

XSearchResults

Quick Start and API Reference

Introduction

XSearchResults is an ASP.net server control that conducts a search on a dtSearch index and returns formatted search results. Provided with the control is a fully functional search application. For search result customization, three sample XSLT files are included.

This document is technical in nature; therefore it is recommended that the reader have a basic background in ASP.net and IIS configuration. While simple implementations will require little or no programming experience, more extensive customizations may require expertise in ASP.net and XSLT.

Product Licensing

The trial version will only display three (3) results. To purchase the product, visit: www.cybergroup.com. Each license allows the application to be run on a single site (IP or URL). For bulk licensing, contact us at support@cybergroup.com

dtSearch must be installed before using XSearchResults.

Before Installing

Before installing, make sure to have the following:

- 1) Access to a Windows Web Server running ASP.net
- 2) dtSearch 7.x or later installed on the web server.
- 3) A dtSearch index to search. If you do not have an index (or access to an index) and would like to make one for testing, you can generate one using dtSearch's Indexer which comes with the dtSearch product. For database-only indexes, try our dbIndexer product to index ODBC-compliant databases (SQL Server, MS Access, etc)
- 4) For file-based indexes, the indexed files should be on the web server or available through network share.

Installation

Summary of steps:

- 1) Run the MSI Installer
- 2) Configure the web.config Index Path setting
- 3) Define relative path translations (for file-based indexes)
- 4) Customize XSLT (optional)
- 5) Begin searching!

Step 1: Run the MSI Installer

To install XSearchResults, run the MSI installer (provided in the installation zip file) from a Windows web server running ASP.net. The installer will prompt you for an application folder name (leave as default if you're not sure).

Step 2: Configure the Web.Config Index Path

After installing XSearchResults to the web server, browse to the application's location and edit the web.config file. The "IndexPath" setting should be pointing to a valid physical path for your dtSearch index.

Step 3: Relative Path Translations

If the search index contains files from a file system (e.g. built using dtSearch Indexer, as opposed to dtSearch Spider) and you'd like to link directly to the file from the result list, the physical path will need translated into a relative path to build a link to the file from the search result. This is accomplished in the XSLT.

If the files are located outside the web application, first you'll need to setup a virtual directory in IIS to point to the folder containing the files. For example, a folder called C:\PDF_Files\ might be setup as a virtual directory called PDFs located at <http://localhost/XSearchResults/PDFs/>

The sample XSLT google-style-v2.xsl provides a means to translate the physical path to a relative path. The following lines in google-style-v2.xsl can be edited to define one or more path translations:

```
<xsl:variable name="webroot1">C:\PDF_Files\</xsl:variable>  
<xsl:variable name="replacement1">PDFs/</xsl:variable>  
<xsl:variable name="webroot2" />  
<xsl:variable name="replacement2" />
```

In the above example, files in C:\PDF_Files\ are translated into a virtual directory PDFs, so that the search results link to files located at <http://localhost/XSearchResults/PDFs/>

Note: The physical path is case-sensitive. Also, make sure to use backslashes for the physical path, and forward slashes for the relative path.

Step 4: Customize XSLT (Optional)

While most search applications will not need customization, certain indexes with special uses may need search results presented in a non-standard way, or linked with other systems. To accomplish this, XSearchResults allows the user to define their own XSLT to construct and re-arrange a result page in almost any manner conceivable. For more, see “Customizing xSearchResults with XSLT” below.

Step 5: Begin Searching

To run a search, simply browse to the web application using a web browser (ex: <http://myserver/xSearchResults/>). Try experimenting with the XSL selection to see different formatted results. If your search returns zero results, make sure you index path is correct and the index isn't empty.

Using the Sample XSLTs

Three sample XSLTs are provided:

- 1) *google-style-v2.xsl* – Results are presented with the title at the top with a synopsis below for each result, showing relevant segments of the document.
- 2) *table.xsl* – Each field in the index is displayed in tabular format. Useful for database-only or hybrid indexes.
- 3) *table2.xsl* -- A few predefined custom fields (ex: CompanyName and TextComments) are displayed in a table. Useful for database-only indexes.

Adding the Control to the Toolbox in Visual Studio.net

For users who wish to build their search page from the bottom up or add the search control to multiple pages, XSearchResults can be added to the Toolbox in VS.net and used like any other ASP.net control. This will enable the design time features of VS.net such as drag-and-drop and the property pane.

To add the control to the Toolbox in VS.net, simply right-click on the Toolbox, click “Add/Remove Items”, click Browse, and find the DLL file packaged with the distribution (XSLTDevLib.dll). Click OK and confirm the addition; XSearchResult should now be an item in the Toolbox. When it is placed on the ASP.NET page, it will display sample search results indicating its position.

Integrating with an Existing Web Application

To integrate into an existing web application, first add a reference in the existing application to the XSLTDevLib.dll file found in the bin folder of the demo application. Then copy the text of the default.aspx file in the demo application to your current application, perhaps into a differently named file. Also copy the text of the default.aspx.vb file into the code-behind for the new file you're adding. Now you can modify the design section to include the options you want and style the page if necessary.

In the code-behind, most lines of code are self-explanatory, and span the gamut of possible customization. Remove lines of code for options you don't need, or replace the right-hand-side with a hard-coded value. Sensible defaults are used if any options aren't set in the code-behind, with the exception of XSLTFile, a path to the XSLT file to run. If UseXSLTFile is true, then the XSLT file listed there will be used to transform the XML search data into XHTML. If UseXSLTFile is false, then raw XML will be dumped into the control, something some browsers can't handle, but useful for debugging the XSLT file. The call to *GetResults* actually performs the search of the index, evaluates the XSLT, fills the XSearchResults control with the results, and places a search filter in the PreviousSearchFilter property. (This property allows you to search within your results, as shown in the sample application.) One advanced option is that in the case that you are using a file index without a cache, you will need to set the control's UseIndexCache property to False.

Customizing XSearchResults with XSLT

XSearchResults utilizes the dtSearch engine for conduct searches, and then builds an XML file from the data and runs it through an XSL transform engine to the result. The transform engine transforms the XML data into XHTML based on the XSL style sheet selected. The XSLT style sheet language, in case you are unfamiliar with it, provides you complete customization capabilities for how the results are display, which is especially useful when working with custom fields. XSLT can be used to insert CSS that may be applicable to your website.

The most important aspect of the program's structure, to a developer who is going to write an XSL file, is the structure of the XML data used. Here is the basic top-level structure of the XML data:

```

<results>
  <resultdata>
    ...
  </resultdata>
  <result>
    ...
  </result>
  ...
  <result>
    ...
  </result>
  <headerinfo>
    ...
  </headerinfo>
</results>

```

The result elements are returned in order by relevance. By default, there are 10 results per page. Starting from the top, the `<resultdata>` element contains some useful information about the search in general, which a style sheet may or may not make use of. With those that are not properties of the control explained, they are:

```

<count>(number of results returned)</count>
<searchtext>(original text of search)</searchtext>
<fuzzlevel></fuzzlevel>
<fuzzy>('True' or 'False')</fuzzy>
<includepredefinedfields>(see below)</includepredefinedfields>
<indexpath></indexpath>
<pagenumber></pagenumber>
<pagesize></pagesize>
<phonic></phonic>
<searchtype></searchtype>
<searchwithin></searchwithin>
<sortascending></sortascending>
<sorttype></sorttype>
<stemming></stemming>
<synonyms></synonyms>
<useindexcache></useindexcache>
<xsltfile></xsltfile>

```

The `<result>` element contains all the fields dtSearch provides for this result. Many of these fields are predefined by dtSearch, such as `displayname`, `name`, `date`, etc. Also, the program computes a `synopsis` field using the dtSearch engine, although it is not predefined. All these fields are prefixed `predef-`, i.e., `<predef-displayname>`, `<predef-date>`, `<predef-synopsis>`, etc. A complete listing of the predefined fields can be found at <http://support.dtsearch.com/webhelp/dtengine/> in the COM interface/SearchResults node.

Your index may also contain custom fields, especially if it was created by dbIndexer. If the predefined fields would only serve to complicate your style sheet, they can be removed by setting XSearchResults' IncludePredefinedFields property to false, a change reflected in the //ResultData/IncludePredefinedFields element of the XML. The <headerinfo> element is a “mock” result which contains all the fields in each result, but instead of data in those fields, it contains the appropriate header name for them. For predefined fields, this means using an _ instead of predef-. Below is a very simple XSL which illustrates the most basic possible result list:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="//ResultData"></xsl:template>
<xsl:template match="//HeaderInfo"></xsl:template>
<xsl:template match="//Result">
  <a href="{predef-docName}">
    <xsl:value-of select="predef-displayName" disable-output-escaping="yes"/>
  </a>
  <br/>
</xsl:template>
</xsl:stylesheet>
```

The first two templates cause the ResultData and HeaderInfo elements to translate into nothing; that is, to be ignored. The third template means that each result should generate a link on its own line to the document's location (called “docName” by dtSearch), with the text of that document's display name, with any &, <, or > symbols displayed as they are and not escaped into &, <, and >. No matter what your display needs may be, they can be satisfied by some such XSLT style sheet. Cybergroup also offers consulting services to assist you in defining and building a XSLT.

Final Changes Before Going Live

You will likely want to remove the XSLT selection options from the form and hard-code the XSL needed. If integrating into an existing website, you may want to use the website's CSS file and integrate the navigation framework of the site.

XSLT Editors

Customizing the result page requires knowledge of XSLT. While XSLT can be hand-coded, third-party tools may assist in designing and testing your XSLT. Two popular third party XSL editors are Altova StyleVision (http://www.altova.com/products_xsl.html) and Marrowsoft Xselerator (<http://www.marrowsoft.com/>).

XSearchResults API Reference

Properties

CaseInsensitive: If set to False, performs a case-sensitive search.

DataSource: The DataTable used internally to represent the results of the search. Each column is a field; each row is a result.

Fuzzy and **FuzzLevel:** Control the tolerance of the search; for example, searching for 'alphabet' with **Fuzzy** = True and **FuzzLevel** = 1 would also search for “alphaqet” or “albhabet”. Searching for 'alphabet' with fuzzy on and fuzzlevel at 3 would also find “alpkaqet”.

IncludePredefinedFields: Determines whether standard dtSearch fields such as docName or docDisplayName are passed through to the XSLT. Usually, for database searches or for searches where the results are displayed in table form, this should be false; while for text searches or searches where the results are displayed in list form, this should be true.

IndexPath: This is the location of the dtSearch index files to use for searching.

PageSize, **PageNumber**, and **Pages:** **PageSize** and **PageNumber** are two read-write properties which the user can set to determine the particular subset of search results shown. **Pages** is a read-only property computed based on the current **PageSize** and result count which represents the number of pages that these results would take up. See the included sample search application for an example of how to use these properties to create a pager.

Phonic: Controls phonic searching; for example, with **Phonic** = True, searching for “Smith” would also find “Smythe”.

PreviousSearchFilter: This allows the developer to create “Search Within Results” functionality, in conjunction with the **SearchWithin** property, described below. This property should be saved to a session variable after the initial search, and restored from it when the user triggers a “Search Within Results”.

SearchType: A String. Valid values are “allwords”, “anywords”, “phrase”, and “boolean”. In the “allwords” setting, dtSearch will search for any document containing each word in the search, in any order or proximity. In the “anywords” setting, dtSearch will search for any documents containing any of the words in the search query, not necessarily all of them in the same document. In the “phrase” setting, dtSearch will consider the entire search query like a single word, and search for documents containing the exact query. In the “boolean” setting, the user can use Boolean logic to specify a query. DtSearch provides the following guidance:

```
tart apple pie - the entire phrase must be present;
apple pie and pear tart - both phrases must be present
apple pie or pear tart - either phrase must be present
```

apple pie and not pear tart - only *apple* must be present
apple w/5 pear - *apple* must occur within 5 words of *pear*
apple not w/27 pear - *apple* must not occur within 27 words of pear
subject contains apple pie - finds *apple pie* in a *subject* field

Use parenthesis if the query contains more than one connector.

SearchWithin: If this property is set to True, and the **PreviousSearchFilter** property is set to a value obtained from it after a previous **GetResults()** call, then the results of the current search will be a subset of the results of the previous search.

SortType: A String. Valid values are “hits”, “date”, “name”, and “size”. If set to “hits”, the documents containing the most occurrences of the search query, or the highest score, will appear on top. If set to “date”, the most recently modified documents will appear on top. If set to “name”, the documents will be sorted in alphabetical order of their title. If set to “size”, the documents with the largest file sizes will appear on top.

Stemming: Controls the word stemming capability of dtSearch. For example, if **Stemming** = True, searches for “apply”, “applying”, “applier”, or “applies” are all equivalent.

Synonyms: Uses an English thesaurus to search for synonyms of the search query in addition to the search query itself.

UseIndexCache: If set, uses a dtSearch 7 index format cache to generate the synopsis of each result, instead of attempting to locate the result on the hard disk.

XSLTFile and **UseXSLTFile:** If the latter is set, the former is a path to the XSLT file to use to transform the raw XML data of the search results into XHTML to display inside the control. We provide “google-style-v2.xsl”, “table.xsl”, and “table2.xsl” as examples of such XSLT files, which you may be able to use as is. Also, see the Developers’ Guide for more information on creating XSLT files.

Methods

GetResults(SearchText As String)

Simply put, this function evaluates a search with the arguments determined by properties on the query string passed, and displays the results selected by the paging properties, transformed by the given XSLT if applicable.