Windows 7 Federated Search Provider Implementer’s Guide

Brandon Paddock, Alwin Vyhmeister

Microsoft Corporation

February 2009

Applies to:

   Windows® 7

Summary: This document describes how to build a basic web service that leverages the Windows OpenSearch provider for federated search. It explains best practices for developing these sources, enabling richer experiences using this and related extensibility features.

#### Legal Notice

This is a preliminary document and may be changed substantially prior to final commercial release of the software described herein.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is intended or should be inferred.

© 2008 Microsoft Corporation. All rights reserved.

Microsoft, MS-DOS, Windows, Windows NT, Windows Server, Windows Vista, Active Directory, ActiveSync, ActiveX, Direct3D, DirectDraw, DirectInput, DirectMusic, DirectPlay, DirectShow, DirectSound, DirectX, Expression, FrontPage, HighMAT, Internet Explorer, JScript, Microsoft Press, MSN, Outlook, PowerPoint, SideShow, Silverlight, Visual Basic, Visual C++, Visual InterDev, Visual J++, Visual Studio, WebTV, Windows Media, Win32, Win32s, and Zune are either registered trademarks or trademarks of Microsoft Corporation in the U.S.A. and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

#### Contents

Introduction 4

What is federated search? 4

Scope of this document 2

Overview - How It Works 2

The OpenSearch Standard 2

The Components Involved 2

Connecting to Your Web Service from Windows 3

Sending Queries and Getting Results 4

What You Need to Do 4

Supported Standards 5

Definitions 6

Step 1 – Enabling Your Data Store 8

Accepting the Search Request 8

Query Syntax 8

Authentication 8

Returning Search Results using RSS or Atom 8

Example of an RSS Feed Output 9

Allowed URL Protocols 10

How Windows Maps Items to File Types 10

Step 2 - Creating an OpenSearch Description File 13

Example OSD file 13

Standard Elements to Include 13

<ShortName> 13

<Url> Template for RSS/Atom results 14

<Url> Template for Web results 14

<Url> Template for Initial Query 14

URL Template Parameters 15

Paged Results 16

<Description> 17

Extended Elements to include 18

Maximum Result Count 18

Property Mapping 18

Customizing Windows Explorer Views 22

Previews 26

Open File Location Menu Item 27

Step 3 - Deploying Search Connectors 29

Pull Deployment 29

Push Deployment 30

Tracking Usage of Your Service 30

When Your Data Source Can’t Be Enabled 30

Middle-man Web Service 30

Using an Existing Search Engine 30

Client-side Data Source 31

Best Practices 31

Conclusion 32

For More Information 32

# Introduction

Windows® 7 introduces support for search federation to remote data stores using OpenSearch technologies that enable users to access and interact with their remote data from within Windows Explorer.

This document describes how to build a web-based data source that can be searched using Windows federated search. Following the best practices described here, you can enable rich integration of your remote data sources with Windows Explorer without having to write or deploy any Windows client-side code.

## What is federated search?

Federated search enables users to search remote data sources from within Windows Explorer. Remote data sources make themselves searchable with a simple web front end that exposes their search capabilities. In fact, some data sources may already support the minimum requirements for search federation.

**Figure 1** shows an example of searching an existing SharePoint 2007 site from within Windows Explorer. The user selects the SharePoint site search connector in the navigation pane rather than navigating to the SharePoint site in a browser.

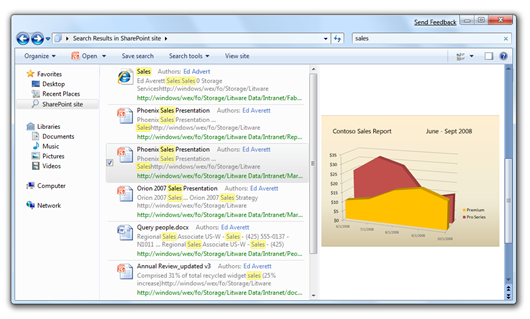


Figure : Searching a SharePoint site from Windows Explorer

The benefit of integrating your data source with Windows Explorer is that users can get at all of their information by using the already familiar Windows Explorer user interface. Your users will be able to see files from your remote data source just as they would local files, with the right application icons and context menus. They can preview documents or web pages, see thumbnails of images, and drag and drop a file directly to the desktop or into an email.

## Scope of this document

This document is intended for the following audiences who wish to enable their existing data stores to be searched from within Windows Explorer:

* Owners of enterprise data stores
* Owners of web services
* Owners of remote search providers
* Developers of 3rd party software products used in all of the above

This document covers the following topics:

* Accepting search queries from Windows
* Returning XML-formatted results back to Windows
* Deploying to users by providing an XML description document
* Enhancing the results returned (how to provide extra information about each item and how to show it in the Windows Explorer views)

# Overview - How It Works

Windows 7 supports hooking up external sources to the Windows Client via the OpenSearch protocol. This allows users to search a remote data store and see results within Windows Explorer.

## The OpenSearch Standard

The [OpenSearch v1.1 standard](http://www.opensearch.org/Specifications/OpenSearch/1.1) defines simple file formats that can be used to describe how a client should query the web service for the data store and how the service should return results to be rendered by the client. Windows federated search connects to web services that can receive OpenSearch queries and that can return results in either the RSS or Atom XML format.

If you are not familiar with these standards, you may need to refer to their specifications to better understand some parts of this document (see [definitions section](#_Definitions) for links to the specs for those formats).

## The Components Involved

The components involved in Windows federated search are shown in Figure 2. To enable your data store to be searched by Windows, you will need to build two of those components:

* The OpenSearch compatible web service
* The .osdx file

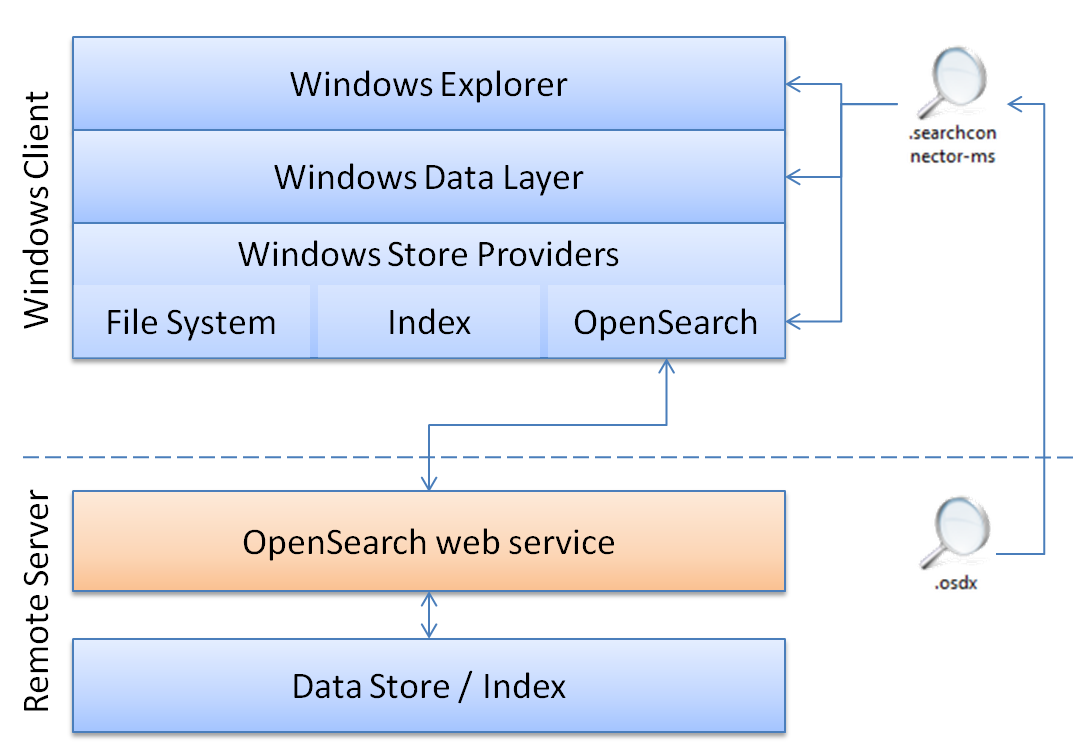


Figure 2: Windows Search Federation functionality

The communication between Windows Explorer and your OpenSearch web service is done through the Windows Data Layer, which can communicate with various data source types via what are called Windows Store Providers. Each provider specializes in communicating with data stores that support a particular protocol and have specific capabilities. The OpenSearch provider communicates with data stores that have a web service which supports the OpenSearch standard.

## Connecting to Your Web Service from Windows

A data source that already provides an OpenSearch web service that is compatible with Windows federated search can be added to Windows Explorer when users open up a Search Connector Description file (.osdx). The internal file format for a .osdx file is an OpenSearch Description XML document which is described later in this guide.

To register a new remote data source with Windows federated search, the end-user can open an .osdx file by clicking on a link to one placed on a web site or by opening one provided by someone else on a share or via an email attachment for example.

Doing that creates a .searchconnector-ms file in the Windows “Searches” folder (%userprofile%/Searches) and places a short-cut in the “Links” folder (%userprofile%/Links). This shortcut shows up in the Windows Explorer navigation pane favorites section, as shown in **Figure 3**. The user can then click this short-cut to navigate into the new data source and query the web service.

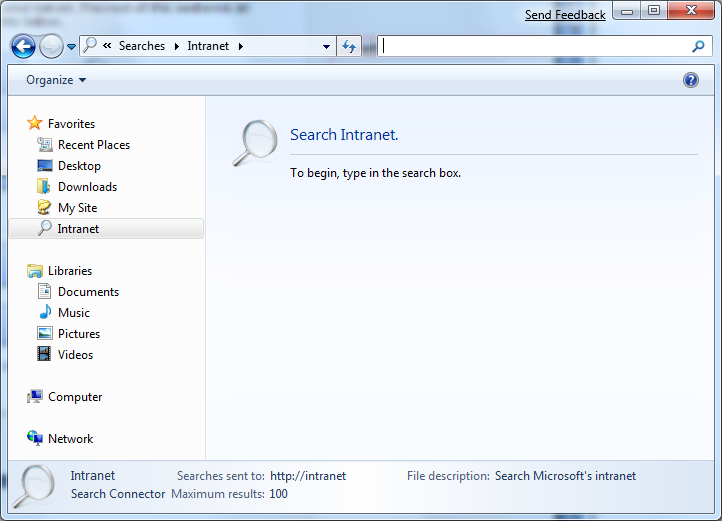


Figure 3: Newly installed Search Connector for an intranet site

## Sending Queries and Getting Results

When the user types a term into the search box in the upper right hand of Windows Explorer, the query is sent to the OpenSearch provider, which then sends the query to the remote data store. The remote web service responds to the query with results in an XML document formatted in either the RSS or Atom format. The OpenSearch provider parses the XML elements of the results into items which eventually make it back to Windows Explorer where they are displayed to the user.

# What You Need to Do

There are three steps to getting your data store’s web service connected to Windows 7, as shown in **Figure 4**.



Figure 4: Enabling, deploying, and querying a new OpenSearch data store

1. Enable the data store to be searched from Windows using OpenSearch with RSS or Atom output
2. Create .osdx files that describe how to connect to the web service and how to map any custom elements in your RSS or Atom XML
3. Deploy the search connectors to Windows client machines using the .osdx files.

The following table lists the tasks developers need to complete to support different Windows 7 federated search scenarios.

|  |  |
| --- | --- |
| If you want to: | Then you will need to: |
| Have Windows Explorer be able to enumerate search results in response to a query | * Implement a web service on your web server that accepts a query string and returns RSS or Atom formatted results (see “Accepting the Search Request”) |
| Allow users to add your location to their Windows Explorer | * Create an OpenSearch Description (.OSDX) file and provide it to your users  (see “”) |
| Have Windows Explorer show your items as file-like items | * Return a URL to the file or content stream using <enclosure> or <media:content> elements * Supply a file extension or a MIME type that the client machine will recognize   (see “”) |
| Have Windows Explorer show custom properties beyond what’s defined in the RSS or Atom standards | * Provide additional metadata using another XML namespace in your RSS/Atom output * Add a Property Map to your .OSDX file   (see “Property Mapping”) |
| Customize the properties Windows Explorer displays for your items | * Add proplist mappings to your .OSDX file(see “”) |
| Have the preview pane show a custom web page preview for your items. | * Return distinct <link> and <enclosure> values. * Map a URL value to the System.WebPreviewUrl Windows Shell property. (see “”) |
| Have Windows Explorer show a command bar button to roll-over the query to your web site | * Provide a <Url type="text/html"> template in the .osdx file (see “”) |

A list of [common terms](#_Definitions) and definitions is provided in the next section of this document.

### Supported Standards

Windows 7 search federation supports the following standards.

For item data:

* RSS 2.0, 1.0, 0.91 & 0.92
* Atom 0.3 & 1.0
* MediaRSS content and thumbnail elements

For connection information:

* OpenSearch 1.1

Authentication

* NTLM
* Kerberos
* Basic (only over https)
* plus any other Security Support Providers installed on the client and the server hosting the web service

# Definitions

Federated Search Provider

A web service implemented by a data source that supports the protocols used by Windows 7 to search that data source remotely.

Search Connector

A file-based namespace junction (with extension .searchConnector-ms) similar to saved searches (.search-ms) or library definitions (.library-ms).

OpenSearch

A collection of simple formats and protocols used for the sharing of search results. See opensearch.org website for more information on the [OpenSearch v1.1 spec](http://www.opensearch.org/Specifications/OpenSearch/1.1).

OpenSearch Description

An XML file that describes available web service connections and result formats for a specific web-based data source. This file contains one or more URL templates, and uses a .osdx file extension when interacting with the Windows shell.

RSS Feed

An XML result set returned by the web service in the [RSS format](http://www.rssboard.org/rss-specification).

Atom Feed

An XML result set returned by the web service in the [Atom format](http://tools.ietf.org/html/rfc4287).

Media RSS

An extension to RSS to include extra metadata about media content, such as images, audio and video. See <http://search.yahoo.com/mrss> for more information.

URL Template

At the core of the OpenSearch standard is the definition of URL templates, which are URL-based connection strings used to query a web service for search results. The template looks like a URL, but contains several placeholder values (such as {searchTerms}) into which the OpenSearch provider slot-fills data about the results it wishes to retrieve.

Item

A Shell Item, often just referred to as an Item, is the currency of the shell UI model and its programming model. Items are files, containers and things users interact with through the Windows Explorer. In some contexts, “Item” is used to distinguish non-containers from containers.

Container

An Item that holds other items, such as a file system directory. Containers expose the shell data source programming model that expresses the items and the functionality of those items to Windows Explorer, including the verbs for those items.

Shell Data Source

A component that exposes Shell Items and containers, and services these items by providing handlers, properties, and so on.

Context Menu

A collection of verbs that can be presented to a user or programmed using the IContextMenu programming model. The context menu for an item is usually a composition of verbs provided from different sources.

Handler

A piece of code that operates on an item. In the context of this document verb implementations and previewers are provided by handlers.

Verb

An individual action that can be invoked on an item, for example “open” or “print”. These are sometimes referred to as commands or tasks. Verbs can be implemented in many different ways.

Previewer

A handler that quickly produces a read-only, simplified view of the item to be displayed in the Windows Explorer’s preview pane.

# Step 1 – Enabling Your Data Store

Your data store must provide a web service that takes a query with a search term in it and returns results in the form of an RSS or Atom feed.

## Accepting the Search Request

The OpenSearch web service you create on the web server MUST accept a GET URL query from the client. The web service MUST allow the search terms to be embedded in the URL somewhere. Here’s an example of one way search terms may be embedded:

http://example.com/search.aspx?query=terms&param=foo

See the “URL Template Parameters” section later in this document for more information on how this is used by the OpenSearch provider to send the user’s query terms to the web service.

Note: Federated search does not support sending POST requests to a web service.

### Query Syntax

There is no specific query syntax expected by Windows 7.

Windows takes whatever the user types in the input box in Windows Explorer and encodes it into the URL using the template as described in the “URL Template Parameters” of this document.

**Note:** Users expect that separate terms will be treated as implicitly ANDed together. This means that a query for “Microsoft Windows” should return only a subset of the combined results of separate queries for “Windows” and “Microsoft”.

### Authentication

Windows Search federation supports Windows-based authentication. Currently it can provide credentials to web services using the following protocols:

* NTLM
* Kerberos
* Basic (only over https)

Other Security Support Providers (SSPs) added to the Windows client may also be supported in the future. See [SSP Interface](http://msdn.microsoft.com/en-us/library/aa380493(VS.85).aspx) SDK documentation.

(See “When Your Data Source Can’t Be Enabled” for other authentication options).

## Returning Search Results using RSS or Atom

After the web service processes the search request, the service should return a set of results to the OpenSearch provider. Those results are returned as an XML document. The two supported formats for this document are RSS and Atom. These types of documents are typically referred to as feeds, and they can contain multiple items in each document.

Each result item in the feed includes XML child elements to represent or describe the item’s metadata, like title, URL, description, thumbnail picture and so on. The OpenSearch provider is responsible for mapping the XML element values to Windows Shell System properties that can be used by Windows applications.

### Example of an RSS Feed Output

The following is an example of the feed file a web service would return. This example returns one result item.

<rss version="2.0" xmlns:media="http://search.yahoo.com/mrss/" xmlns:example=”http://example.com/namespace”>

<channel>

<title>Search Results</title>

<item>

<title>An example result</title>

<link>http://example.com/pictures.aspx?id=01</link>

<description>This is a test of the emergency search results system. If this were a real emergency result, you’d be reading something more useful.</description>

<pubDate>Wed, 1 Oct 2008 23:12:00 GMT</pubDate>

<media:content url="http://example.com/pictures/picture01.jpg" fileSize="212889" type="image/jpeg" height="768" width="1024"/>

<media:thumbnail url="http://example.com/thumbnails/picture01.jpg" height="120" width="160"/>

<example:dateTaken>Mon, 22 Sep 2008 23:12:00 GMT</example:dateTaken>

</item>

</channel>

</rss>

Items MUST return a <link> or <enclosure> value (or equivalent, such as <media:content>) for each item.

Follow these recommendations for the RSS output:

* SHOULD NOT include any HTML formatting tags in the <title> attribute or those tags will show up in the title. For example, if the value was “Title with <b>important</b> word”, the <b> and </b> tags would be displayed just like that in Windows Explorer.
* Description element:
  + SHOULD show enough information so the user knows why this result might be relevant. For example, if the user searches for “Windows” and the item has matched because it has this text inside it: “…the datacenter building has few windows which prevent…,” you should include that snippet in the description.
  + SHOULD NOT include HTML formatting. The OpenSearch provider removes the formatting which might result in less than desirable results for your description.
  + SHOULD NOT include metadata that’s already provided in other elements such as enclosure file name, size, modified date, etc. Windows Explorer already shows that metadata so this would be just redundant information.
* For enclosure or content URLs:
  + The type attribute MUST be specified.
  + The type attribute MUST be a valid MIME type
  + The file size MUST be specified in bytes.
* If you’re implementing this in .NET using DateTime, use date.ToString("R") to get the proper date format.
* Test your feed in Internet Explorer to see if the basics are valid. If it displays properly there, it should work in Windows Explorer.

### Allowed URL Protocols

URLs used in the feed (such as within the <item>, <enclosure>, <media:thumbnail> elements) must use only http:, https:, or file: protocols. URLs using other protocols such as mms:, mapi: are discarded.

In addition, the file: protocol is blocked if the URL template being used to query the server is in the “Internet” URL security zone (see [URL Security Zones](http://msdn.microsoft.com/en-us/library/ms537021(VS.85).aspx) on MSDN for more details).

## How Windows Maps Items to File Types

One of the great benefits of searching from Windows is the ability to treat results as files when the RSS item is pointing to a file stored remotely. The user can drag/drop items to the desktop, the Windows Explorer user interface will show the right icon for it, and right-clicking the item brings up the correct context menu for it, and so on. If the RSS item does not point to a remotely stored file, it’s treated as a link, and users can perform limited actions on it, such as saving a shortcut to it or opening it in the browser.

The overall steps for determining how to treat the item are the following:

1. Identify whether the item should be treated as a file or a web link (see **Figure 5**).
2. Identify the correct file extension to use, so the system knows what type of file it is

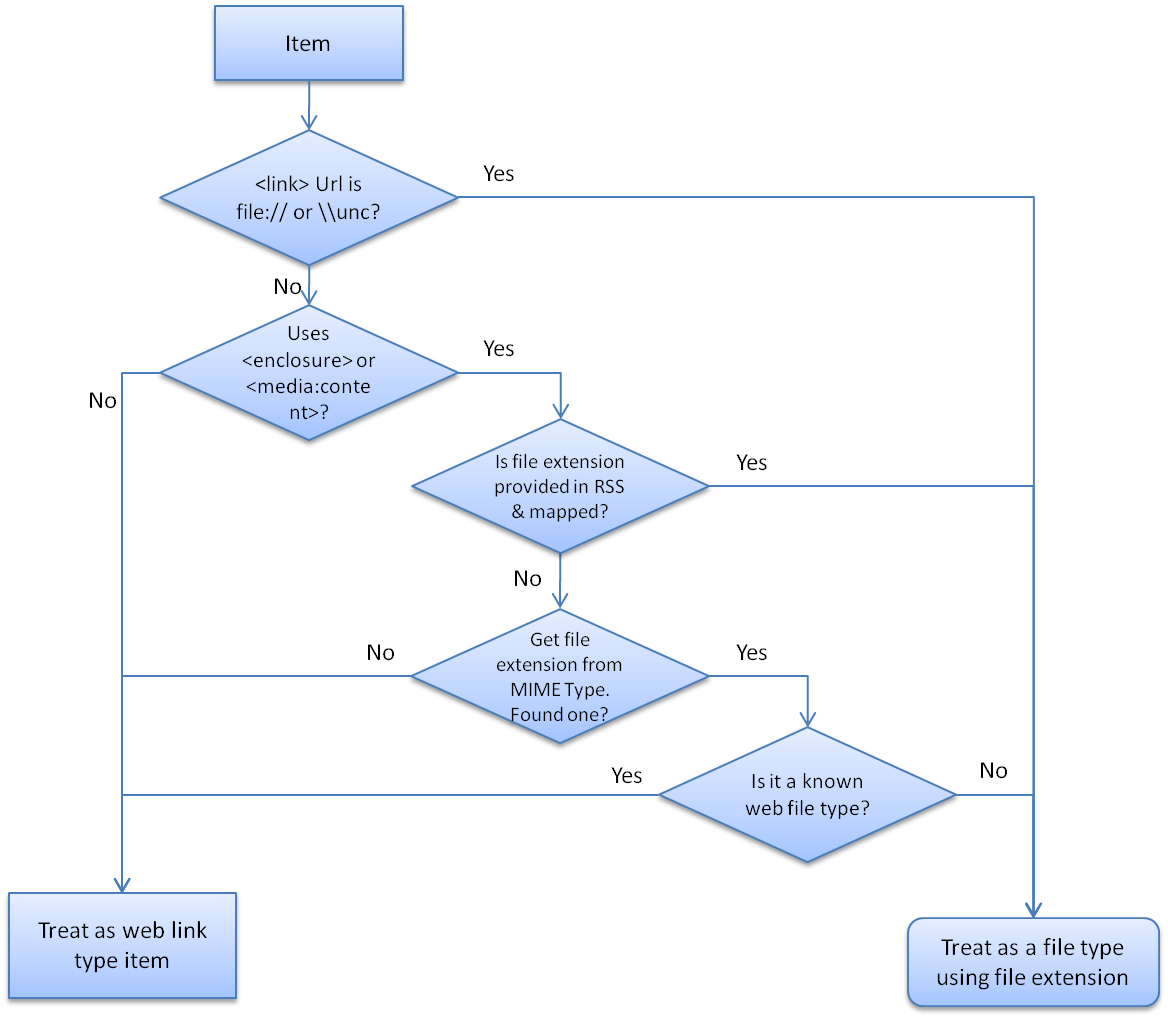


Figure : Steps to determine an item’s file type

**Figure 5** describes the entire process of determining an item’s file type. If the item has a <link> URL that uses a file system path such as <file:///\\server\share\etc\item.ext>, the OpenSearch provider treats the link as a file and determines the type by the file extension used in the path (.ext in this example).

If the item uses the standard RSS <enclosure> element or MediaRSS <media:content> element, the provider assumes that it should treat the item as a file. Then the provider identifies the correct file extension:

* If the System.FileExtension Windows Shell property has been mapped for the item, the provider uses that file extension.
* If the System.FileExtension Windows Shell property has not been mapped, the provider uses the “Type” attribute specified in the enclosure or content element. This element should contain a MIMEType string, such as “image/jpeg”. If the MIMEType is associated with a file extension registered on the client machine, the item is treated as a file of that type. If the MIMEType is not associated with a file extension registered on the client machine, the item is treated as a web link type.

Note: The provider doesn’t attempt to parse the URL attribute to find the file extension because this is relatively error prone for web URLs. For example this URL: <http://www.wikipedia.com/wiki/.ppt> is not a PowerPoint file, just a Wikipedia article about the file format. Also, some URLs that don’t have a .ppt extension at the end really are PowerPoint files such as this URL: <http://www.ldonline.org/?module=uploads&func=downlsoad&fileId=749>.

* If the MIMEType is associated with a file extension registered on the client machine, the provider then determines whether the file extension is a known web file type: .htm, .html, .asp, .aspx, .php, .swf, .stm. If yes, the item is treated as a web link type. Otherwise, it’s treated as a file type.

For example, the MIMEType “text/html” is associated with the .htm file extension, which is on the list, so an item with that MIMEType is treated as a web link instead of as the .htm file type.

# Step 2 - Creating an OpenSearch Description File

You should create an OpenSearch Description (OSD) file for your web service to enable users to add your SearchConnector for your remote data store to their Windows Explorer experience. OSD files specify a web service’s short name and the URL type and the template for sending queries to the web service.

OSD files:

* MUST be a valid OpenSearch Description document, following the OpenSearch 1.1 specification.
* MUST provide a URL template with either an RSS or an Atom format type.
* MUST use the .OSDX file extension, or when downloading from the web, the application/opensearchdescription+xml MIME content type MUST be associated with it.
* Should provide a "ShortName" value.

## Example OSD file

The following is an example of a minimal OpenSearch Description file for a web service called "My Web Service." The <Url> element and <ShortName> are the minimum set of child elements required to create an OSD file that Windows Explorer can use.

<OpenSearchDescription xmlns="http://a9.com/-/spec/opensearch/1.1/">

<ShortName>My Web Service</ShortName>

<Url type="application/rss+xml" template="http://example.com/rss.php?query={searchTerms}&amp;start={startIndex}&amp;cnt={count}" />

</OpenSearchDescription>

## Standard Elements to Include

The example above includes only the minimal set of elements required to describe the connection. This section describes the elements supported by Windows.

### <ShortName>

<OpenSearchDescription xmlns="http://a9.com/-/spec/opensearch/1.1/">

<ShortName>My Web Service</ShortName>

...

</OpenSearchDescription>

The ShortName element value is used by Windows to name the .searchconnector-ms file that’s created when the user opens the .osdx. Windows ensures that the created file name uses only characters allowed for Windows file names.

If no ShortName is provided, the .searchconnector-ms file will attempt to use the file name of the .osdx file instead.

### <Url> Template for RSS/Atom results

<OpenSearchDescription xmlns="http://a9.com/-/spec/opensearch/1.1/">

...

<Url type="application/rss+xml" template="http://example.com/rss.php?query={searchTerms}&amp;start={startIndex}&amp;cnt={count}" />

</OpenSearchDescription>

The OSD file must include one URL template that returns results in either RSS or Atom format. The “type” attribute must be set to “application/rss+xml” for RSS formatted results or “application/atom+xml” for Atom formatted results.

### <Url> Template for Web results

<OpenSearchDescription xmlns="http://a9.com/-/spec/opensearch/1.1/">

...

<Url type="text/html" template="http://example.com/html.php?query={searchTerms}"/>

</OpenSearchDescription>

If there is a version of the search results which can be viewed in a web browser, you should provide a text/html type URL template. Then when the user does a query, a button like the one shown in **Figure 6** appears in the Windows Explorer command bar that allows the user to launch the web browser to view the search results.



Figure : Web search roll-over command bar button

This rolling-over of the query back to the data source’s own web UI is important for users in some scenarios. For example, users may want to view more than 100 results, which is the default number of items the OpenSearch provider will request. They might also want to use search features only available on the data source’s web site such as re-querying with a different sort order or pivoting and filtering the query with related metadata.

### <Url> Template for Initial Query

<OpenSearchDescription xmlns="http://a9.com/-/spec/opensearch/1.1/">

...

<Url type="text/html" rel="collection" template="http://example.com/latest.php" />

</OpenSearchDescription>

In some special cases, you may want to provide a set of results immediately upon navigating into a search connector without the user having to type a query term. This could be used to always show users the twenty most recent purchase orders they created or the most recent pictures they uploaded to your picture sharing server.

In such cases, the Url template should not include a {searchTerms} token.

When using one of these Urls, it is recommended to also provide a searchable <Url> template as well so that when the user types a search term in the search box, a new query is sent to the server to get all results matching that term.

Otherwise, if there is no <Url> template that includes a {searchTerms} and returns RSS, when the user types a query in the search box, only the results already in memory are matched against the user’s search terms. They see just a subset of the items already returned by the initial query.

Make sure that these two issues don’t create false negatives for your users:

* The matching is done only against the metadata in memory for these items. If there are other files that would match but are not in the initial query results, they do not appear in the results.
* Servers typically match search terms against file contents, which isn’t possible in this case since the file contents are not available. Items with the search terms inside of the file contents will not be matched.

Here are a few sample .osdx files that show this being used:



Tip: Notice that the KEXP .osdx file has a mapping from the <description> tag to System.Search.AutoSummary and System.Comment. This is necessary so that searches find matches within the description. The System.Search.AutoSummary property is not searched when doing an in-memory search as in this case. Putting the same content in the Comment property allows the in-memory search to find hits from within that property.

### URL Template Parameters

The sample OSD file above includes a “Url” element with a “template” attribute. The URL template is used by the OpenSearch provider to send the request to the web service. The provider attempts to replace the tokens it finds in the URL template before sending the request.

The provider knows how to replace the standard OpenSearch tokens in the following table. Any other tokens found are removed before sending the query to the web service.

|  |  |
| --- | --- |
| **Token Supported** | **How it’s used by the OpenSearch provider** |
| {searchTerms} | Replaced with the search terms the user types in the Windows Explorer search input box |
| {startIndex} | Used when getting results in “pages.”  Replaced with the index for the first result item to return. |
| {startPage} | Used when getting results in “pages.”  Replaced with the page number of the set of search results to return. |
| {count} | Used when getting results in “pages.”  Replaced with the number of search results per page that Windows Explorer is asking for. |
| {language} | Replaced with a string that indicates the language of the query being sent. |
| {inputEncoding} | Replaced with a string (such as “UTF-16”) that indicates the character encoding of the query being sent. |
| {outputEncoding} | Replaced with a string (such as “UTF-16”) that indicates the desired character encoding for the response from the web service. |

### Paged Results

You may not want your web service to return more than a limited number of results per request. If so, you can choose to return just a “page” of results at a time. You can allow the OpenSearch provider to get additional pages of results either by item number or page number.

For example, if you send twenty results per page, the first page you send starts at item index 1 and at page 1; the second page you send starts at item index 21 and at page 2. You define how you want clients like the OpenSearch provider to request items by either using the startItem or the startPage token in the URL template.

#### Paging Using Item Index

An item index identifies the first result item in a page of results. If you want clients to send requests using an item index, you can use the {startIndex} token in your URL template attribute:

<Url type="application/rss+xml" template="http://example.com/rss.php?query={searchTerms}&amp;start={startIndex}" />

Then the OpenSearch provider replaces the token in the URL with a starting index value. The first request starts with the first item, as shown in the following example:

http://example.com/rss.php?query=frogs&start=1

The OpenSearch provider can get additional items by changing the startIndex parameter value and issuing a new request. The provider repeats this process until it gets enough results to satisfy its limit, or reaches the end of the results.

The OpenSearch provider looks at the number of items returned by the web service in the first page of results and sets the expected page size to that number. It uses that number to increment the startIndex value for subsequent requests.

For example, if the web service returns 20 results in the first request, the provider sets the expected page size to 20. For the next request, the provider would replace {startIndex} with the value of 21 to get the next 20 items.

When a page of results returned by the web service has fewer items than the expected page size, the OpenSearch provider assumes it has received the last page of results and stops making requests.

#### Paging Using Page Index

A page index identifies a specified page of results. If you want clients to send requests using a page number, you can use the {startPage} token in your URL template attribute:

<Url type="application/rss+xml" template="http://example.com/rss.php?query={searchTerms}&amp;page={startPage}" />

Then the OpenSearch provider replaces the token in the URL with a page number parameter. The first request starts with the first page, as shown in the following example:

http://example.com/rss.php?query=frogs&page=1

The OpenSearch provider can use this template to get multiple pages of results, one after the other, by increasing the startPage value by 1 and resending the request.

Note: The provider assumes that all pages contain the same number of items. If a subsequent page is returned with fewer items, it assumes it is the last page of results.

#### Page Size

The web service may want to allow a request to specify the size of the pages using some parameter in the URL. This must be specified in the OSD file using the {count} token:

<Url type="application/rss+xml" template="http://example.com/rss.php?query={searchTerms}&amp;start={startIndex}&amp;cnt={count}" />

The OpenSearch provider can then set the desired page size (in terms of number of results per page) as in the following example:

http://example.com/rss.php?query=frogs&start=1&cnt=50

By default the OpenSearch provider makes requests using a page size of 50. If you would like to use a different page size, then do not provide a {count} token and instead place the desired number directly in the URL template.

The provider determines the page size based on the number of results returned on the first request. If the first page of results received has fewer items than the count requested, the provider resets the page size for any subsequent page requests. If subsequent page requests return fewer items, it assumes that it has reached the end of the results.

### <Description>

<OpenSearchDescription xmlns="http://a9.com/-/spec/opensearch/1.1/">

...

<Description>Searches the example company book catalog</Description>

</OpenSearchDescription>

Windows uses the Description element value to populate the file description shown in the Windows Explorer details pane when users select a .searchconnector-ms file.

## Extended Elements to include

To improve how the OpenSearch provider works with remote data stores, we support these extra child elements in the OSD file:

* MaximumResultCount
* ResultsProcessing

Because these child elements are not supported in the OpenSearch v1.1 specification, they must be added using this namespace:

http://schemas.microsoft.com/opensearchext/2009/

### Maximum Result Count

Often there are more results than you would want the web service to send back to the user. Search Connectors by default only get up to 100 results for any given user query. This limit can be customized by including the MaximumResultCount element within the OSD file as shown in the following example:

<OpenSearchDescription xmlns="http://a9.com/-/spec/opensearch/1.1/" xmlns:ms-ose="http://schemas.microsoft.com/opensearchext/2009/">

...

<ms-ose:MaximumResultCount>200</ms-ose:MaximumResultCount>

</OpenSearchDescription>

Note: This is not a standard OpenSearch Description property so the namespace MUST be declared using this URI: "http://schemas.microsoft.com/opensearchext/2009/”. The above example declares the namespace prefix “ms-ose” in the top-level OpenSearchDescription element and then uses it to prefix the element name.

### Property Mapping

When results are returned by the web service as RSS or Atom feeds, the provider needs to map the metadata in the feed’s items to properties that the Windows Shell can use. **Figure 7** illustrates how the OpenSearch provider maps some of the default RSS elements.

Figure 7: Built-in RSS to Windows Shell Property Mappings

The OpenSearch provider maps some standard RSS or Atom elements by default (as shown in the Default Mappings list). In addition to these default mappings, you can map other Windows Shell system properties by including additional XML elements in the Windows namespace (more information below) for each of the properties. You can also map elements from other existing XML namespaces such as MediaRSS, Itunes, and so on, by doing custom property mapping in the .osdx file (explained later in this document).

#### Default Mappings

The following are the default mappings of RSS XML elements to System properties. XML paths are relative to the <item> element.

| RSS XML Path | Windows Shell Property (Canonical Name) |
| --- | --- |
| Link | System.ItemUrl |
| Title | System.ItemName |
| Author | System.Author |
| pubDate | System.DateModified |
| Description | System.AutoSummary |
| Category | System.Keywords |
| enclosure/@type | System.MIMEType |
| enclosure/@length | System.Size |
| enclosure/@url | System.ContentUrl |
| media:category | System.Keywords |
| media:content/@fileSize | System.Size |
| media:content/@type | System.MIMEType |
| media:content/@url | System.ContentUrl |
| media:group/content/@fileSize | System.Size |
| media:group/content/@type | System.MIMEType |
| media:group/content/@url | System.ContentUrl |
| media:thumbnail/@url | System.ImageParsingName |

The “media:” prefix is defined by the <http://search.yahoo.com/mrss/> namespace.

Important system properties are listed on [System-Defined Properties for Custom File Formats](http://msdn.microsoft.com/en-us/library/dd368864(VS.85).aspx) page on MSDN.

#### Automatic Windows Shell Property Mappings (RC & RTM)

Note: The Beta version of Windows 7 does not support automatic mappings, so you’ll need to use [Custom Property Mapping](#_Custom_Property_Mapping) until the RC versions or later of Windows 7.

You can choose to include other XML elements within the items in your RSS feed that automatically map to Windows Shell system properties. To do this, include an element named after the Windows Shell property and prefixed with the Windows Shell system namespace. This example shows the namespace declaration (win=” http://schemas.microsoft.com/windows/2008/propertynamespace”) and the inclusion of an element for the property mapping (win:System.Contact.PrimaryEmailAddress):

<rss version="2.0" xmlns:example="http://example.com/schema/2009" xmlns:win="http://schemas.microsoft.com/windows/2008/propertynamespace">

...

<item>

<title>Bill Gates</title>

<win:System.Contact.PrimaryEmailAddress>billg@microsoft.com

</win:System.Contact.PrimaryEmailAddress>

</item>

The namespace prefix used here (“win”) is just a suggestion; you MAY use any prefix. However, you MUST use the exact URI:

http://schemas.microsoft.com/windows/2008/propertynamespace

and exact Windows Shell property names for this mapping to work correctly.

##### List of Windows Shell System Properties

See the [MSDN Shell properties documentation](http://msdn.microsoft.com/en-us/library/bb763010(VS.85).aspx) for a complete listing of Windows Shell properties defined in Windows and the value type format required for each property.

As an example, the documentation for the [System.FileExtension](http://msdn.microsoft.com/en-us/library/bb760699(VS.85).aspx) Window Shell property specifies that the value MUST contain the leading period (for example, “.docx” and not “docx”).

##### Date + Time Values

The preferred date + time format is ISO-8601, for example: 2008-01-16T 19:20:30:.45+01:00. .NET developers SHOULD use DateTime class with ToString("R") to output the correct format.

#### Custom Property Mapping

You can customize the mapping of elements from your RSS output to Windows system properties by specifying the mapping in the .osdx file. The RSS output specifies an XML namespace and, for any child element of an item, an element name to map against. The OSD file identifies a Windows Shell property for each element name in a namespace.

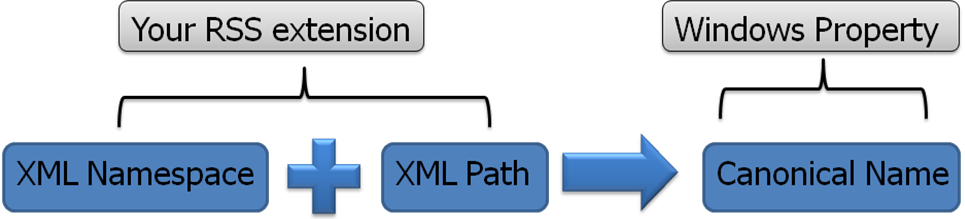


Figure 8: RSS extensions use XML namespace and paths to map to Windows properties.

#### Example RSS results and OSD Property Mapping

This example RSS output identifies http://example.com/schema/2009 as the XML namespace with the prefix example. This prefix MUST appear again before the element email.

<rss version="2.0" xmlns:example="http://example.com/schema/2009">

...

<item>

<title>Bill Gates</title>

<example:email>billg@microsoft.com</example:email>

</item>

This example .osdx file maps the XML element email to the Windows Shell property System.Contact.EmailAddress.

<OpenSearchDescription xmlns="http://a9.com/-/spec/opensearch/1.1/" xmlns:ms-ose="http://schemas.microsoft.com/opensearchext/2009/">

...

<ms-ose:ResultsProcessing type="application/rss+xml">

<ms-ose:PropertyMapList>

<ms-ose:PropertyMap sourceNamespaceURI="http://example.com/schema/2009/" >

<ms-ose:Source path="email">

<ms-ose:Property schema="http://schemas.microsoft.com/windows/2008/propertynamespace" name="System.Contact.EmailAddress" />

</ms-ose:Source>

</ms-ose:PropertyMap>

</ms-ose:PropertyMapList>

</ms-ose:ResultsProcessing>

...

</OpenSearchDescription>

The property mappings that you define in your .osdx file override the default mappings, if they exist, for those specified properties.

There are some properties that cannot be mapped because values for them are either overridden later or not editable. For example System.ItemPathDisplay or System.ItemPathDisplayNarrow cannot be mapped because they are calculated from the URL value provided in either the link or enclosure elements.

#### Thumbnails

Thumbnail image URLs may be provided for any item using the <media:thumbnail url="" /> element. They look best when they’re around 150x150 pixels. The largest thumbnails supported are 256x256 pixels. Providing larger images will take more bandwidth for no extra benefit to the user.

#### Open File Location Context Menu

Windows provides a context menu named “Open file location” for result items. If the user selects that menu item, the “parent” URL for the selected item is opened. If the URL is a web URL, such as http://..., the web browser is opened and navigated to that URL. Your feed should provide a custom URL for each item to ensure that Windows opens a valid URL. This can be done by including the URL within an element inside the item’s XML, as shown in the following example:

<rss version="2.0" xmlns:example="http://example.com/schema/2009" xmlns:win="http://schemas.microsoft.com/windows/2008/propertynamespace">

...

<item>

<title>Bill Gates</title>

<link>http://example.com/pictures.aspx?id=01</link>

<win:System.ItemFolderPathDisplay>http://example.com/pictures\_list.aspx

</win:System.ItemFolderPathDisplay>

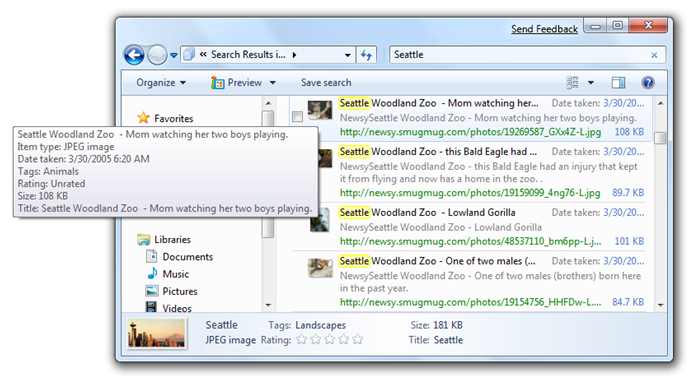
</item>

...

If this property is not explicitly set in the item’s XML, the provider sets it to the parent folder of the URL of the item. For our example above, the provider would have used the <link> value and set the System.ItemFolderPathDisplay Windows Shell property value to “http://example.com/”.

### Customizing Windows Explorer Views

Some of the Windows Explorer view layouts are defined by property description lists, or proplists. A proplist is simply a semicolon-delimited list of properties (such as “prop:System.ItemName; System.Author”) that are used to control how your results appear in Windows Explorer. These areas of Explorer can be customized using proplists, as shown in **Figure 9.**



Items shown using the “Content” view mode.

Infotip shown when hovering over an item with the mouse

Details Pane

Figure : Areas of Windows Explorer that you can control with proplists

Each part of Windows Explorer has an associated set of proplists, which themselves are specified as properties, as shown in the following table. You can specify custom proplists for individual items in your result sets or for all items in a set of results. These custom proplists affect how your results appear in Windows Explorer.

|  |  |
| --- | --- |
| User Interface part to customize | Windows Shell property that customizes it |
| Content view mode (when searching) | System.PropList.ContentViewModeForSearch |
| Content view mode (when browsing) | System.PropList.ContentViewModeForBrowse |
| Tile view mode | System.PropList.TileInfo |
| Details pane | System.PropList.PreviewDetails |
| Infotip (item’s hover tool tip) | System.PropList.InfoTip |

To specify a unique proplist for an individual item, follow these instructions:

1. In your RSS output, add a custom element representing the proplist you want to customize. For example, this sets the list for the details pane:

<win:System.PropList.PreviewDetails>prop:System.ItemName;System.Author</win:System.PropList.PreviewDetails>

1. For Beta clients, you’ll also need to map the extension to the same proplist property in the .osdx file, as shown in the [Property Mapping](#_Custom_Property_Mapping) section of this paper.

To apply a property such as one of the proplists above to every item in the search results without modifying the RSS output, specify a proplist within a <ms-ose:PropertyDefaultValues> element in the .osdx file:

<ms-ose:ResultsProcessing type="application/rss+xml">

<ms-ose:PropertyDefaultValues>

<ms-ose:Property schema="http://schemas.microsoft.com/windows/2008/propertynamespace" name="System.PropList.ContentViewModeForSearch">prop:~System.ItemNameDisplay;System.Photo.DateTaken;~System.ItemPathDisplay;~System.Search.AutoSummary;System.Size;System.Author;System.Keywords</ms-ose:Property>

</ms-ose:PropertyDefaultValues>

</ms-ose:ResultsProcessing>

#### Content View Mode Layout of Properties

The list of properties specified in the System.PropList.ContentViewModeForSearch and System.PropList.ContentViewModeForBrowse proplists determines what is shown in Content view mode.

##### Layout for Windows 7 Beta

For Windows 7 Beta, the properties are laid out according to the numbers shown in the layout pattern shown in **Figure 10**.

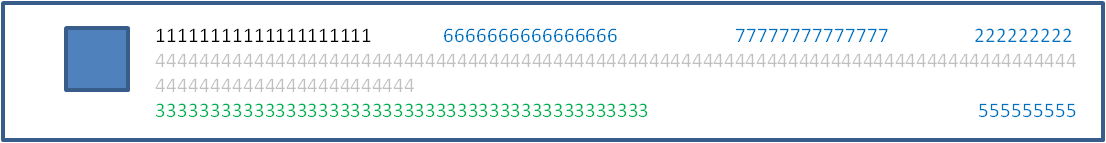


Figure : OpenSearch items default layout pattern in content view mode (Beta)

The first property in the list is placed where the “1111111” placeholder is in the layout pattern. For example, if we use this list of properties:

prop:~System.ItemNameDisplay;System.Author;System.ItemPathDisplay;~System.Search.AutoSummary;~System.Size;~System.Photo.DateTaken;System.Keywords

with this sample result in RSS format:

<rss version="2.0" xmlns:exif="http://www.exif.org/specifications.html" xmlns:media="http://search.yahoo.com/mrss/">

<channel>

...

<item>

<title>Seattle Zoo - King of the beasts!</title>

<link>http://my.contoso.com/gallery/09823123oilk1234kjh1234.jpg</link>

<description>What a shot of this majestic animal! Like he's saying - who are you to even look at me.

He was obviously warm and in no hurry to go anywhere. Master of his domain.</description>

<category>animals</category>

<category>zoo</category>

<category>lion</category>

<pubDate>Tue, 27 May 2008 16:21:10 -0700</pubDate>

<author>contoso guy</author>

<exif:DateTimeOriginal>2005-03-30 16:02:19</exif:DateTimeOriginal>

<media:content url="http://my.contoso.com/gallery/09823123oilk1234kjh1234.jpg" fileSize="99260" type="image/jpeg" medium="image" width="800" height="600">

<media:hash algo="md5">de7b0f71c0887e6ad7a4565523bd2eaf</media:hash>

</media:content>

<media:thumbnail url="http://my.contoso.com/gallery/09823123oilk1234kjh1234\_t.jpg" width="150" height="113"/>

</item>

...

We will see the item displayed as shown in **Figure 11**:

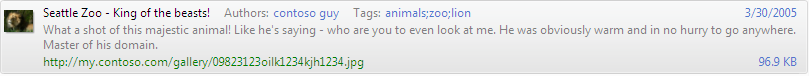


Figure : Sample property layout for OpenSearch items in content view mode (Beta)

The fifth property in the proplist is ~System.Size and it appears where the “55555555” placeholder is in the layout pattern.

NOTE: The reason you see the date 3/30/2005 displayed in this example instead of 5/27/2008 is because the OSD file (not shown) maps <exif:DateTimeOriginal> to the Windows Shell property System.Photo.DateTaken.

The layout patterns are also flexibly defined for various view widths. You’ll notice that as you make the Windows Explorer smaller horizontally, the layout pattern hides some properties to accommodate the smaller view.

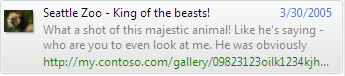


Figure : Resized window

##### Layout for Windows RC and RTM

For Windows 7 RC and RTM, the properties are laid out according to the numbers shown in this layout pattern:

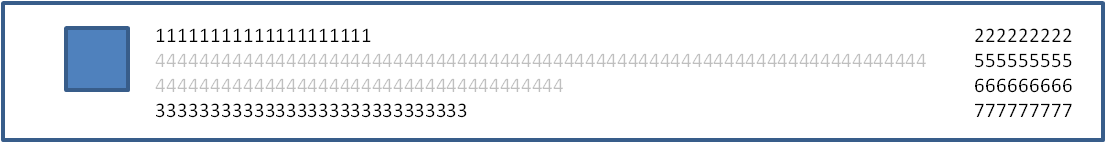


Figure : OpenSearch items default layout pattern in content view mode (RC & RTM)

If we use this list of properties:

prop:~System.ItemNameDisplay;System.Author;System.ItemPathDisplay;~System.Search.AutoSummary;System.Size;System.Photo.DateTaken;System.Keywords

We see the item displayed as shown in **Figure 14**.

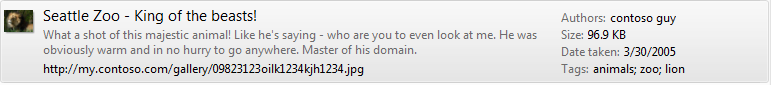


Figure 14: Sample property layout for OpenSearch items in content view mode (RC & RTM)

##### Leaving a Blank Space in the Layout

If you want to leave a blank space for one of the slots in the layout pattern, you should use this property in the proplist: System.LayoutPattern.PlaceHolder. For example, this would leave the 555555 placeholder blank:

prop:~System.ItemNameDisplay;System.Author;System.ItemPathDisplay;~System.Search.AutoSummary;System.LayoutPattern.PlaceHolder;System.Photo.DateTaken;System.Keywords

It is not enough to just remove the property and leave an empty “…;;…” item in the proplist.

#### Property List Flags

Only one of the flags defined in the [proplists documentation](http://msdn.microsoft.com/en-us/library/bb761436(VS.85).aspx) apply to the display of items in Content View mode layouts: “~”.

##### Hiding Labels With The ~ Prefix

In the previous examples, the Explorer view labels some of the properties, such as Tags: animals; zoo; lion. That is the default behavior when you specify a property in the list. For example, the proplist has “System.Author” which is displayed formatted as “Authors: value”. When you want to hide the property label, place a “~” in front of the property name. For example, if the proplist has “~System.Size”, the property is displayed as just a value, without the label.

For best readability, we recommend that the properties shown in the right hand column be labeled.

## Previews

When the user selects a result item in Windows Explorer and the preview pane is open, the content of the item is previewed.

The content to show in the preview is specified by a URL which is determined as follows:

* If the System.WebPreviewUrl Windows Shell property is set for the item, use that URL.   
  (Note: The property would need to be provided in the RSS using the Windows Shell namespace or explicitly mapped in the .osdx file. The URL can only use the http or https protocol. The file: protocol is not supported for WebPreviewUrl.)
* If not, use the <link> URL instead.

Figure : How Windows Explorer identifies which URL to use for previews

It is possible to use a different URL for the preview than for the item itself. This means that if you provide different URLs for the <link> URL and the <enclosure> or <media:content> URL, Windows Explorer uses the <link> URL for previews of the item but uses the other URL for file type detection, opening, downloading, and so on.

If the item’s URL uses a file: protocol, the default previewer associated with the file type is used. Note that there is no default preview handler for images.

## Open File Location Menu Item

When users right-click an item, they see an “Open file location” menu option. This command takes the user to the container for or location of that item. For example, on a SharePoint search, selecting this for a file in a document library would open the document library root in the web browser.

When this menu item is invoked, Windows Explorer attempts to find a parent container, as shown in **Figure 16**.

Figure : How Windows Explorer identifies a parent container

Here is how Windows Explorer determines what URL to use:

* If you provide a mapping to System.ItemFolderPathDisplay, Explorer uses that URL.
* If you don’t provide a mapping, Explorer identifies whether the <link> and <enclosure> URLs are different. If so, Explorer uses the <link> URL.
* If the URLs are the same or if there’s only a <link> URL, Explorer parses the <link> to find the parent container by removing the filename from the full URL.

**Note**: If you recognize that this URL parsing would result in dead links for your service, you SHOULD provide an explicit mapping for this property.

# Step 3 - Deploying Search Connectors

Opening up one of the .osdx files that you created in step 2 creates a .searchconnector-ms file or “Search Connector” in the %userprofile%/searches directory and places a link to it in the %userprofile%/links directory.

Once the .osdx and .searchconnector-ms files have been created, deployment can be accomplished on an enterprise scale through one of several methods. Each of these methods must be evaluated in light of the individual goals and scale of your organization’s planned rollout of Windows 7.

It is important to understand that search connectors are just files. They can be deployed to users as files and do not require any special installation or configuration. The process for adding a search connector to a user’s machine is essentially copying that file locally and exposing it to the user via a shortcut. The methods outlined here describe different ways of accomplishing this.

**Deployment methods:**

1. Pull Deployment
2. Push Deployment

There is a separate document covering enterprise deployment of search connectors that goes into more depth on each of these methods. Here’s a brief description of each of these methods.

## Pull Deployment

Pull deployment describes any scenario in which the end-user must take the initiative to install the search connectors. This is generally the easiest and least costly deployment method. There are potentially many more methods of pull deployment, but the most common scenarios are by attaching the .osdx file in an email, posting the file on a web page, or providing a dynamic link on your site that generates custom .osdx files based on user choices or the current scope within a site, for example.

Note that for the file to be downloaded when the user clicks the link in their browser, the web server hosting the web service must be set up to deliver the .osdx as a file. You will need to configure the MIME Types on the web server to treat .osdx files as “application/opensearchdescription+xml”.

You can use the icon Microsoft uses in Windows for search connectors to identify the link on your web site:



The images are also attached in this embedded zip file:



You should also consider making this link hidden for users viewing the web site on other Operating Systems or older versions of Windows. The version for Windows 7 is 6.1. The browser agent string contains “Windows NT 6.1”.

## Push Deployment

One of the other options for deploying search connectors in your organization is a push deployment model. This model encompasses any deployment method that does not depend on user initiative to install search connectors. While there are many potential methods of pushing search connectors to users, the most common methods are deployment through Group Policy Preferences, a logon script, Roaming profiles or imaging.

## Tracking Usage of Your Service

Once you’re up and running if you want to track the usage of your OpenSearch service by users searching from Windows Explorer, you can filter your web server log files for this user agent string:

“Windows-Search+(Windows+NT+6.1)”

# When Your Data Source Can’t Be Enabled

There are cases when the data store can’t provide an OpenSearch compatible web service, for example:

* Remote indexes with authentication methods which Windows 7 search federation doesn’t support, like forms-based authentication or other custom authentication methods.
* High value public stores of vertical data which are not controlled by the developer (Library of Congress, medical research databases, etc.) and which don’t provide OpenSearch output support today but have public web APIs.
* Proprietary enterprise data stores or indexes and legacy content management stores for which it might not be possible to implement a front end.

Below are options that could help work around these cases.

## Middle-man Web Service

If you don’t have access to modifying the existing data source’s web service or the web service provides a custom API, it may be possible to write a middle-man web service that can take the Windows 7 query, and then (1) connect to your data source and and retrieve the results, (2) reformat them in RSS or Atom formats, and (3) return them to the Windows 7 client.

For enterprise data services and many internet data services, you will likely need to pass the user credentials through for the web service to do the appropriate results trimming based on that user’s permissions.

## Using an Existing Search Engine

For public content, you can also use a public search engine that already supports OpenSearch with RSS, like live.com or yahoo.com. You can do this by providing your users an .osdx file that has a URL template that restricts results to only those for your specific domain.

For example, here is a sample OpenSearch description for searching just the help content for Windows using a query against live.com:

<?xml version="1.0" encoding="UTF-8"?>

<OpenSearchDescription xmlns="http://a9.com/-/spec/opensearch/1.1/">

<ShortName>Windows Help</ShortName>

<Description>Search Windows Help using the live.com search engine</Description>

<Language></Language>

<Url type="text/html" template="http://windowshelp.microsoft.com/windows/en-us/search.aspx?=&amp;qu={searchTerms}"/>

<Url type="application/rss+xml" template="http://api.search.live.com/rss.aspx?source=web&amp;query={searchTerms} site:windowshelp.microsoft.com&amp;web.count=50"/>

</OpenSearchDescription>

For content inside the enterprise, you can use an existing indexing server that supports OpenSearch such as SharePoint Search Server to index your content. You can then create a .osdx file that restricts the results from the SharePoint index to just those from your server by using their KeyWord syntax within the URL template: Site:"<http://myserver/>".

## Client-side Data Source

For those cases where a server-side only solution will not work, it is possible to write a client-side OpenSearch data source that sits in between the Windows OpenSearch provider and the external data source.

The [IOpenSearchSource API](http://msdn.microsoft.com/en-us/library/dd378289(VS.85).aspx) is documented in the Windows SDK for Windows.

With an appropriately configured .searchconnector-ms (see Windows SDK for details on this file type), Windows Explorer calls your implementation with the query parameters. Your implementation can then return results formatted in RSS or Atom format. That allows your implementation to provide custom authentication UI and connect to the data source using its proprietary API.

# Best Practices

This section lists tips and best practices for working with Windows 7 and OpenSearch.

* Support the {startIndex} and {count} parameters, and be sure to always return the number of items requested unless you are returning the last of the results.
* Using file extensions is a better way to identify a file type than using a MIME type. If you know the file extension, map it to the System.FileExtension Windows Shell property.
* Make sure that the MIME type or file extension you specify in the RSS matches the filename and MIME type returned in the HTTP header by the web server that hosts the item when the item content is requested.
* If you’re returning file items, return a file size whenever possible. This ensures that the download progress dialog is accurate.
* Verify that requests for items beyond the end of the results set return no results. DO NOT repeat results!
* Don’t put HTML tags where they don’t belong. Per the RSS specification, they are valid in the <description> field, but not in the <title> field.
* Don’t create enclosures for web page items. For example, if you create an enclosure and map a file extension of .aspx, you would receive undesired functionality. The file would be downloaded by Windows Explorer to the internet cache and executed from there. In the case of a .aspx file, web browsers do not handle this file type. The user would get an Open With dialog, or the file may be opened by an application like Visual Studio. Avoid this by only returning a <link> element for web pages.
* Provide a web roll-over URL in the .osdx file using a URL template with type=“text\html”.
* Provide a URL to the parent folder, container, or web page by mapping a custom element URL value to the “System.ItemFolderPathDisplay” Windows Shell property.

# Conclusion

By following the three steps described in this document, you should be able to enable your web data source to support Windows federated search and know how to deploy search connectors to get your users using it from within Windows.

# For More Information

To see a demonstration of building an OpenSearch web service for a SQL database. View the “[Windows 7: Empower users to find, visualize and organize their data with Libraries and the Explorer](http://channel9.msdn.com/pdc2008/PC16/)” presentation given at the PDC in October 2008.

The [Windows 7 beta SDK](http://www.microsoft.com/downloads/details.aspx?FamilyID=a91dc12a-fc94-4027-b67e-46bab7c5226c&DisplayLang=en) also contains a sample of how to build an OpenSearch source similar to the demo shown at the PDC.

For information on Windows System properties, see “[System-Defined Properties for Custom File Formats](http://msdn.microsoft.com/en-us/library/dd368864(VS.85).aspx)” page on MSDN.

For information about deploying search connectors in the enterprise read the deployment white paper at <http://go.microsoft.com/fwlink/?linkid=99999>.

For information about the OpenSearch standard, see their Web site (<http://www.opensearch.org>).

SharePoint Search Server 2008 and MOSS 2007 SP2 also support federated search of remote servers using OpenSearch. For more information read the MSDN documentation (<http://msdn.microsoft.com/en-us/library/cc789675.aspx>).

Web addresses can change, so you might be unable to connect to the Web site or sites mentioned in this paper.