, and the other one is for the maximum date. Each calendar is associated to a check-box that enables or disables it.

```
// The child ASP.NET controls that we use
private CheckBox CHKUseMinimumDate;
private Calendar CALMinimumDate;
private CheckBox CHKUseMaximumDate;
private Calendar CALMaximumDate;
```

The child controls are instantiated in the `CreateChildControls` method (inherited from `Control`), which is automatically called by the ASP.NET Framework when the child controls has be created.

```
protected override void CreateChildControls()
{
   CHKUseMinimumDate = new CheckBox();
   CHKUseMinimumDate.ID = "CHKUseMinimumDate";
   CHKUseMinimumDate.Text = "Specify Minimum Date";
   CHKUseMinimumDate.AutoPostBack = true;
   CHKUseMinimumDate.CheckedChanged += this.CHKUseMinimumDate_CheckedChanged;
   Controls.Add(CHKUseMinimumDate);

   CALMinimumDate = new Calendar();
   CALMinimumDate.ID = "CALMinimumDate";
   CALMinimumDate.SelectedDate = DateTime.Today;
   CALMinimumDate.SelectionChanged += this.CALMinimumDate_SelectionChanged;
   Controls.Add(CALMinimumDate);

   CHKUseMaximumDate = new CheckBox();
   CHKUseMaximumDate.ID = "CHKUseMaximumDate";
   CHKUseMaximumDate.Text = "Specify Maximum Date";
   CHKUseMaximumDate.AutoPostBack = true;
   CHKUseMaximumDate.CheckedChanged += this.CHKUseMaximumDate_CheckedChanged;
   Controls.Add(CHKUseMaximumDate);

   CALMaximumDate = new Calendar();
   CALMaximumDate.ID = "CALMaximumDate";
   CALMaximumDate.SelectedDate = DateTime.Today;
   CALMaximumDate.SelectionChanged += this.CALMaximumDate_SelectionChanged;
   Controls.Add(CALMaximumDate);
}
```

On each control, the `AutoPostBack` property is set to `True`, so that ASP.NET automatically performs a postback each time the user changes something.

The `Render` Method

The `CalendarDateSearch` control overrides the `Render` method to render additional HTML markup between the child controls.

```
protected override void Render(HtmlTextWriter p_Writer)
{
   RenderBeginTag(p_Writer);
       p_Writer.RenderBeginTag("tr");
           p_Writer.RenderBeginTag("td");
               CHKUseMinimumDate.RenderControl(p_Writer);
               p_Writer.Write("<br/>");
               CALMinimumDate.RenderControl(p_Writer);
           p_Writer.RenderEndTag();
           p_Writer.RenderBeginTag("td");
               CHKUseMaximumDate.RenderControl(p_Writer);
               p_Writer.Write("<br/>");
               CALMaximumDate.RenderControl(p_Writer);
           p_Writer.RenderEndTag();
       p_Writer.RenderEndTag();
   RenderEndTag(p_Writer);
}
```

Updating the `CalendarDateSearchSetting` on a User's Action

Whenever the user modifies the value of a child control (e.g. selecting a date in a calendar), the `CalendarDateSearchSetting` object, that is persisted across postbacks by the `SearchControl` object, has to be updated. To do so, the sample hooks up to the proper child control events that signal when a value changes. This is done in the `CreateChildControls` method presented earlier in this topic. Whenever one of those events is fired, it is important to create a new `CalendarDateSearchSetting` that reflects the updated state, and to add it to the `SearchControl`'s `Staging` setting collection, overwriting any previous similar setting. This is done in a private method called `UpdateSearchSetting` that is called from every event handler.

```
private void UpdateSearchSetting()
{
   // Build a CalendarDateSearchSetting from the content of our child controls
   CalendarDateSearchSetting setting = new CalendarDateSearchSetting();
   if (CHKUseMinimumDate.Checked) {
       setting.MinimumDate = CALMinimumDate.SelectedDate;
   }
   if (CHKUseMaximumDate.Checked) {
       setting.MaximumDate = CALMaximumDate.SelectedDate;
   }
   // If the setting is non-empty, add, otherwise clear any existing one
   if (!setting.Empty) {
       SearchObject.State.Staging.Add(setting);
   } else {
       SearchObject.State.Staging.Remove(CalendarDateSearchSetting.NAME);
   }
}

private void CHKUseMinimumDate_CheckedChanged(object  p_Sender,
                                              EventArgs p_Args)
{
   UpdateSearchSetting();
}

private void CHKUseMaximumDate_CheckedChanged(object  p_Sender,
                                              EventArgs p_Args)
{
   UpdateSearchSetting();
}

private void CALMinimumDate_SelectionChanged(object  p_Sender,
                                             EventArgs p_Args)
{
   UpdateSearchSetting();
}

private void CALMaximumDate_SelectionChanged(object  p_Sender,
                                             EventArgs p_Args)
{
   UpdateSearchSetting();
}
```

Displaying a Comment to the User When Search Results Are Being Restricted

It is often useful to display a message to the user to mention that search results are restricted by some criteria (modified dates in our particular case). This can be done by hooking up on the `GenerateComments` event fired by the `SearchControl` and adding a comment to the list of those to be displayed. A comment is represented by an `ClearComment` object, that stores the message to display along with a **Clear** link that is displayed next to the comment. Here is the related code for the `CalendarDateSearch` control.

```
private void Search_GenerateComments(object  p_Sender,
                                     GenerateCommentsEventArgs p_Args)
{
   // If there is an effective CalendarDateSearchSetting setting, we want
   // to generate a comment that will be displayed to the user.
   CalendarDateSearchSetting setting = (CalendarDateSearchSetting)
SearchObject.State.Staging[CalendarDateSearchSetting.NAME];
   if (setting != null && !setting.Empty) {
       // Create a new comment that will remove the CalendarDateSearchSetting when
cleared
       ClearComment comment = new ClearComment();
       comment.Settings.Add(setting);
       // Depending on what we're restricting on the comment will be phrased
differently
       if (setting.MinimumDate != DateTime.MinValue && setting.MaximumDate !=
DateTime.MinValue) {
           comment.Message = String.Format("Only results modified between {0} and {1}
will be displayed.", setting.MinimumDate.ToString("D",
System.Globalization.CultureInfo.CurrentCulture), setting.MaximumDate.ToString("D",
System.Globalization.CultureInfo.CurrentCulture));
       } else if (setting.MinimumDate != DateTime.MinValue) {
           comment.Message = String.Format("Only results modified after {0} will be
displayed.", setting.MinimumDate.ToString("D",
System.Globalization.CultureInfo.CurrentCulture));
       } else if (setting.MaximumDate != DateTime.MinValue) {
           comment.Message = String.Format("Only results modified before {0} will be
displayed.", setting.MaximumDate.ToString("D",
System.Globalization.CultureInfo.CurrentCulture));
       } else {
           // Should never happen!
       }
       p_Args.Controls.Add(comment);
   }
}
```

The `GenerateComments` event is wired up at the `Init` phase using the following code.

```
SearchObject.GenerateComments += this.Search_GenerateComments;
```

SearchInitialization Sample

This topic describes the SearchInitialization sample that was made for the Coveo .NET Front-End.

Viewing the Sample

When the samples are installed (see Search Interface Samples), the search page to test this sample can be opened in a browser using the following path starting from the Coveo .NET Front-End search site:
`[http://localhost:8080/] Samples/SearchInitialization/TestSearchInitialization.aspx`

The ASPX Page

This sample is made up of a single ASPX test page named `TestSearchInitialization.aspx`. It contains only a `SearchHub` control to show a Coveo .NET search UI.

```
<body style="margin:0px">
  <form id="f" runat="server">
    <ces:SearchHub runat="server"/>
  </form>
</body>
```

Registering and Implementing the `InitializeSettings` Event

The `InitializeSettings` event is called at the load of the search page to configure the initial search settings. This handler is not called when the user executes further actions in the search page like clicking in facets.

To be called by the `SearchControl` object, the `InitializeSettings` event must be registered at the `Init` stage.

```
protected override void OnInit(EventArgs p_Args)
    {
        SearchBinding.MainSearchObject.InitializeSettings += new
EventHandler(this.Search_InitializeSettings);
        base.OnInit(p_Args);
    }
```

This sample configures 2 initial search settings:

- It adds a filter to show only Word documents.
- It also makes the documents to be listed in descending order of modification date.

```
private void Search_InitializeSettings(object p_Sender, EventArgs p_Args)
    {
        SearchControl search = (SearchControl) p_Sender;

        // Initialize the Format control to Word
        FormatSetting format = new FormatSetting();
        format.FileType = FormatUtilities.WORD_FILETYPE;
        search.State.Staging.Add(format);

        // Change default sort order to By Date
        SortByFieldSetting sort = new SortByFieldSetting();
        sort.Field = "@sysdate";
        sort.Descending = true;
        search.State.Staging.Add(sort);
    }
```

⚠ To be consistent with the packaging of the other samples, this one is an ASPX test page. It does not provide its own search interface and skin. It rather simply displays the search interfaces configured with the Interface Editor. Therefore, the `InitializeSettings` event is registered on the `MainSearchObject` object, which is in this case the `SearchControl` object of the default search interface. It is generally better to add the `InitializeSettings` code in the skin (e.g. in `CoveoSearch.ascx`), as the code is often different from one search interface to another.

SettingUpSearchBuilder Sample

This topic describes the SettingUpSearchBuilder sample that was made for the Coveo .NET Front-End.

View the Sample

When the samples are installed (see Search Interface Samples), the search page to test this sample can be opened in a browser using the following path starting from the Coveo .NET Front-End search site:
`[http://localhost:8080/] Samples/SettingUpSearchBuilder/TestSettingUpSearchBuilder.aspx`

The ASPX Page

This sample is made up of a single ASPX test page named `TestSettingUpSearchBuilder.aspx`. It contains only a `SearchHub` control to show a Coveo .NET search UI.

```
<body style="margin:0px">
  <form id="f" method="get" runat="server">
    <ces:SearchHub runat="server"/>
  </form>
</body>
```

Registering and Implementing the `SetupSearchBuilder` Event

The `SetupSearchBuilder` event (either on the `SearchControl` or on the `QueryControl` objects) is called right before a query is sent to the index to be executed (no matter if it is the first postback or not). It allows modifying the query parameters to control which results will be returned by the Coveo index server.

To be called by the `SearchControl` object, the `SetupSearchBuilder` event must be registered at the `Init` stage.

```
protected override void OnInit(EventArgs p_Args)
    {
        SearchBinding.MainSearchObject.SetupSearchBuilder += new
SetupSearchBuilderEventHandler(Search_SetupSearchBuilder);
        base.OnInit(p_Args);
    }
```

This sample adds a filter expression to show only documents that have more than 5 pages.

```
private void Search_SetupSearchBuilder(object p_Sender, SetupSearchBuilderEventArgs
p_Args)
    {
        p_Args.Builder.AddAdvancedExpression("@syspages>5");
    }
```

⚠ To be consistent with the packaging of the other samples, this one is an ASPX test page. It does not provide its own search interface and skin. It rather simply displays the search interfaces configured with the Interface Editor. Therefore, the `InitializeSettings` event is registered on the `MainSearchObject` object, which is in this case the `SearchControl` object of the default search interface. It is generally better to add the `SetupSearchBuilder` code in the skin (e.g. in `CoveoSearch.ascx`), as the code is often different from one search interface to another.

SimpleParam Control Sample

This topic describes the SimpleParamControl sample that was made for the Coveo .NET Front-End.

Its purpose and its structure are pretty similar to the ones of the CalendarDateSearch Control Sample for i.e. it is made up of 2 classes:

- `SimpleParamControl`:

This class is a control that allows the user to specify some information (a simple text box in this case).

- `SimpleParamControlSetting`:
  This class represents its associated search setting object (see Search Settings). This class inherits from `SearchSetting`.

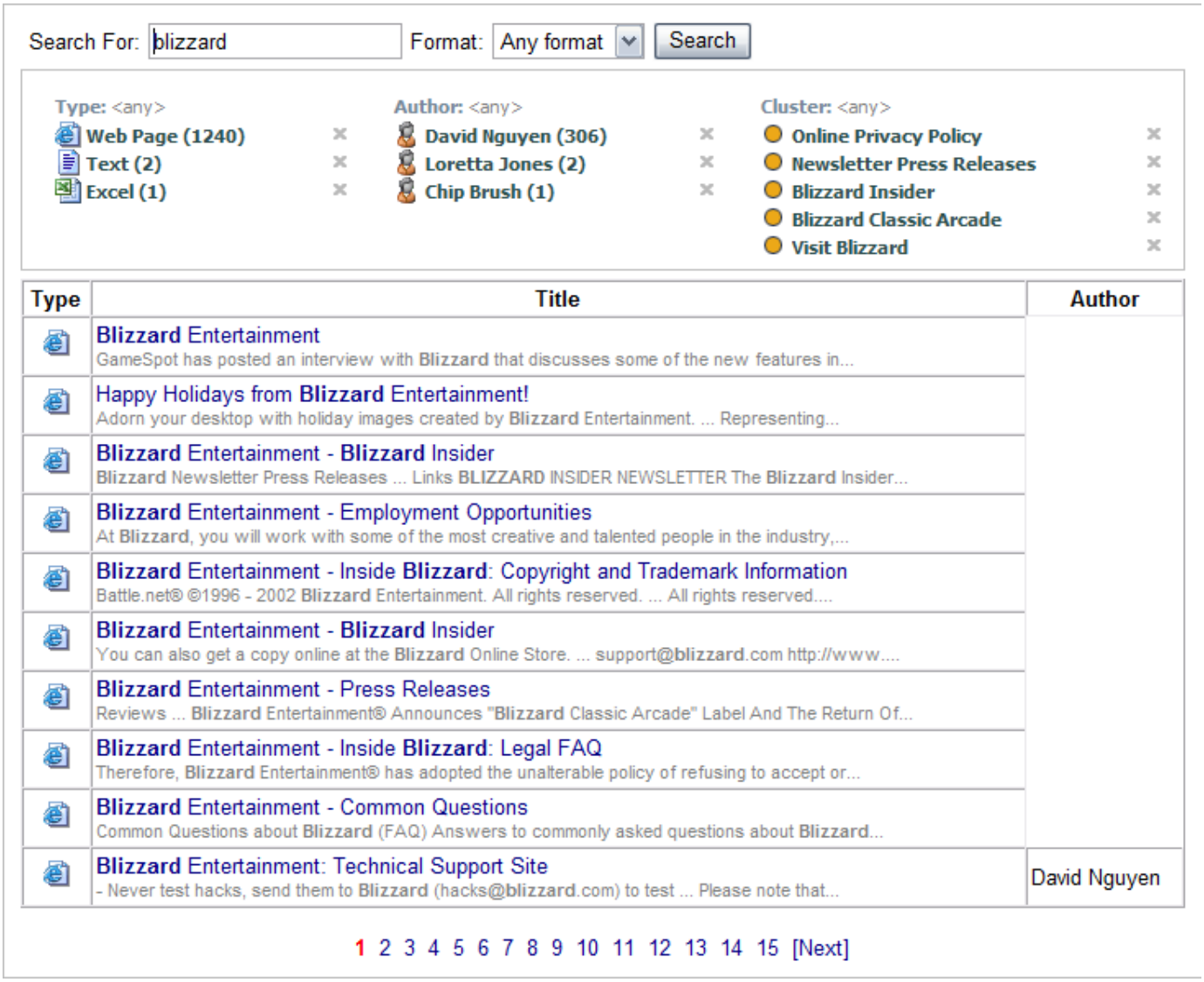
Please refer to the CalendarDateSearch Control Sample page for a description about how the sample works.

.

SkinFromScratch Sample

This topic describes the steps required to create a skin from scratch, instead of copying an existing Coveo built-in one and to start from it. The skin that this sample describes implements a simple search interface that displays results in a table, along with facets displayed at the top.

A skin consists of one or more user control (`.ascx`) files usually located in a directory under `[.NET_Front-End_Path]Web\Coveo\Skins`. This directory must contain at least a user control named `CoveoSearch.ascx`. This file may include other user controls, but this is not strictly necessary and is up to the skin developer. In this sample, all the code required is included directly in `CoveoSearch.ascx`.



Viewing the Sample

When the samples are installed (see Search Interface Samples), the search page to test this sample can be opened in a browser using the following path starting from the Coveo .NET Front-End search site:
`[http://localhost:8080/] Samples/SkinFromScratch/TestSkinFromScratch.aspx`

The skin files are located in the following directory:
`[.NET_Front-End_Path]\Web\Samples\SkinFromScratch\Skin`

To make it easier testing this sample, the skin files are loaded directly from the directory above (see the virtual path specified in `TestSkinFromScratch.aspx`). However, note that the good practice is to create the skin directory under: `[.NET_Front-End_Path]\Web\Coveo\Skins`

The `CoveoSearch.ascx` file

Control Directives

Typically, a user control file begins with a list of directives that will be processed by the ASP.NET runtime to determine: the base class for the user control, the referenced assemblies, the namespaces to import, mapping control libraries to prefixes, etc. Since the common Coveo directives necessary for a skin to work are added to the web.config file by the Coveo .NET Front-End install kit, the directive list of this sample is very short.

```
<%@ Control Language="c#" AutoEventWireup="false"
Inherits="Coveo.CES.Web.Search.Controls.SearchControl, Coveo.CES.Web.Search" %>
```

The Search Panel Controls

The sample skin provides a box to enter the query, a drop-down list that allows selecting among common file formats, as well as a search button. All these functionalities are implemented using stock controls provided by Coveo.

```
<table  cellspacing="5"  cellpadding="0">
     <tr>
       <td>
         Search For:
       </td>
       <td>
         <ces:Query  id="qry"  runat="server"/>
       </td>
       <td>
         Format:
       </td>
       <td>
         <ces:Format  runat="server"/>
       </td>
       <td>
         <ces:SearchButton  id="sbt"  runat="server"/>
       </td>
     </tr>
    </table>
```

The `ResultsPanel` Control

Note that all the controls related to a query output (result list, facets, pager, etc.) should be enclosed in a `ResultsPanel` control (which inherits from `QueryControl`).

```
<ces:ResultsPanel runat="server">
      . . .
    </ces:ResultsPanel>
```

The Facets

In this sample, the facets are placed in an horizontal fashion above the results.

```
<table width="100%" cellspacing="5" cellpadding="0" style="border: 1px solid silver;
margin-bottom: 5px">
        <tr>
          <td valign="top">
            <ces:FieldValueFacet Field="@sysfiletype" Title="Type" FacetZone="top"
runat="server" />
          </td>
          <td valign="top">
            <ces:FieldValueFacet Field="@sysauthor" Title="Author" FacetZone="top"
runat="server" />
          </td>
          <td style="width:80%">
          </td>
        </tr>
      </table>
```

This sample includes instances of the `FieldValueFacet` control directly in the skin, each bound to a specific field (`@sysfiletype` and `@sysauthor`). This works fine. But generally, a skin rather includes a `CustomRefineByFields` control which internally instantiates a series of `FieldValueFacet` controls that match the facets configured in the Interface Editor.

The Results

As usual, the results are rendered through the use of the `ResultList` control, specifying a header and a footer that render the opening and closing tags (an HTML table in this sample). Each result is rendered as a separate HTML table row.

```
<ces:ResultList  runat="server">
 <Header>
   <table  width="100%"  cellspacing="0"  cellpadding="2"  border="1">
     <tr>
       <th>
         Type
       </th>
       <th>
         Title
       </th>
       <th>
         Author
       </th>
     </tr>
 </Header>
 <ResultTemplate>
     <tr>
       <td  align="center">
         <ces:ResultFormatIcon  runat="server"/>
       </td>
       <td  valign="top">
         <div>
           <ces:ResultOpenLink  runat="server">
             <ces:ResultTitle  id="ttl"  runat="server"/>
           </ces:ResultOpenLink>
         </div>
         <div  style="font-size: 8pt; color: gray">
           <ces:ResultExcerpt  Length="100"  runat="server"/>
         </div>
       </td>
       <td>
         <ces:ResultAuthor  runat="server"/>
       </td>
     </tr>
 </ResultTemplate>
 <Footer>
   </table>
 </Footer>
</ces:ResultList>
```

The Pager

The skin also includes a `Pager` control, to allow the user jumping to other result pages.

```
<center>
  <ces:Pager id="pgr" CssClass="CesPager" ShowIfOnlyOnePage="false" Pages="5"
runat="server"/>
</center>
```

The `CoveoSearch.css` File

In addition to a `CoveoSearch.ascx` file, every skin also contains a file named `CoveoSearch.css`. As the file extension suggests, that file contains the display styles of the HTML elements. When an administrator modifies styles using the Interface Editor (**Search Interfaces** tab > **Styles** sub-tab), the changes made end up being saved in the `CoveoSearch.css` file of the skin.

The `CoveoSearch.css` file included in this sample contains a subset of the styles of the Coveo built-in skins. Some minor changes (e.g. background color) have also been made.

UsingWebService Sample

This topic describes the steps needed to perform queries through the Coveo .NET UI search web service using Microsoft Visual Studio.

Viewing the Sample

When the samples are installed (see Search Interface Samples), the search page to test this sample can be opened in a browser using the following path starting from the Coveo .NET Front-End search site:
`[http://localhost:8080/] Samples/UsingWebService/TestUsingWebService.aspx`

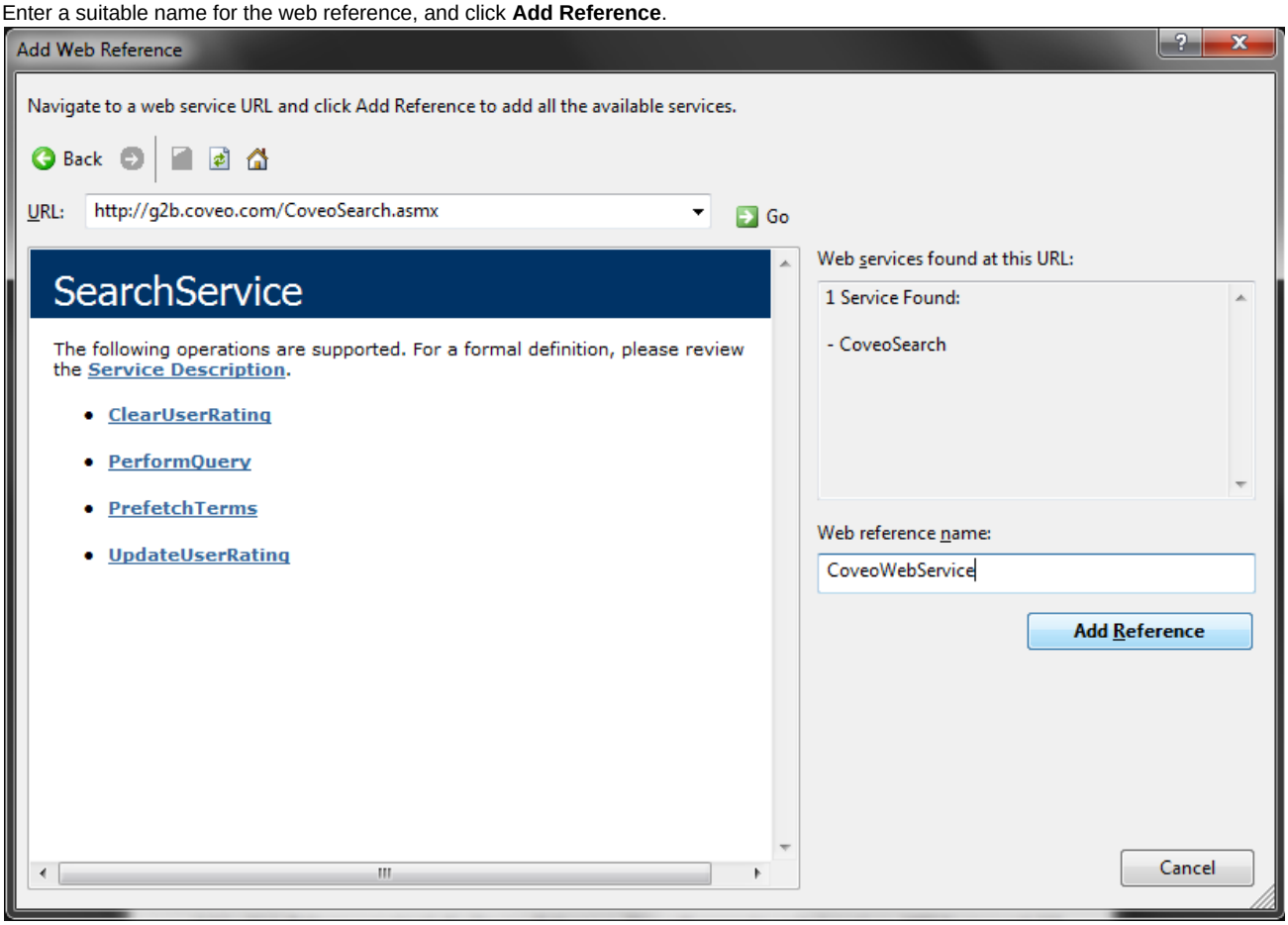
> The code of this sample is packaged as an ASPX test page. This is mainly for consistency with the other samples. Note that calling the search web service could also be useful from a non-web (e.g. Windows) application.

Adding a Reference to the Web Service

To use a web service in Microsoft Visual Studio, you must add a web reference to your project that points to the web service URL. Microsoft Visual Studio will then generate a set of .NET wrappers that you can use to call the web service in the same way than you would call a regular .NET object.

Follow these steps to add a web reference:

1. In the **Solution Explorer**, right click the **References** folder and select **Add Web Reference**.
   If you are using Visual Studio 2008 or higher, you must select **Add Service Reference**, click **Advanced**, and then **Add Web Reference**.
2. Enter the URL of the web service in the **URL** text box
3. Click **Go.**
   The list of available methods from the web service should be displayed.
4. Enter a suitable name for the web reference, and click **Add Reference**.



Creating the Web Service Object and Setting the Appropriate Credentials

To use the web service in C# code, you simply have to instantiate the wrapper class that was automatically generated, and call its methods that correspond to the ones exposed by the web service. If the website hosting the web service requires authentication (the default), you will have to specify credentials to the wrapper object.

Here is how the sample creates an instance of the web service wrapper and instructs it to use the user authenticated by IIS to access the test page to perform the query.

```
// This part is needed only for this sample to resolve the absolute address of the web
service (since this value depends on your installation).
        // In a real application using the web service the correct uri should have
been entered when creating the web reference, so there is no need to override it.
        UriBuilder webServiceUrl = new UriBuilder(Request.Url);
        webServiceUrl.Path = Path.GetDirectoryName(webServiceUrl.Path);
        webServiceUrl.Path = Path.GetDirectoryName(webServiceUrl.Path);
        webServiceUrl.Path = Path.GetDirectoryName(webServiceUrl.Path);
        webServiceUrl.Path = Path.Combine(webServiceUrl.Path, "CoveoSearch.asmx");
        webServiceUrl.Query = "";

        // Connect to the web service using Windows authentication.
        BasicHttpBinding binding = new BasicHttpBinding();
        binding.Security.Mode = BasicHttpSecurityMode.TransportCredentialOnly;
        binding.Security.Transport.ClientCredentialType =
HttpClientCredentialType.Windows;
        SearchServiceSoapClient service = new SearchServiceSoapClient(binding, new
EndpointAddress(webServiceUrl.ToString()));
        service.ClientCredentials.Windows.AllowedImpersonationLevel =
TokenImpersonationLevel.Impersonation;
        service.ChannelFactory.Credentials.Windows.ClientCredential =
CredentialCache.DefaultNetworkCredentials;
```

Performing a Basic Search and Processing the Results

In most cases, the only method you'll ever need to call to perform searches is the `PerformQuery` method. This method receives a single parameter of type `QueryParameters` that holds all the information about the query to perform.

```
// Build the query parameters
        QueryParameters prms = new QueryParameters();
        prms.BasicQuery = TXTQuery.Text;
        prms.NumberOfResults = 10;
        prms.ExcerptLength = 200;
```

Once the `QueryParameters` object is filled with the appropriate values, you can call the `PerformQuery` method to execute the search query and retrieve the results.

```
QueryResults results = service.PerformQuery(prms);
```

The `PerformQuery` method returns an object of type `QueryResults` that holds the information about the query that was executed, as well as a `QueryResult` array for the results that were returned (documents that matched the query).

The corresponding part of the sample code loops over the query results and programmatically inserts HTML markup for each into the ASP.NET page.

```
PNLResults.Controls.Add(new  LiteralControl(String.Format("Query returned {0}
results.", results.TotalCount)));
PNLResults.Controls.Add(new  LiteralControl("<br/>"));
PNLResults.Controls.Add(new  LiteralControl("<br/>"));
foreach  (QueryResult result  in  results.Results) {
   PNLResults.Controls.Add(new  LiteralControl(result.Title));
   PNLResults.Controls.Add(new  LiteralControl("<br/>"));
   PNLResults.Controls.Add(new  LiteralControl(result.Uri));
   PNLResults.Controls.Add(new  LiteralControl("<br/>"));
   PNLResults.Controls.Add(new  LiteralControl(result.Excerpt));
   PNLResults.Controls.Add(new  LiteralControl("<br/>"));
   PNLResults.Controls.Add(new  LiteralControl("<br/>"));
}
```

Computing Facets

A facet (like the ones displayed in a search interface) is basically a list of the values found in a given field for the documents that matched a search query. Facet values can be retrieved as well from the search web service. To do this, you must pass the desired field list through the `QueryParameters` object.

The sample asks to retrieve the values for the `@sysauthor` field should be retrieved.

```
RefineByField refineBy = new  RefineByField();
refineBy.Field =  "@sysauthor";
refineBy.Maximum = 10;
refineBy.SortBy = FacetSortByEnum.Occurences;
prms.RefineByFields =  new  RefineByField[] {  refineBy };
```

Once the query is executed, the facet values are available through the `RefineByResults` property of the `QueryResults` object. This property is an array of `RefineByResult` objects, one for each facet that was requested, and in the same order. Each `RefineByResult` object contains a list of `RefineByValue` objects that hold the values for a given field, and the occurrence count for each.

The corresponding part of the sample code processes the returned values for the `@sysauthor` facet and programmatically inserts HTML markup for each into the ASP.NET page.

```
PNLFacets.Controls.Add(new  LiteralControl("<b>Author:</b>"));
PNLFacets.Controls.Add(new  LiteralControl("<br/>"));
foreach  (RefineByValue value  in  results.RefineByResults[0].Values) {
   PNLFacets.Controls.Add(new  LiteralControl(String.Format("{0} ({1})", value.Value,
value.Count)));
   PNLFacets.Controls.Add(new  LiteralControl("<br/>"));
}
```

Retrieving More Information

With the exception of basic information such as the title and the URL, most properties of the `QueryResult` object will not be populated by default, to reduce the amount of data transferred over the network. You have to explicitly request those you need through the `QueryParameters` object. For example, the `Excerpt` field will be set to null if the `ExcerptLength` property is not set to a non-zero value before executing the query.

In the same manner, if you want to retrieve the values of one or several fields for each result, you must request them using the `NeededFields` property. The field values will then be made available through the `Fields` array on each `QueryResult` object.

API Breaking Changes in Coveo Platform Version 7 from 6.5

The topics in this section list the name spaces, methods, properties, parameters, constructors, or events that were renamed, moved, changed, or

removed in the Coveo Platform 7 compared to the Coveo Platform 6.5 (CES 6.5.4889).

Breaking Changes in the Coveo.CNL Assembly

NameSpace Coveo.CNL.SecurityModel

Has been entirely revised.

Coveo.CNL.IO.FileUtilities

- Method `SafeOpen` now requires a new `FileAccess` parameter.

Coveo.CNL.Text.ConvertToString

- Method `AmoutToString` has been renamed to `LongToString`.

Coveo.CNL.Text.StringSerializer

- Method `AppendList` parameter type has changed from `IList` to `ICollection<T>`.
- Method `AppendIntList` parameter type has changed from `IList` to `ICollection<T>`.
- Method `AppendStringList` parameter type has changed from `IList` to `ICollection<T>`.

Coveo.CNL.Text.StringUtilities

- Method `AddWbrTags` has been replaced by `Coveo.CNL.Text.WbrTagsInserterAlgorithm`.

Coveo.CNL.Localization.LocalizedString

- Method `GetSchema` has been removed.
- Method `ReadXML` has been removed. Use the `Constructor` and the `ToStringMethod` instead.
- Method `WriteXML` has been removed. Use the `Constructor` and the `ToStringMethod` instead.

Coveo.CNL.Collections.ObjectCache<V, K>

- The `Constructor` now requires a third parameter: `P_MaxSize`, the maximum size of the cache.

Breaking Changes in the Coveo.CNL.Web Assembly

Coveo.CNL.Web.Ajax.AjaxManagers

- Method `IsRegionUpdated` has been removed.

Coveo.CNL.Web.WebUtilities

- Method `InsertWBRs` has been replaced by `Coveo.CNL.Text.WbrTagsInserterAlgorithm`.

Coveo.CNL.Web.Widgets.Widget

- Property `Title` is now a `LocalizedString`.
  Impacts
  - `Coveo.CES.Web.Search.Controls.SearchInterfaceWidget`
  - `Coveo.CES.Web.Search.Controls.WidgetBoundToHub`
  - `Coveo.CNL.Web.Widgets.SubWidgetZone`

Coveo.CNL.Web.Misc.HeaderAndBody

- Property Type `ShouldFixMinimumHeight` has been removed. Use `Continuous` mode instead.
  Impacts
  - `Coveo.CNL.Web.Ajax.ModalBoxPanel`
  - `Coveo.CNL.Web.Widgets.ChooseWidgetModalBox`
  - `Coveo.CNL.Web.Widgets.EditWidgetModalBox`
  - `Coveo.CES.Web.Search.Controls.ModalBoxBoundToHub`

Coveo.CNL.Web.BetterControls.FillAutoCompleteEventArgs

- `Constructor` now requires two parameters. The last key typed was added.
- Property `Completions` is now a `List<QueryCompletion>` instead of `List<string>`.

Coveo.CNL.Web.BetterControls.BetterTextBox

- Property `AutoCompleteHeight` has been removed, use the CSS class properties instead.
  Impacts
  - `Coveo.CNL.Web.Ajax.ModalBoxPanel`
  - `Coveo.CNL.Web.Widgets.ChooseWidgetModalBox`
  - `Coveo.CNL.Web.Widgets.EditWidgetModalBox`
  - `Coveo.CES.Web.Search.Controls.ModalBoxBoundToHub`

Coveo.CNL.Web.Ajax.ModalBoxPanel

- Property `HeaderStyle` has been replaced by CSS classes `CnlModalBoxHeader` and `CnlModalBox`.
- Property `BodyStyle` has been replaced by CSS classes `CnlModalBoxBody` and `CnlModalBox`.
  Impacts
  - `Coveo.CNL.Web.Widgets.ChooseWidgetModalBox`
  - `Coveo.CNL.Web.Widgets.EditWidgetModalBox`
  - `Coveo.CES.Web.Search.Controls.ModalBoxBoundToHub`

Coveo.CNL.Web.BetterControls.TabControl

- Event `AddTab` is now a `AddingTabEventHandler`.
- Event `TabAdded` is now a `TabAddedEventHandler`.
- Event `TabRenamed` is now a `TabRenamedEventHandler`.
- Event `TabClosed` is now a `TabClosedEventHandler`.
  Impacts
  - `Coveo.CNL.Web.WidgetTabs`

Coveo.CNL.Web.BetterControls.BetterStyle

- Property `BackGradientMode` is now a `LinearGradientMode?` (nullable).
- Property `BackGradientLenght` has been removed, gradient are now rendered in CSS instead of dynamic images.

Coveo.CNL.Web.Widgets.WidgetTab"

- Property `FirstZone` has changed from `WidgetZone` to `ITemplate`.
- Property `SecondZone` has changed from `WidgetZone` to `ITemplate`.
- Property `ThirdZone` has changed from `WidgetZone` to `ITemplate`.

Breaking Changes in the Coveo.CES.Web.Search Assembly

Coveo.CES.Web.Search.Controls.BaseFacet

Facets have been completely rewritten.

Coveo.CES.Web.Search.Controls.Facet

Facets have been completely rewritten.
Impact

- `Coveo.CES.Web.Search.Controls.BaseRefineBy`
- `Coveo.CES.Web.Search.Controls.CustomerService.RefineByAccountCountry`
- `Coveo.CES.Web.Search.Controls.CustomerService.RefineByAccountIdTag`
- `Coveo.CES.Web.Search.Controls.CustomerService.RefineByAccountIndustry`
- `Coveo.CES.Web.Search.Controls.CustomerService.RefineByAccountMailDomainTag`
- `Coveo.CES.Web.Search.Controls.CustomerService.RefineByAccountOwner`
- `Coveo.CES.Web.Search.Controls.CustomerService.RefineByAccountPendingOpportunity`
- `Coveo.CES.Web.Search.Controls.CustomerService.RefineByAccountQuarterOpportunity`
- `Coveo.CES.Web.Search.Controls.CustomerService.RefineByAccountState`
- `Coveo.CES.Web.Search.Controls.CustomerService.RefineByAccountWonOpportunity`
- `Coveo.CES.Web.Search.Controls.CustomerService.RefineByCaseAccount`
- `Coveo.CES.Web.Search.Controls.CustomerService.RefineByCaseIsEscalated`
- `Coveo.CES.Web.Search.Controls.CustomerService.RefineByCaseOwner`

- `Coveo.CES.Web.Search.Controls.CustomerService.RefineByCasePriority`
- `Coveo.CES.Web.Search.Controls.CustomerService.RefineByCaseProduct`
- `Coveo.CES.Web.Search.Controls.CustomerService.RefineByCaseProductBuild`
- `Coveo.CES.Web.Search.Controls.CustomerService.RefineByCaseProductVersion`
- `Coveo.CES.Web.Search.Controls.CustomerService.RefineByCaseStatus`
- `Coveo.CES.Web.Search.Controls.CustomerService.RefineByCommunicationFrom`
- `Coveo.CES.Web.Search.Controls.CustomerService.RefineByCommunicationTo`
- `Coveo.CES.Web.Search.Controls.CustomerService.RefineByOpportunityLeadSource`
- `Coveo.CES.Web.Search.Controls.CustomerService.RefineByOpportunityOwner`
- `Coveo.CES.Web.Search.Controls.CustomerService.RefineByOpportunityStage`
- `Coveo.CES.Web.Search.Controls.CustomerService.RefineByOpportunityType`
- `Coveo.CES.Web.Search.Controls.RefineByAuthor`
- `Coveo.CES.Web.Search.Controls.RefineByClearspaceCommunity`
- `Coveo.CES.Web.Search.Controls.RefineByClearspacePlace`
- `Coveo.CES.Web.Search.Controls.RefineByClearspacePlaceType`
- `Coveo.CES.Web.Search.Controls.RefineByClearspaceTag`
- `Coveo.CES.Web.Search.Controls.RefineByClearspaceTagGroup`
- `Coveo.CES.Web.Search.Controls.RefineByClearspaceTaskAssignedTo`
- `Coveo.CES.Web.Search.Controls.RefineByClearspaceType`
- `Coveo.CES.Web.Search.Controls.RefineByCollection`
- `Coveo.CES.Web.Search.Controls.RefineByConfluenceLabel`
- `Coveo.CES.Web.Search.Controls.RefineByConfluenceSpace`
- `Coveo.CES.Web.Search.Controls.RefineByConfluenceType`
- `Coveo.CES.Web.Search.Controls.RefineByDynamicsAccount`
- `Coveo.CES.Web.Search.Controls.RefineByDynamicsOwner`
- `Coveo.CES.Web.Search.Controls.RefineByDynamicsProduct`
- `Coveo.CES.Web.Search.Controls.RefineByDynamicsType`
- `Coveo.CES.Web.Search.Controls.RefineByExchangeFolder`
- `Coveo.CES.Web.Search.Controls.RefineByExchangeType`
- `Coveo.CES.Web.Search.Controls.RefineByField`
- `Coveo.CES.Web.Search.Controls.RefineByLanguage`
- `Coveo.CES.Web.Search.Controls.RefineByMailbox`
- `Coveo.CES.Web.Search.Controls.RefineByMonth`
- `Coveo.CES.Web.Search.Controls.RefineByNotesType`
- `Coveo.CES.Web.Search.Controls.RefineByPeopleCompany`
- `Coveo.CES.Web.Search.Controls.RefineByPeopleManager`
- `Coveo.CES.Web.Search.Controls.RefineByPeopleWorkTitle`
- `Coveo.CES.Web.Search.Controls.RefineByRecipient`
- `Coveo.CES.Web.Search.Controls.RefineBySalesforceAccount`
- `Coveo.CES.Web.Search.Controls.RefineBySalesforceAmount`
- `Coveo.CES.Web.Search.Controls.RefineBySalesforceCloseQuarter`
- `Coveo.CES.Web.Search.Controls.RefineBySalesforceCountry`
- `Coveo.CES.Web.Search.Controls.RefineBySalesforceIndustry`
- `Coveo.CES.Web.Search.Controls.RefineBySalesforceOpportunityType`
- `Coveo.CES.Web.Search.Controls.RefineBySalesforceOwner`
- `Coveo.CES.Web.Search.Controls.RefineBySalesforceProduct`
- `Coveo.CES.Web.Search.Controls.RefineBySalesforceStage`
- `Coveo.CES.Web.Search.Controls.RefineBySalesforceType`
- `Coveo.CES.Web.Search.Controls.RefineBySender`
- `Coveo.CES.Web.Search.Controls.RefineBySharePointContentType`
- `Coveo.CES.Web.Search.Controls.RefineBySharePointDocumentSet`
- `Coveo.CES.Web.Search.Controls.RefineBySharePointParent`
- `Coveo.CES.Web.Search.Controls.RefineBySharePointSite`
- `Coveo.CES.Web.Search.Controls.RefineBySharePointType`
- `Coveo.CES.Web.Search.Controls.RefineByType`
- `Coveo.CES.Web.Search.Controls.RefineByYear`

Coveo.CES.Web.Search.Controls.FacetChart

Class has been completely rewritten.

Coveo.CES.Web.Search.Settings.RefineByClusterSetting

- Property `Clusters` type has changed to a `List<string>`.
- Property `RemovedClusters` type has changed to a `List<string>`.

Coveo.CES.Web.Search.Controls.BootstrapControl

Class has been removed, use the `Coveo.CES.Web.Search.Controls.SearchHub` instead.

Coveo.CES.Web.Search.Controls.SearchControl

Method `ToggleInlineAdvancedSearch` has been removed.
Impact

- `Coveo.CES.Web.Search.Controls.AnalyticsSearch`
- `Coveo.CES.Web.Search.Controls.BaseEmailSearch`
- `Coveo.CES.Web.Search.Controls.EmailSearch`
- `Coveo.CES.Web.Search.Controls.SharedEmailsSearch`
- `Coveo.CES.Web.Search.Controls.BaseRelatedResults`
- `Coveo.CES.Web.Search.Controls.CustomerService.BugsRelatedToCase`
- `Coveo.CES.Web.Search.Controls.CustomerService.CaseRelatedOpportunities`
- `Coveo.CES.Web.Search.Controls.CustomerService.CasesRelatedToAccount`
- `Coveo.CES.Web.Search.Controls.CustomerService.CasesRelatedToCase`
- `Coveo.CES.Web.Search.Controls.CustomerService.CommunicationsRelatedToAccount`
- `Coveo.CES.Web.Search.Controls.CustomerService.CommunicationsRelatedToCase`
- `Coveo.CES.Web.Search.Controls.CustomerService.FilesRelatedToCase`
- `Coveo.CES.Web.Search.Controls.CustomerService.OpportunitiesRelatedToAccount`
- `Coveo.CES.Web.Search.Controls.CustomerService.PeopleRelatedToAccount`
- `Coveo.CES.Web.Search.Controls.ClearspaceSearch`
- `Coveo.CES.Web.Search.Controls.CustomerService.AccountConsoleSearch`
- `Coveo.CES.Web.Search.Controls.CustomerService.AccountOverviewSearch`
- `Coveo.CES.Web.Search.Controls.CustomerService.CaseConsoleSearch`
- `Coveo.CES.Web.Search.Controls.FileSearch`
- `Coveo.CES.Web.Search.Controls.FileShareSearch`
- `Coveo.CES.Web.Search.Controls.ImageSearch`
- `Coveo.CES.Web.Search.Controls.IntranetSearch`
- `Coveo.CES.Web.Search.Controls.OutlookSearch`
- `Coveo.CES.Web.Search.Controls.OutlookBaseEmailQuickSearch`
- `Coveo.CES.Web.Search.Controls.OutlookEmailQuickSearch`
- `Coveo.CES.Web.Search.Controls.OutlookSharedEmailsQuickSearch`
- `Coveo.CES.Web.Search.Controls.OutlookFileSharesQuickSearch`
- `Coveo.CES.Web.Search.Controls.OutlookPeopleQuickSearch`
- `Coveo.CES.Web.Search.Controls.OutlookPerson360ViewSearch`
- `Coveo.CES.Web.Search.Controls.OutlookPersonalFilesQuickSearch`
- `Coveo.CES.Web.Search.Controls.OutlookSalesforceQuickSearch`
- `Coveo.CES.Web.Search.Controls.OutlookSharePointQuickSearch`
- `Coveo.CES.Web.Search.Controls.PeopleSearch`
- `Coveo.CES.Web.Search.Controls.PersonalFilesSearch`
- `Coveo.CES.Web.Search.Controls.QuickSearch`
- `Coveo.CES.Web.Search.Controls.SalesforceSearch`
- `Coveo.CES.Web.Search.Controls.SitesAndListsSearch`

Coveo.CES.Web.Search.Providers.ICESUserIdentityFactory

- Method `CreateExchangeUser` has been replaced by `CreateSecurityProviderUser`.
- Method `CreateNovellUser` has been replaced by `CreateSecurityProviderUser`.
- Method `CreateNotesUser` has been replaced by `CreateSecurityProviderUser`.
- Method `CreateSecurityProviderUser` has been replaced by `CreateSecurityProviderUser`.
  Impact
  - `Coveo.CES.Web.Search.Providers.Cmf.CmfUserIdentityFactory`
  - `Coveo.CES.Web.Search.Providers.DummyUserIdentityFactory`

Coveo.CES.Web.Search.Providers.ICESSearchProvider

- Method `IsSplitGroupByField` has been removed. This information is now available on the `SharedInformation(SearchObject.Provider.SharedInformation.IsSplitGroupByField)`.
- Method `CloneWithRemoteIndexes` has been replaced by an overload of the `Clone` method.
- Method `AuthenticateExternalUser` has been removed and replaced by method `AuthenticateUser` that perform a login against a security provider.
- Method `TagDocument(string, string, int, int)` now requires a string as for the persistent document id (third parameter).
- Method `CreateWindowsUser` has been removed. Use `ICESUserIdentityFactory.CreateWindowsUser()` instead.
- Method `CreateWindowsUserFromSid` has been removed. Use `ICESUserIdentityFactory. CreateWindowsUserFromSid()` instead.
- Method `CreateExchangeUser` has been removed. Use `ICESUserIdentityFactory.CreateExchangeUser()` instead.
- Method `CreateNovellUser` has been removed. Use `ICESUserIdentityFactory.CreateNovellUser` instead.
- Method `CreateNotesUser` has been removed. Use `ICESUserIdentityFactory.CreateNotesUser` instead.
- Method `CreateSecurityProviderUser` has been removed. Use `ICESUserIdentityFactory.CreateSecurityProviderUser` instead.

- Method `RefreshConnection` has been removed.
- Property `ExternalSecurityAuthentication` has been removed.
- Property `AvailableQueryFields` has moved to `SharedInformation.IncludeInQueryFields`.
- Property `AvailableFields` has moved to `SharedInformation.AllFields`.
- Property `AmIAdmin` has been removed, use `IsAdministrator` instead.

  Impact

  - `Coveo.CES.Web.Search.Providers.Cmf.CmfSearchProvider`
  - `Coveo.CES.Web.Search.Providers.Xml.XmlSearchProvider`
  - `Coveo.CES.Web.Search.Providers.Cmf465.CmfSearchProvider`
  - `Coveo.CES.Web.Search.Providers.BaseSearchProvider`
  - `Coveo.CES.Web.Search.Providers.TweakedSearchProvider`

Coveo.CES.Web.Search.Providers. ICESQuery

- Method `AddAggregatedMirror` has been removed, use `RemoteIndexes` instead.
- Property `Collections` has been removed, use `CollectionIds` instead.
- Property `TimeZoneOffset` has been removed, use `TimeZone` instead.
- Property `MaximumNumberOfGroupByResults` has been removed.
- Property `EnableOptimizations` has been removed.
- Property `RemoteIndexes` is now an `IList<RemoteIndex>`.

  Impact

  - `Coveo.CES.Web.Search.Providers.Cmf.CmfQuery`
  - `Coveo.CES.Web.Search.Providers.TweakedQuery`
  - `Coveo.CES.Web.Search.Providers.Demo.DemoQuery`
  - `Coveo.CES.Web.Search.Providers.TweakedQuery`
  - `Coveo.CES.Web.Search.Providers.Xml.XmlQuery`

Coveo.CES.Web.Search.Providers.ICESResult

- Method `GetSummary(int)` has been removed, the summary size is now specified on the query.
- Method `get_Summary` has been removed, use `GetSummary` instead.
- Method `get_Fields` has been removed, use `GetField` instead.
- Property `UniqueId` has been removed use `UniqueID` instead.
- Property `CollectionId` has been removed use `CollectionID` instead.
- Property `SourceId` has been removed use `SourceID` instead.
- Property `TargetUri` has been removed use `ClickUri` instead.
- Property `PctScore` has been removed use `PercentageScore` instead.
- Property `FileSize` has been removed use `Size` instead.
- Property `Date` has been removed use `ModifiedDate` instead.
- Property `HasHTMLVersion` has been removed use `HasHtmlVersion` instead.
- Property `IsRecord` has been removed use `IsXmlrecord` instead.
- Property `IsFixed` has been removed use `IsTopResult` instead.
- Property `IsIndexBrowserResult` has been removed.

  Impact

  - `Coveo.CES.Web.Search.Providers.Cmf.CmfResult`
  - `Coveo.CES.Web.Search.Providers.FakeResult`
  - `Coveo.CES.Web.Search.Providers.MetaResult`
  - `Coveo.CES.Web.Search.Controls.MergedConversation`
  - `Coveo.CES.Web.Search.Controls.MergedDocumentSet`
  - `Coveo.CES.Web.Search.Controls.MergedPerson`
  - `Coveo.CES.Web.Search.Providers.TweakedResult`
  - `Coveo.CES.Web.Search.Providers.Demo.DemoResult`
  - `Coveo.CES.Web.Search.Providers.Xml.XmlResult`

Coveo.CES.Web.Search.Providers.HighlightedString

- Property `Concept` has been removed, use `Value` instead.
- Property `Sentence` has been removed, use `Value` instead.

Coveo.CES.Web.Search.Providers.BaseSearchProvider

- Method `SysDocIdQueryExpression` has been replaced by `GetDocIDQueryExpression`.

Coveo.CES.Web.Search.Providers.Cmf.CmfSearchProvider

- `Constructors` have been restructured, use the `SeardchProviderFoctory` Instead.
- `Constructors` have been restructured, use the `SeardchProviderFoctory` Instead.

Coveo.CES.Web.Search.Providers.SearchProviderFactory

- Method `CreateDefaultSearchProvider` now requires only three parameters `Impersonator` chain has been removed.

Coveo.CES.Web.Search.Providers.SharedInformation

- Method `GetSearchInterfaceData` is now obsolete.
- Method `GetSearchHubData` is now obsolete.
- Property `AvailableSortByFields` has been moved to `SearchProviderData`.
- Property `AvailableGroupByFields` has been moved to `SearchProviderData`.
- Property `AvailableSplitGroupByFields` has been moved to `SearchProviderData`.

Coveo.CES.Web.Search.Providers.Cmf.CmfQuery

- Method `AddAggregatedMirror` has been removed. Remote indexes should be specified on the `SearchProvider`.

Coveo.CES.Web.Search.Controls.BaseRelatedResults

Class has been replaced by `Coveo.CES.Web.Search.Controls.SearchInterfaceWidget`.
Impact

- `Coveo.CES.Web.Search.Controls.CustomerService.BugsRelatedToCase`
- `Coveo.CES.Web.Search.Controls.CustomerService.CaseRelatedOpportunities`
- `Coveo.CES.Web.Search.Controls.CustomerService.CasesRelatedToAccount`
- `Coveo.CES.Web.Search.Controls.CustomerService.CasesRelatedToCase`
- `Coveo.CES.Web.Search.Controls.CustomerService.CommunicationsRelatedToAccount`
- `Coveo.CES.Web.Search.Controls.CustomerService.CommunicationsRelatedToCase`
- `Coveo.CES.Web.Search.Controls.CustomerService.FilesRelatedToCase`
- `Coveo.CES.Web.Search.Controls.CustomerService.OpportunitiesRelatedToAccount`
- `Coveo.CES.Web.Search.Controls.CustomerService.PeopleRelatedToAccount`

Coveo.CES.Web.Search.Controls.ResultList

- Method `FetchResultsFromServer` has changed and now takes as second parameter a `MatchresultDelegate`.

Coveo.CES.Web.Search.Controls.SearchBox

- Property `EnablePrefetch` has been moved to `Coveo.CES.Web.Search.Controls.Query`.

Coveo.CES.Web.Search.Controls.SearchBuilder

The only constructor available is the one without any parameters. This class is serialized via the `ToString` method.

- Property `Collections` has been removed, uses `CollectionIds` instead.
- Property `CollectionIds` has changed from an `IList` to a `List<int>`.
- Property `CollectionNames` has changed from an `IList` to a `List<string>`.
- Property `EnabledCollection` has changed from an `IList` to a `List<int>`.
- Property `RankingExpressions` has changed from an `IList` to a `List<RankingExpression>`.
- Property `RankingOverrides` has changed from an `IList` to a `List<RankingOverride>`.
- Property `Mirrors` has changed from an `IList` to a `List<RemoteIndex>`.
- Property `TimeZoneOffset` has changed from an `int` to a `TimeZoneInfo`.
- Property `Optimize` has been removed, all queries are optimized.
- Property `GroupFields` has changed from an `IList` to a `List<GroupBy>`.
- Method `Serialize` has been removed, use `ToStringInstead`.

Coveo.CES.Web.Search.ModeEnum

- Enum value `AdvancedSearch` has been removed, advanced is now a panel instead of a mode.
- Enum value `Preferences` has been removed, preferences is now a panel instead of a mode.
- Enum value `Filters` has been removed, filter is now a panel instead of a mode.

Coveo.CES.Web.Search.SecurityUtilities

Most changes in this class are related to the fact that security providers have changed significantly from the Coveo Platform versions 6.5 to 7.0.

- Method `BeginLoginToProvider` return type has changed from `Boolean` to `Void`.
- Method `IsWaitingForSamlComeBack` has been removed.
- Method `PerformSamlLogin` has been removed.
- Method `BumpUpSearchIdIfNecessary` has been removed.
- Method `Search_BeforeQueries` has been removed.
- Method `GetInstance` has been removed.
- Method `AddExternalNetworkIDs` has been removed.

Coveo.CES.Web.Search.Global.SecurityProviderConfig

Everything related to SAML have been removed.

- Property `SamlRedirUri` has been removed.
- Property `SamlArtifactResolverUri` has been removed.
- Property `SamlLogoutUri` has been removed.
- Property `SamlSignatureCertificate` has been removed.
- Property `SamlArtifactArgName` has been removed.
- Property `SamlPostResponseArgName` has been removed.
- Property `SamlBackTargetArgName` has been removed.
- Method `LookForUpdatedConfig` has been removed.
- `Constructor` with SAML parameters have been removed.

Coveo.CES.Web.Search.Controls.SearchHub

- Method `ChangeCurrentInterface` has been removed.

Coveo.CES.Web.Search.Controls.SearchState

- Property `TimeZoneOffset` has been replaced by `TimeZone`, which is now an `enum` of available time zones.

Coveo.CES.Web.Search.SearchConfig

- Property `Server` has been replaced by `Hostname`.
- Property `AlternateServers` has been replaced by a new way to configure back-end server failover.
- Property `PhysicalIndexName` has been removed.

Coveo.CES.Web.Search.SearchUriOptions

- Enum value `UseUriShortenerThreshold` has been removed.
- Enum value `UseUriShortener` has been removed.

Coveo.CES.Web.Search.Controls.SearchInterface

- Property `Files` has been removed, use `Type` instead.
- Property `Email` has been removed, use `Type` instead.
- Property `People` has been removed, use `Type` instead.
- Property `Salesforce` has been removed, use `Type` instead.
- Property `SharePoint2007` has been removed, use `Type` instead.
- Property `SharePointSitesAndLists` has been removed, use `Type` instead.
- Property `NoMiniQueriesWhenActive` has been removed, use `Type` instead.
- Property `NoMiniQueriesFromHere` has been removed, use `Type` instead.

Coveo.CES.Web.Search.Interfaces.HubSettings

- Method `ContainsInterface` now requires a third boolean parameter to retrieve settings for mobile interfaces.

Coveo.CES.Web.Search.Interfaces.InterfaceSettings

- `ContainsInterface(System.String, Coveo.CES.Interops.CESSearch.BaseSearchInterfaceEnumeration)` has been removed.
- Method `ListDependentSettings` now requires a second parameter.
- Property `DefaultConcreteFacetPosition` has been removed.
- Property `ModalBoxStyle` has been removed.
- Property `ModalBoxHeaderStyle` has been removed.

- Property `ModalBoxHeaderTitleStyle` has been removed.
- Property `ModalBoxHeaderLinkStyle` has been removed.
- Property `ModalBoxHeaderLinkHoveredStyle` has been removed.
- Property `ModalBoxBodyStyle` has been removed.
- Property `InterfaceLogoUri` has been removed use `LargeIcon` instead.

Coveo.CES.Web.Search.SearchUtilities

- Method `CreateConnectedSearch4` has been removed, use the `Coveo.CES.Web.Search.Providers.COM.ConnectionFactory` instead.
- Method `CreateDefaultSearchProvider` has been removed, use `Coveo.CES.Web.Search.Providers.SearchProviderFactory` instead
- Method `CreateDefaultSearchProviderForBackgroundRequests` has been removed, use `Coveo.CES.Web.Search.Providers.SearchProviderFactory` instead
- Method `CreateDefaultUserIdentityFactory` has been removed, use `Coveo.CES.Web.Search.Providers.SearchProviderFactory` instead
- Method `GetSecurityProvidersConfig` has been removed, use `Coveo.CES.Web.Search.Providers.SearchProviderFactory` instead
- Method `CreateConnectedAdmin` has been removed. Search cannot connect to the admin anymore.
- Method `CloseCachedConnections` has been removed. Search cannot connect to the admin anymore.
- Method `WriteHighlightedString` has been removed.
- Method `PrefetchTerms` has been moved to `ISearchProvider`.

Coveo.CES.Web.Search.Controls.ParentChildMapping

- Method `RegisterPotentialParentOrChild` has been replaced by `RegisterPotentialParent` and `RegisterPotentialChild`.
- Method `RegisterPossibleParent` has been removed.

Coveo.CES.Web.Search.Controls.QueryWrapper

- Method `GetResultById(Coveo.CES.Web.Search.Providers.ICESSearchProvider, System.String, System.String, System.String, System.String, Int32)` has been removed.

Coveo.CES.Web.Search.Interfaces.HubSettings

- Method `LookForUpdatedSettings` has been removed.

Coveo.CES.Web.Search.Global.GlobalSettings

- Method `LookForUpdatedSettings` has been removed.

Coveo.CES.Web.Search.Controls.BoundToHub

- Method `IsHubObjectAvailable` has been removed, now there is always a hub.

Coveo.CES.Web.Search.Controls.BoundToGroup

- Class has been removed.

Coveo.CES.Web.Search.Controls.HubBinding

- Method `IsHubObjectAvailable` has been removed, now there is always a hub.
- Method `AddDeferredBinding` now uses a `TypedControlBinder.DeferredBinding<SearchHub>`.

Coveo.CES.Web.Search.Controls.SearachBinding

- Method `AddDeferredBinding` now uses a `TypedControlBinder.DeferredBinding<SearchControl>`.

Coveo.CES.Web.Search.Controls.QueryBinding

- Method `AddDeferredBinding` now uses a `TypedControlBinder.DeferredBinding<QueryControl>`.

Coveo.CES.Web.Search.Controls.ResultBinding

- Property `ResultObject` has been replaced by `ResultInfo`.
- Method `AddDeferredBinding` now uses a `TypedControlBinder.DeferredBinding<QueryControl>`.

Coveo.CES.Web.Search.Controls.BindingException

- Class has been moved to the assembly `CoveoCNL.Web`.

Coveo.CES.Web.Search.Controls.ResultInfo

- Property `Result` has been replaced by `ResultObject`.

Coveo.CES.Web.Search.Controls.CustomRefineByFields

- Property `ShowSelectedFacetOnTop` has been removed.

Coveo.CES.Web.Search.Controls.ResultUri

- Property `TrimSelectedFolder` has been removed.

Coveo.CES.Web.Search.Controls.IndexBrowserSearch

This class have been removed and implemented in the Administration Tool instead.

Coveo.CES.Web.Search.Controls.IfIndexBrowserResult

Class has been removed.

Coveo.CES.Web.Search.Controls.SelectedResultsManager

- Method `EnterEditionMode` does not take any parameter anymore.
- Method `LeaveEditionMode` does not take any parameter anymore.
- Method has been removed.
- Property `Current` has been removed; this class can now be instantiated more than once.

Coveo.CES.Web.Search.Controls.IPhoneAppManager

Class has been removed.

Coveo.CES.Web.Search.Controls.IfIPhoneApp

Class has been removed.

Coveo.CES.Web.Search.Interfaces.InterfaceUtilities

- Method `GetEditInterfaceUri` now requires the name of the hub as second parameter.
- Method `LookForUpdatedSettings` has been removed.

Coveo.CES.Web.Search.Analytics.ResultData

- Property `PhysicalIndexName` has been removed.

Coveo.CES.Web.Search.Settings.CollectionsSetting

- Property `Collections` type has changed from an `IList` to a `List<int>`.

Coveo.CES.Web.Search.SearchStrings

Many strings have been removed or renamed due to the redesign of the `searchInterface`.

Coveo.CES.Web.Search.Global.AudienceInfo

- Property `Groups` has change from a `List<string>` to a `List<SecurityEntryInfo>`.
- Property `Users` has change from a `List<string>` to a `List<SecurityEntryInfo>`.

Coveo.CES.Web.Search.Controls.IfInterfaceDefaultFacetPosition

Class has been removed. Use `IfFacetPosition` instead.

Coveo.CES.Web.Search.Controls.RankingExpressionInfo

Class has been removed, use `Coveo.CES.Web.Search.Controls.RankingExpression` instead.

Coveo.CES.Web.Search.Controls.AdvancedSearchSection

Class has been removed, use `Coveo.CES.Web.Search.Controls.AdvancedSearchPanel` instead.

Coveo.CES.Web.Search.Controls.AdvancedDate

Class has been removed.

Coveo.CES.Web.Search.Controls.GroupFieldInfo”

Class has been removed, use `Coveo.CES.Web.Search.Controls.GroupBy` instead.

Coveo.CES.Web.Search.Controls.MCMSSecurity

Class has been removed.

Coveo.CES.Web.Search.Controls.InterfaceLogo

Class has been removed, use `Coveo.CES.Web.Search.Controls.InterfaceIcon` instead.

Coveo.CES.Web.Search.Controls.TimeDistributionWidget

Class has been removed.

Coveo.CES.Web.Search.Controls.QueryOptimized

Class has been removed, all queries are optimized.

Coveo.CES.Web.Search.Controls.QueryEditingParam

Class has been removed, use `TextBoxParam` instead.

Coveo.CES.Web.Search.Controls.AdvancedProperties

Class has been removed.

Coveo.CES.Web.Search.Controls.FacetChartWidget

Class has been renamed `FacetChart`.

Coveo.CES.Web.Search.Controls.ChildSearchInterface

Class has been removed, use `SearchIntrefaceWidget` instead.

Coveo.CES.Web.Search.Controls.InlineAdvancedSearch

Class has been removed, use `Coveo.CES.Web.Search.Controls.AdvancedSearchPanel` instead.

Coveo.CES.Web.Search.Controls.CustomerService.OpportunityRevenueSummaryWidget

Class has been renamed to `OpportunityRevenueSummary`.

Coveo.CES.Web.Search.Controls.CustomerService. OpportunityDurationWidget

Class has been renamed to `OpportunityDuration`.

Coveo.CES.Web.Search.Controls.CustomerService. OpportunityStageSummaryWidget

Class has been renamed to `OpportunityStageSummary`.

Coveo.CES.Web.Search.Controls.CustomerService. OpportunityWinLossSummaryWidget

Class has been renamed to `OpportunityWinLossSummary`.

Coveo.CES.Web.Search.Controls.CustomerService. CaseRelatedOpportunities

Class has been removed.

Coveo.CES.Web.Search.Controls.CustomerService. CasesActivity

Class has been removed.

Coveo.CES.Web.Search.Controls.CustomerService.CSEnvironment

- Method `GetQueryToMatchCasesRelatedToCaseCustomer` has been removed.

Coveo.CES.Web.Search.SearchUserProfile

- `Constructors` now requires a `SearhControl`.

Coveo.CES.Web.Search.SearchInterfaceType

- Enum value `Files` has been replaced by `PersonalFiles`.

Coveo.CES.Web.Search.PeopleUtilities

- Method `GetPersonInfo` now requires a `ISearchProvider` as second parameter.
- Method `GetPersonPictureUri` now requires a `SearchControl` as second parameter.

Coveo.CES.Web.Search.Controls.QueryControl

- Property `Optimize` has been removed, all queries are optimized now.

Coveo.CES.Web.Search.Controls.GenerateCommentsEventArgs

- Property `Comments` has been removed. Use `Controls` instead.

Coveo.CES.Web.Search.SearchConfig

- Property `EnableNewSearchAPI` has been removed, CMF Search Provider is now the default one.
- Property `MediaEndpoint` has been removed.

Coveo.CES.Web.Search.Controls.OverrideUserEventArgs

- Property `Impersonators` has been removed. SSL certificate are now used to secure communications between Back-End and Front-End.

Coveo.CES.Web.Search.QueryTransform

- Method `ConvertQueryToWildcards` now requires three parameters to limit the length of the transformed query.
- Method `ConvertQueryToMatchTitle` now requires four parameters to limit the length of the transformed query.
- Method `ConvertQueryToMatchAtLeastOneOf` now requires three parameters to limit the length of the transformed query.

Coveo.CES.Web.Search.Controls.HistoryState

- Method `Serialize` has been removed.

Coveo.CES.Web.Search.Controls.ResultPrintableUri

- Property `StayExpanded` has been removed.

Coveo.CES.Web.Search.Controls.Logo

- Property `Compact` has been removed.

Coveo.CES.Web.Search.Controls.QueryState

- Property `StopOptimizing` has been removed, all queries are optimized now.

Coveo.CES.Web.Search.Controls.Logos

- Property `Compact` has been removed.

Coveo.CES.Web.Search.Controls.QueryLink

- Property `DropDownStyle` has been removed.

Coveo.CES.Web.Search.Controls.LookupRelationshipsEventArgs

- `Constructor` now uses a `BaseResultList` instead of `ResultList`.
- Property `ResultList` is now a `BaseResultList` instead of `ResultList`.

Coveo.CES.Web.Search.Controls.BaseEmailSearch

- Method `GetEmailAddressToUse` has been removed, use `UserProfile.Preferences.GetEmailAddressToUse` instead.
- Method `SimplifyFromToCC` has been removed, use `UserProfile.Preferences. SimplifyFromToCC` instead.

Coveo.CES.Web.Search.Controls.ResultSummary

- Property `HeaderStyle` has been removed.

Coveo.CES.Web.Search.Controls.LoginPrompt

Class has been removed.

Coveo.CES.Web.Search.Controls.ErrorPanel

Class has been replaced by `ErrorReporting`.

Coveo.CES.Web.Search.ErrorConstants

Class has been removed.

Customizing Desktop Integration Package (DIP) Bitmaps and Colors

The Desktop Integration Package (DIP) allows to integrate Coveo into a Windows workstation by providing a floating Desktop Searchbar, and integration in Microsoft Outlook, and allowing end-users to index files stored on their computer (see Desktop Integration Package).

Starting with the September 2016 release of the Coveo Desktop Integration Package and the Coveo .NET Front-End, it is possible to control some of the DIP visual aspects from the search web server, mostly bitmaps and colors. To enable that feature, some files need to be created on the web server. The purpose of this topic is to explain how to proceed.

Where to Create the Files on the Web Server

The files containing the customized color schemes and bitmap resources need to be created in the `Coveo\Anonymous\SearchBar` subdirector y under the IIS search web site physical directory, which is by default `C:\Program Files\Coveo .NET Front-End 12\Web\`. The custom color scheme JSON file must be named `CustomColorSchemes.json`, whereas the customized bitmap files must be copied in a subdirectory named `CustomResources`. So the whole default paths are:

- `C:\Program Files\Coveo .NET Front-End 12\Web\Coveo\Anonymous\SearchBar\CustomColorSchemes.json`
- `C:\Program Files\Coveo .NET Front-End 12\Web\Coveo\Anonymous\SearchBar\CustomResources\`

Bitmap Files that Can Be Customized

The DIP bitmaps that can be customized are the following:

Name	Searchbar or Outlook Sidebar?
back_disabled_x26.png	Outlook Sidebar
back_disabled_x32.png	Searchbar
back_hovered_x26.png	Outlook Sidebar
back_hovered_x32.png	Searchbar
back_x26.png	Outlook Sidebar
back_x32.png	Searchbar
clearsearch.png	Both
close.png	Both
collapse.png	Outlook Sidebar
default_itf_icon.png	Both
detach.png	Searchbar
disconnected.png	Both
dropdown.png	Both
expand.png	Outlook Sidebar
expandwnd_hovered_x26.png	Outlook Sidebar
expandwnd_x26.png	Outlook Sidebar
forward_disabled_x26.png	Outlook Sidebar
forward_disabled_x32.png	Searchbar
forward_hovered_x26.png	Outlook Sidebar
forward_hovered_x32.png	Searchbar
forward_x26.png	Outlook Sidebar
forward_x32.png	Searchbar
help_x16.png	Outlook Sidebar
home_hovered_x26.png	Outlook Sidebar
home_x26.png	Outlook Sidebar
info_x16.png	Outlook Sidebar
loading.gif	Both
lockedBW.png	Both (Options dialog)
logocoveo.png	Searchbar
logocoveo2.png	Outlook Sidebar
logocoveo2_vertical.png	Outlook Sidebar
pinned_x24.png	Searchbar *(Added in March 2018)*
search_x12.png	Outlook Sidebar
setting_x16.png	Outlook Sidebar
sidebar_collapse_x64.png	Outlook Sidebar
sidebar_hide_x64.png	Outlook Sidebar
unpinned_x24.png	Searchbar *(Added in March 2018)*

For each of them, when a customized file is not provided, the DIP (either the floating Searchbar or the Outlook Sidebar) displays its default bitmap. The following zip file contains the default DIP bitmap files: Default Bitmaps.zip. You may have to experiment a bit to find where each one is displayed exactly.

> **Bitmap Width and Height**
> It's important for any customized bitmap to have the same width and height than the default one. Otherwise, that will likely cause display problems.

Custom Color Schemes

The DIP comes with a few predefined Searchbar and Sidebar color schemes, which can be selected from the DIP options dialog. Starting with the October 2013 release of the DIP, it is possible to create custom color schemes in the Windows registry of a machine on which the DIP is installed, as described in the following help pages:

- Creating a Custom Color Scheme for the Desktop Searchbar
- Creating a Custom Color Scheme for the Outlook Sidebar

Starting with the September 2016 release of the DIP, it is now also possible to create custom color schemes in a JSON file stored on the search web server. Doing so, every DIP configured with that search page will automatically make possible to the users to start using them, without requiring each user to create the registry keys on their machine.

Color Scheme File Example:

```
{
  SearchBarSchemes: [
    {
      ID: "MySearchBarScheme",
      Name: "My Search Bar Scheme",
      SearchbarBackgroundResultColor: "#123456",
      SearchbarBackgroundSearchColor: "#234567",
      SearchbarCloseButtonBorderColor: "#345678",
      SearchbarInterfaceLabelColor: "#456789",
      SearchbarSearchButtonOverColor: "#56789A",
      SearchbarSearchFormBorderColor: "#6789AB",
      SearchbarSearchTextboxBorderColor: "#789ABC",
      SearchbarUndockButtonBorderColor: "#89ABCD",
      SearchButtonBorderColor: "#9ABCDE",
      SearchButtonColor: "#ABCDEF"
    }
  ],
  OutlookSchemes: [
    {
      ID: "MySideBarScheme",
      Name: "My Side Bar Scheme",
      OutlookTheme: "Blue",
      BrowserBackgroundColor: "#123456",
      BrowserConversationSelectedColor: "#234567",
      BrowserStrokeColor: "#345678",
      BrowserTabBackgroundColor: "#456789",
      GripColor: "#56789A",
      GripPointsColor: ""#6789AB,
      MainColor: "#789ABC",
      SearchButtonColor: "#89ABCD",
      SearchButtonColorPressed: "#9ABCDE",
      SidebarBorderColor: "#ABCDEF",
      TabColor: "#012345",
      TabHoveredColor: "#123456",
      TabSelectedColor: "#234567",
      TabSeparatorColor: "#345678"
    },
    {
      ID: "MySideBarScheme2",
      Name: "My Side Bar Scheme 2",
      OutlookTheme: "Blue",
      BrowserBackgroundColor: "#123456",
      BrowserConversationSelectedColor: "#234567",
      BrowserStrokeColor: "#345678",
      BrowserTabBackgroundColor: "#456789",
      GripColor: "#56789A",
      GripPointsColor: "#6789AB",
      MainColor: "#789ABC",
      SearchButtonColor: "#89ABCD",
      SearchButtonColorPressed: "#9ABCDE",
      SidebarBorderColor: "#ABCDEF",
      TabColor: "#012345",
      TabHoveredColor: "#123456",
      TabSelectedColor: "#234567",
      TabSeparatorColor: "#345678"
    }
  ]
}
```