Web Based iViews- Introduction

This article discusses the general concept of iViews that present data from an HTML data source - i.e. a web application/web page.

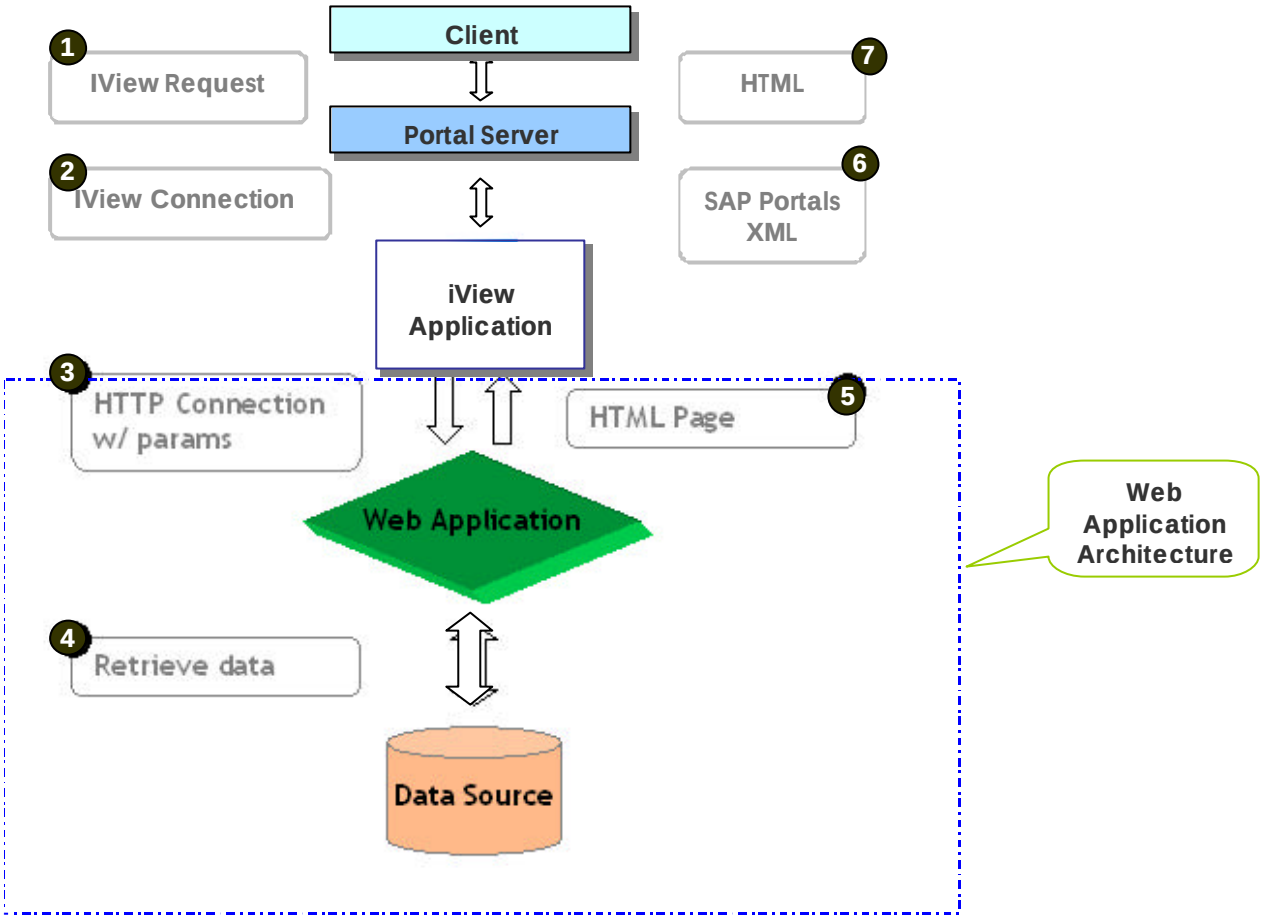
A web application is any application that users access through a browser. They can be In**ter**net or In**tra**net applications - it doesn't really matter.
Most web applications send their data to the client (user's browser) as **HTML** pages.

Web applications can be written in a variety of formats and programming languages.
They can be scripts, such as ASP or PERL.
They can also be compiled code in the format of an executable app, or a dll.
It does not matter as long as the output of the web application is in HTML format.
Many times, especially when working with **Internet** apps, some parts of the source code will not be transparent.
What you will do is analyze the web application "remotely", without looking into its code.

In the following sections we will see how to parse the HTML data from the web and convert the HTML data into SAP Portals XML through the following examples.

- HTML as Data Source- A Simple example
- Web application using GET method- NASDAQ
- Web application using POST method- Amazon

Architecture



The Flow:

1. When the user accesses the portal, a request is sent for the iView to the Portal Server.
2. The Portal Server connects to the iView Application (ASP file) and sends the parameters as defined by the iView. The HTTP request can be GET or POST method.
3. The iView application sends an HTTP request (***Request method can be GET or POST- depends on the Web Application***) to the Web Application.
4. The Web application serves as the data source of the iView application.
5. The Web application retrieves data from its data source and sends back the data as an HTML page.
6. The iView application translates the HTML page to SAP Portals XML.
7. The Portal Server translates the SAP Portals XML to HTML and serves the page to the client.

Tools Available

The following tools can be used to retrieve data from the Web-

- iView Catcher
- URL Helper
- MSXML3.0- ServerXMLHTTP object

iView Catcher

The iView Catcher is an Enterprise Portal tool that helps you create iViews. Using the iView Catcher, you can select any area embedded in a Web page or Web-based application, and then transform it into an iView to be displayed in the Enterprise Portal. With the iView Catcher's intuitive and user friendly interface, you can create dozens of completely usable iViews in a matter of minutes.

For detailed information on the iView Catcher, please refer Enterprise Portal Guide.

URL Helper

URL Helper is SAP Portals proprietary technology, built for the Enterprise Portal and used for retrieving content from a web resource.

URL Helper is a COM interface for several dynamic link libraries and consists of the following object:

- URLHelperMod.dll

The URL Helper object, which is by default located in **%SystemRoot%\SAPPortalsComponents** e.g C:\WINNT\SAPPortalsComponents must be registered using regsvr32.exe

You can use URL Helper to retrieve content from a web resource by calling the functions of the COM interface from the ASP that you use to render the iView.

For detailed information on methods and properties of the URL Helper Object and examples of usage, please refer the Enterprise Portal Guide.

MSXML3.0- ServerXMLHTTP

MSXML3.0 is the Microsoft XML parser and with it, Microsoft has provided a server-safe tool called the **ServerXMLHTTP** object. ServerXMLHTTP is an http GET/POST component with some major added advantages. ServerXMLHTTP provides methods and properties for server-safe HTTP access between different Web servers.

Using this object, you could grab weather information from a public weather site based on a specific zip code, strip out a small portion of it (eliminating all the ads and other extraneous content) and then display this in your own page.

In all the examples related to this article, we will be using the ServerXMLHTTP object. We will learn about the object methods and properties that we commonly use, through the examples.

MSXML3.0 is installed with the Enterprise Portal. You do not have to install it. You can run the *checkregistry* utility to find out the version of MSXML on your machine. Please check for the *checkregistry* utility in the following location-

<Drive>:<path to Enterprise Portal Installation folder>/Utilities

Note- *Functionality to retrieve content from HTTPS web sites is provided in the service pack 1 release. Please refer to the following article from Microsoft.*

Proxy Environments

For Web Based iViews, it is very important to consider proxy related issues. In proxy environments, we recommend using the MSXML3- ServerXMLHTTP object rather than the URL Helper Object.

Microsoft has a proxy configuration utility-*proxycfg.exe*, which must be configured for the ServerXMLHTTP object to work. In the Enterprise Portal Version 5.0, the proxy configuration utility is automatically installed and configured.

If you are using Enterprise Portal 5.0, please check for the Proxycfg.exe tool in the following location-

<Drive>:<path to Enterprise Portal Installation folder>/Utilities

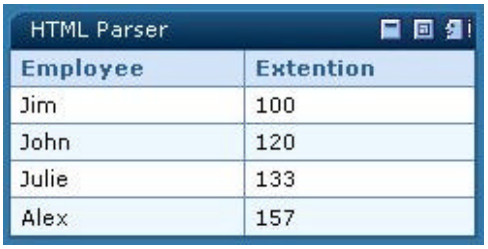
If you want to reconfigure the proxycfg.exe tool, or if you are using older versions of the Enterprise Portal please refer to MSXML3 and Proxy in this document.

For proxy environments with access lists please refer to Proxy Environment with Access Lists in this document.

HTML as a Data Source- A Simple example

We will examine an ASP that connects to a very simple web page, retrieves data and displays the data in SAP Portals XML format.

- Install the iView from the zip file.
- Place the htmlsource.htm under your wwwroot directory.
- Browse to the htmlsource.htm in your Internet browser by accessing the following URL- http://your machine name/htmlsource.htm
- **Parameter** for the iView: The **value** is the URL to the HTML page- htmlsource.htm
- Test the iView - it should appear like the screen shot below



ASP Code:

The ASP code does the following

- Connects to the URL and pulls the HTML document.
  This is done using a proprietary Microsoft object called **MSXML3** (see below)
- Builds a simple SAP Portals XML: HRRow (matrix) with two columns
- Parses the HTML source code for the values within the table cells.
  The values are then 'injected' into the SAP Portals XML

Connect to the web site and retrieve data

set HTMLData = Server.CreateObject("MSXML2.ServerXMLHTTP")	Instantiates a MSXML3 object. It enables the retrieval of resources from the Web. For full information about this object please visit the link http://www.perfectxml.com/msxml.asp Note: Though we are using MSXML3, while instantiating the object we need to use MSXML2.ServerXMLHTTP
HTMLData.open "GET",url	The ***Open*** method initializes the connection to the URL. The URL is read as a GET parameter.
HTMLData.Send	Send method send the request
HTMLData = HTMLData.ResponseText	***ResponseText*** contains the HTML code of the web page

Loop on the data and parse it

The data in the HTML is contained in the <TD> elements. The code simply parses the text inside them.
Here is an example of the HTML source. The values are marked **bold**:

```
<tr>
<td width="50%">Jim
<td width="50%">100
</tr>
```

position1 = instr(1, HTMLData, "<TD", 1)	***instr*** searches for a string within a string. It searches for the first "<TD" string.
while position1 <>0	Starts a loop that ends when there are no more "<TD" strings
position1 = instr(position1, HTMLData, ">", 1)+1	Searches for the ">" following the "<TD". This is where the value **begins**. Note the first argument of ***instr****: position1*. This argument marks where the search begins. In this case, it begins where the last search ended.
position2 = instr(position1, HTMLData, "<", 1)	Searches for the **end** of the value
Response.Write Mid(HTMLData, position1, position2 - position1)	Writes the value to the client. Uses the ***mid*** function to return part of the HTMLData string.

Web Application using GET method –NASDAQ

In this example, we will create a Web Application iView on the NASDAQ web site. The NASDAQ website uses the GET method.

We will discuss the following:

- Analyzing the NASDAQ Website
- How to handle the parsing of a large HTML source
- Solving problems related to the fact that the HTML is not well-formed (as SAP Portals XML should be)

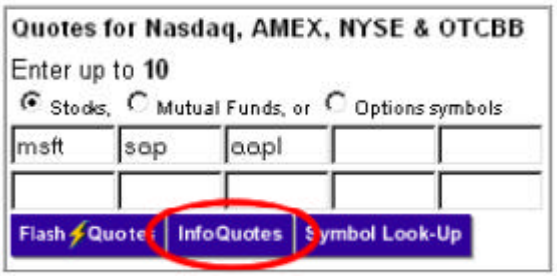
- How to efficiently parse the HTML so that we will be able to reuse the fonts and color settings of the original HTML inside SAP Portals XML

The NASDAQ Search Site

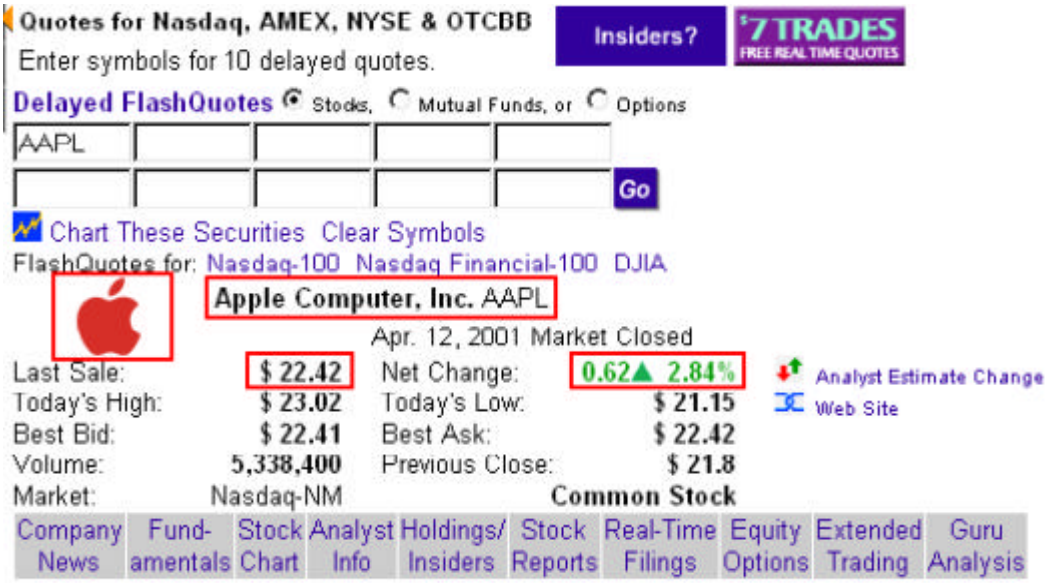
Let's take a look at the NASDAQ search application, and what parts of it we wish to display in our iView.

Open a browser window and go to www.nasdaq.com

Note the area shown below, for searching using stock symbols.
Enter some stock names and click **InfoQuotes**.



The site displays the following information on each stock:



The information we want in the iView

This is the information we wish to display in the iView.
It is highlighted on the screen shot above:

1. Company logo
2. Company name and ticker symbol
3. Last sale
4. Net change

HTTP Request Format

The request to the NASDAQ search engine is sent using the **GET** method.
You can look at the URL of the **result** page:

http://quotes.nasdaq.com/quote.dll?page=multi&mode=stock&symbol=aapl&symbol=&symbol=&symbol=&symbol=&symbol=&symbol=&symbol=&symbol=&symbol=&symbol=

You can see all the stock parameters are named '**symbol=**'.

Alternatively, you can examine the source code of the search form for the request format. View the source of the page- www.nasdaq.com
Search for '<form'. The form method is GET.

The iView and ASP Code

- Install the iview from the Zip file
- Parameter to be personalized – symbol1, symbol2…..

We'll now go over the code of the iView application. We suggest you take a look at the following site to get a full picture of what the parsed source looks like. We will quote the important parts of it over here.

http://quotes.nasdaq.com/Quote.dll?mode=stock&symbol=csco&symbol=aapl&symbol=&symbol=&symbol=&symbol=&symbol=&symbol=&symbol=&symbol=&multi.x=25&multi.y=6

SAP Portals XML Constructor

`'==========================`
`' SAP Portals XML Constructor`
`'==========================`
	This code section defines the constructor for the SAP Portals XML structure.
	It's a standard matrix with 3 columns:
	1. field1: Labeled Stock, holds the stock image and name
	2. field2: Last Sale
	3. field3: Net Change

Connect to the site and pull the HTML data

'========== ' Connect to the site and pull the HTML data '==========

set HTMLData = Server.CreateObject("MSXML2.ServerXMLHTTP")	Instantiates a MSXML3 object. It enables the retrieval of resources from the Web and facilitates the parsing of any XML content returned from the Web. For full information about this object please visit the link http://www.perfectxml.com/msxml.asp Note: Though we are using MSXML3, while instantiating the object we need to use MSXML2.ServerXMLHTTP
'URL beginning targetURL = "http://quotes.nasdaq.com/quote.dll?mode=stock&"	This is the beginning of the URL that is to be sent to the Web Application. This part of the URL doesn't change.
'Loop over all the symbol parameters. 'If they are spaces (meaning empty parameters) - do not include them in the URL because this causes problems for i = 1 to 10 targetURL = targetURL & "symbol=" if Request.QueryString("symbol"&i) <> " " then targetURL = targetURL & Request.QueryString("symbol"&i) targetURL = targetURL & "&" next	If the user will leave one of the ten stock parameters empty, the Portal sends a space character rather than an empty value. This causes a problem with the site we are working against here. This simple loop code checks the parameters sent from the Portal. Each one is named 'symbolX' (X between 1 and 10). First we concatenate the string "symbol=" to the URL. If the current param isn't a space we concatenate it's value as well.
'URL end targetURL = targetURL & "multi.x=0&multi.y=0"	This is the end of the URL that is to be sent to the Web Application. This part of the URL doesn't change.
HTMLData.open "GET",targetURL HTMLData.send HTMLData = HTMLData.ResponseText 'The contents of the site is now placed into the variable HTMLData	Sends the request to the site, and places the data into the local variable **HTMLData**.
'Translate escape characters HTMLData = Replace(HTMLData, " ", " ") 'Translate space characters ( ) into spaces (" ") HTMLData = Replace(HTMLData, "&", "&") 'Translate & to &amp	We now need to translate (decode/encode) certain escape characters that aren't legal in XML. Those are mostly the ones that involve the **&** character.

	The first code line translates spaces ( ) into 'real' spaces. The second encodes the rest of the & symbols.

Loop on the data and parse it

This section holds all the parsing code. Let's take a general look at it first.

' ========== 'Loop on the data and parse it ' ========== '"Last Sale:" is our "anchor" string - we know each "record" contains it position1 = InStr(1, HTMLData, "Last Sale:", 1) while position1 <>0	The main parsing loop needs a main "anchor". By anchor we mean some unique text in the HTML source that identifies each row of data. We chose the text **Last Sale**. So we start the loop by looking for that string in the source.
%> <tuple> <old> <dataset_1> ...	Then we start writing the SAP Portals XML tuple structure. We will go into each field in detail later. At the mo ment, please scroll down in the code to find this line...
... position1 = InStr(position1, HTMLData, "Last Sale:", 1) %>	After writing all the fields, we look for the next "anchor" line.
</dataset_1> </old> </tuple> <% wend %> </dataset> </busdoc>	Close the <tuple> structure, close the loop, and close the SAP Portals XML.

Now that we've seen the large structure, let's look inside the loop and see how each field is parsed.
For each field, we will first show you the **HTML** source that we want to parse, and then the **ASP** code that parses it.

field1: Stock icon

Stock Icon HTML Source

The code for the image is located **above** our "Last Sale:" anchor.

```
<img src="http://a676.g.akamaitech.net/f/676/838/1h/
nasdaq.com/logos/msft.gif" border=0 align="absmiddle" height="40">
```

Please note this code is **not well-formed**! This is a problem our ASP will have to deal with. To make it well formed we need:

1. Quotes around the attributes (`border="0"` and not `border=0`)
2. Slash before the closing bracket (`<img.....>`)

Also note:

- The code in green is well formed. It also changes because it contains the image URL.
- The code in red is not well formed. However, it is the same for each image as it just contains the formatting instructions.

What we'll do is parse the source for the green part, and write the static red part manually.

Stock Icon ASP Code

`'Image - stock icon`	
`position1 = InStrRev(HTMLData, "<img", position1, 1)` `'Use InStrRev to search _backwords_`	Search from the current position in the code ("Last Sale:") **backwords**, for the string "<img">, which is the beginning of the image tag. To search backwords we use `InStrRev`
`position2 = InStr(position1, HTMLData, """", 1)+1`	Look for the first double quotes which begin the ***src*** attribute. Note that to search for the string " we need to write it as "" (VB convention).
`position2 = InStr(position2, HTMLData, """", 1)`	Look for the next set of double quotes which end the ***src*** attribute.
`Response.Write Mid(HTMLData, position1, position2 - position1)` `'Start of the IMG tag`	Write the contents from the beginning of the IMG element to the end of the src attribute.
`'End the IMG tag manually because original HTML is not well-formed` `'Note the format "" to write double-quotes (") when you already`	Write the rest of the IMG tag manually. Note how we write it well-formed.

inside double-quotes... 'Note we changed the height to 20 instead of 40, to make the iView smaller in size Response.Write (""" border=""0"" align=""absmiddle"" height=""20""/>")	

field1 (cont.): Stock name

Stock Name HTML Source

```
<td nowrap><font face="Arial, Helvetica, Verdana" size="2"><b>Microsoft Corporation</b> MSFT</font></td>
```

This code appears a little below the image. The actual data is the code in yellow. We'll do this by parsing the code for the element.

Stock Name ASP Code

'Stock Name	
position1 = InStr(position1, HTMLData, "<font", 1)	Search from the current position in the source for the beginning of the element (position1).
position2 = InStr(position1, HTMLData, "", 1)+7	Search for the end of the <font> element (position2).
Response.Write Mid(HTMLData, position1, position2 - position1)	Write the contents between the two positions. Note this includes the font element as well: so we get the site's formatting, and not the standard iView formatting. We could have chosen to skip the formatting tags and parse only the code inside them.

field2: Last Sale

Last Sale HTML Source

```
<td noWrap width="85">Last Sale:</td>
<td align="right" width="85"><nobr><b>$ 60.1</b></nobr></td>
```

Our anchor point is again `Last Sale:`

The text we will parse is highlighted in yellow. As you can see, we need all the code inside the `<td>` element.
Inside that, the actual value that we are interested in is marked **bold**. The rest of the yellow text are HTML formatting tags.

Last Sale ASP Code

`'=====================================`	
`'Last Sale`	
`position1 = InStr(position1, HTMLData, "Last Sale:", 1)`	Find the **anchor** - the string "Last Sale:"
`position1 = InStr(position1, HTMLData, "<td", 1)`	Find the **start tag** of the **TD** element
`position1 = InStr(position1, HTMLData, ">", 1)+1`	Find the **closing bracket** of the TD start tag (*position1*)
`position2 = InStr(position1, HTMLData, "</td", 1)`	Find the **closing tag** of the TD element (*position2*)
`Response.Write Mid(HTMLData, position1, position2 - position1)`	Write the contents between *position1* and *position2*

field3: Net Change

Net Change HTML Source

```
<td align="left" noWrap width="100">Net Change:</td>
<td align="right"><nobr><b> <font color=".009900">0.06</font>
<img src="http://a676.g.akamaitech.net/7/676/838/b8899d358480a4
/nasdaq.com/images/nc_up.gif" border=0 width=11 height=10>
```

We will search for `Net Change:` as an initial anchor.
We will parse the `<font>` element which contains the net change amount (in yellow).
After that we have an **img** element which is the up/down arrow image.
As before, the green part of the img element is well formed and we'll use it as is. The red part is not well formed and we will write it manually.

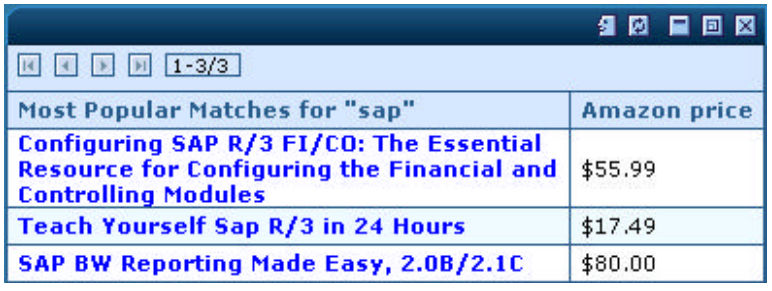
`'Net Change`	
`position1 = InStr(position2, HTMLData, "Net Change:", 1)`	Look for the "anchor' `Net Change:`
`'================================` `'Amount`	
`position1 = InStr(position1, HTMLData, "<font", 1)`	Search from the current position in the source for the beginning of the `<font>` element (position1).

`position2 = InStr(position1, HTMLData, "</font>", 1)+7`	Search for the end of the <font> element (position2).
`Response.Write Mid(HTMLData, position1, position2 - position1)`	Write the contents between the two positions. Note this includes the font element as well: so we get the site's formatting, and not the standard iView formatting. This includes the color code of green for up and red for down. We could have chosen to skip the formatting tags and parse only the code inside them.
`'Image - up/down arrow`	
`position1 = InStr(position2, HTMLData, "<img", 1)`	Search for the beginning of the image tag.
`position2 = InStr(position1, HTMLData, """", 1)+1`	Look for the first double quotes which begin the src attribute.
`position2 = InStr(position2, HTMLData, """", 1)`	Look for the next set of double quotes which end the src attribute.
`Response.Write Mid(HTMLData, position1, position2 - position1)`	Write the contents from the beginning of the IMG element to the end of the src attribute.
`'End the IMG tag manually because original HTML is not well-formed` `Response.Write (""" border=""0"" width=""11"" height=""10""/>")`	Write the rest of the IMG tag manually. Note how we write it well-formed.

Web Application using POST method – Amazon

In this example, we will create a Web Application iView on the Amazon web site. The Amazon website uses the POST method.

The iView displays the Most Popular Books for a Keyword search. The iView is rendered in SAP Portals XML.



We will discuss the following:

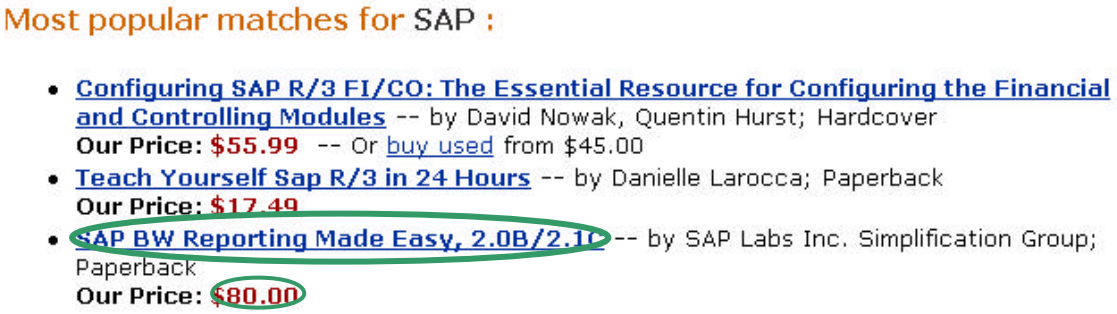
- Analyzing the Amazon Website
- Determining Parameters to be posted

- Using the ServerXMLHTTP object with the POST method
- Error Handling

The Amazon web site

Open a browser and go to the site http://www.amazon.com
Note the area below for searching Books.

Search for a book and view the results.



The information we want in the iView (Highlighted above for one Book):

- Name of the Books and the links
- Price of the Books

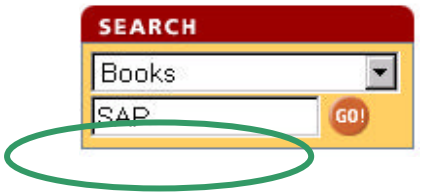
HTTP Request Format

The HTTP Request is the POST method.
You won't see the name of the book in the URL of the **result** page. This indicates that the web site uses the POST method for HTTP Request.

Alternatively, you can examine the source code of the search form for the request format. View the source of the page- www.amazon.com
Search for '<form'. The form method is POST.

Parameters
We searched on a keyword "SAP" and came up with results.
We need to find out the name of the parameter that submits the keyword.

Examine the source code of the page- www.amazon.com.
Do a search for '<input type="text'
We see that the name of the textbox in which we enter our keyword 'SAP' is '**field-keywords'**.



The iView and ASP Code

Install the iview from the Zip file

Parameter to be personalized – keyword

The ASP code does the following

- Connects to the URL, posts the parameters and retrieves data.
  We use the ServerXMLHTTP and POST method
- Builds a simple SAP Portals XML: HRRow (matrix) with two columns
- Handles errors

Connect to the web site and retrieve data

Note the targetURL = http://www.amazon.com/exec/obidos/search-handle-form/107-9696481-0308549. This is the URL of the result page we want to POST to.

set HTMLData = Server.CreateObject("MSXML2.ServerXMLHTTP")	Instantiates a MSXML3 object. It enables the retrieval of resources from the Web. For full information about this object please visit the link http://www.perfectxml.com/msxml.asp Note: Though we are using MSXML3, while instantiating the object we need to use MSXML2.ServerXMLHTTP
HTMLData.open "POST", False	The ***Open*** method initializes the connection to the URL. The URL is read as a POST parameter.
HTMLData.setRequestHeader "Content-Type", "application/x-www-form-urlencoded" HTMLData.send "field-keywords="&keyword	In the Post method the, the request is sent in two parts- Header and Body. We set the header information We then send the parameters as a name and value pair For more information on using ServerXMLHTTP with the POST method refer to the following Microsoft article.
HTMLData = HTMLData.ResponseText	***ResponseText*** contains the HTML code of the web page

Handle Errors

We have three check points for errors

- After we create an object
- After we connect to the site
- After we search for the data we want to display

We have used an error function ShowError to display errors in a neat format

Loop on the data and parse it

The data in the HTML we are interested in is contained in the <ul>.. elements and separated with <li> element. We parse the data (Name of the Book, the Link behind it and the price) inside these elements.

anchorpos = Instr(1, HTMLData, "Most popular matches for", 1) anchorpos = Instr(anchorpos, HTMLData, " ", 1) lastanchorpos= Instr(anchorpos, HTMLData, " ", 1)	We search for the section which contains our data-most popular matches ***instr*** searches for a string within a string. We then search for the first " " string. We search for the end of the section " " and then loop over data
Do while (anchorpos <> 0 AND i <= 10 AND anchorpos<lastanchorpos)	Starts a loop that ends when there are no more " " strings within …
linkstartpos= Instr(anchorpos, HTMLData, "href", 1) +6 linkendpos = Instr(linkstartpos,HTMLData, ">", 1) link=Mid(HTMLData,linkstartpos,linkendpos-linkstartpos)	Search for the "href" following the " "". This is where the link URL value begins. If you examine the link URL, you will see that we need to append "www.amazon.com" for a full path name
endpos= Instr(linkendpos, HTMLData, "", 1) temp=Mid(HTMLData,linkendpos,endpos-linkendpos) Response.Write(""<a target=""_new"" href=""http://www.amazon.com/"&link&""""") Response.Write(temp) Response.Write("")	We then parse the book name and put an "A Href" link around it
pricestartpos=Instr(endpos, HTMLData, "$", 1) priceendpos=Instr(pricestartpos, HTMLData, "", 1) temp=Mid(HTMLData, pricestartpos, priceendpos-pricestartpos) response.write(temp)	We parse the price offered by Amazon

Note:

Websites with login forms mostly follow the POST method.

MSXML3 and Proxy

It is necessary to run Proxycfg.exe, in intranets that use proxy servers to connect to the Internet or to other servers.

When you use MSXML3 -**ServerXMLHTTP** object and do not run **Proxycfg** or do not restart IIS after the installation, you receive an error.

You can download **Proxycfg.exe** from following MSDN Web site at:

> http://msdn.microsoft.com/code/sample.asp?url=/msdn-files/027/001/468/msdncompositedoc.xml

When you download and unzip the files, you find two files, Proxycfg.exe and Readme.txt. The Readme.txt file contains the information you need to configure Proxycfg.exe.

Steps to Install and Configure Proxy Config

1. Determine the proxy server name (if any) which you use.
2. Download the **Proxy Config Utility** , and then save it on your computer.
3. Double-click the executable, and select the place you want the files to be unzipped.
4. On the **Command** prompt, navigate to the folder in which the files are located.
5. At the folder, execute the **proxycfg** program with the settings you want. A list of commands can be found in the ReadMe.Txt-file.
6. A recommended method is to assign the Proxy settings of the IE browser if the proxy server is set in the IE browser—tools—internet options---Connections---- LAN settings.

   In the command prompt, you would type >Proxycfg –u and hit the return key. You will see that the Proxycfg has been configured with the proxy settings in your IE browser.
7. Stop and restart Microsoft Internet Information Server (IIS).
8. If you want to clear the proxy configuration when you move out of a proxy environment, you will need to type in the command prompt

   >proxycfg –d
9. Do not forget to Stop and restart Microsoft Internet Information Server (IIS) after running proxycfg.

Proxy Environment with Access Lists

For Proxy environments with access lists, the following steps have to be followed to run the iView successfully in the portal

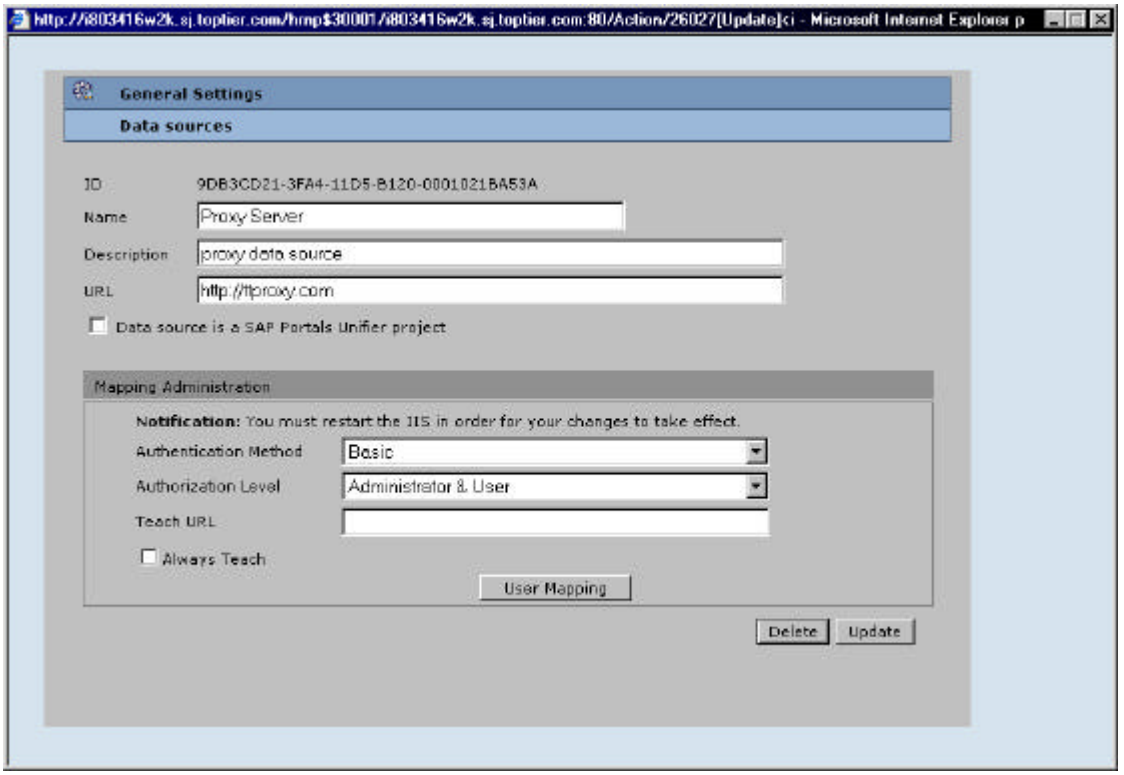
- Use the Data source-Proxy Server, and configure it
- Assign the Data Source to the iView
- Change Directory security settings in IIS

Configure the Data Source

Authentication method= Basic
Authorization level= Administrator and user.

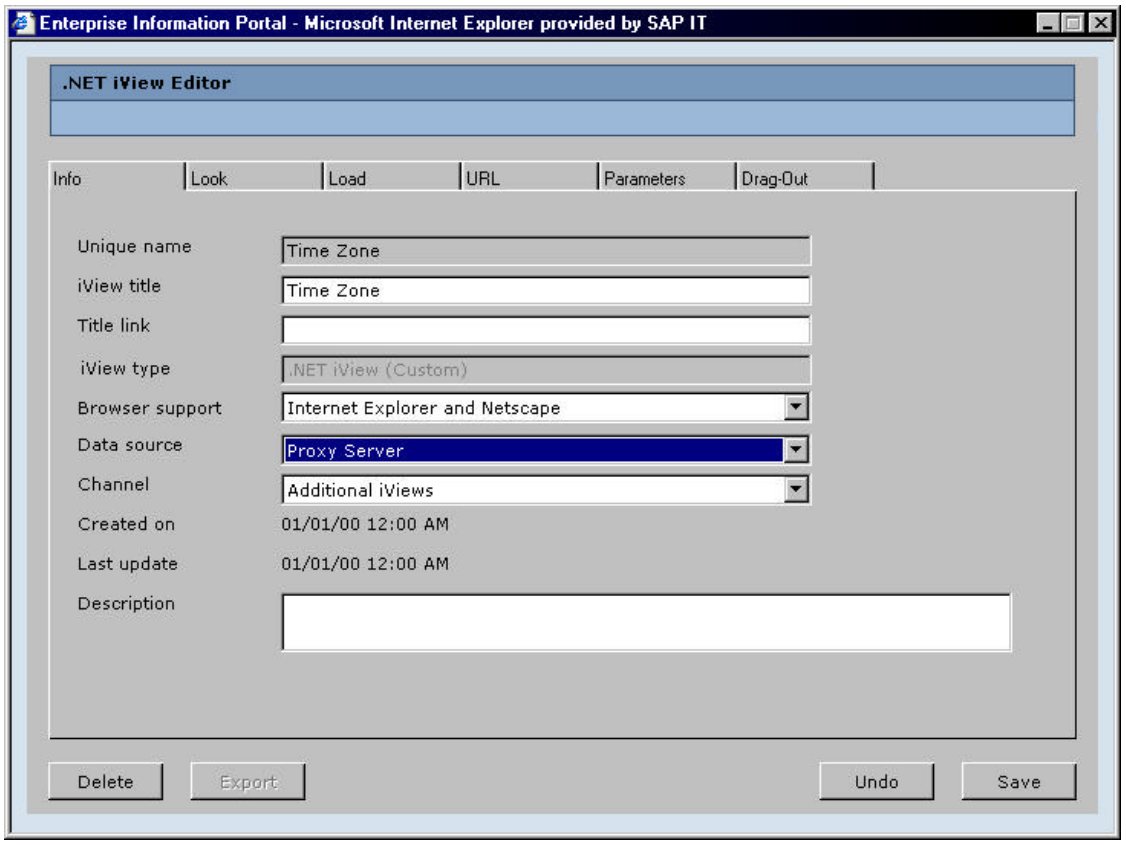
Note

If you want the end user to personalize the authentication information for the proxy server, choose Administrator and User else choose administrator only.



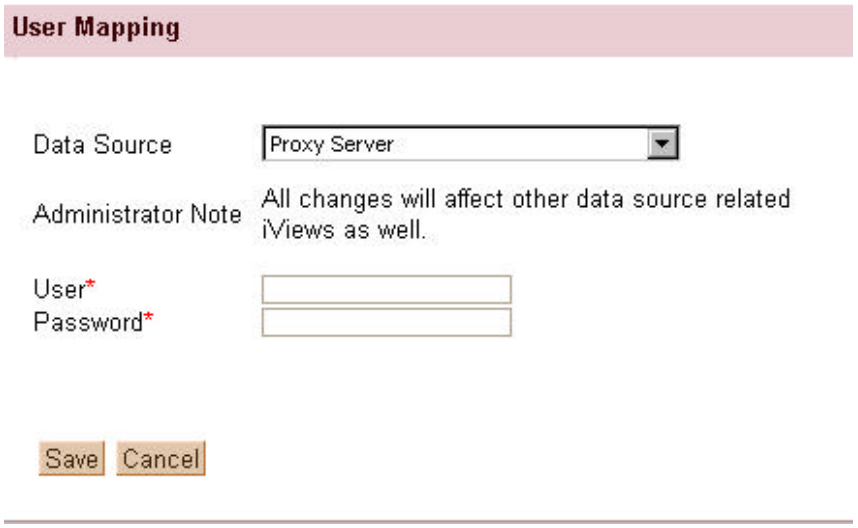
Assign the Data Source to the iView

In Info Tab of the iview editor, choose the Data Source – Proxy Server.



Note
If the iView is using another Data Source, change it to the Proxy server Data Source and pass the authentication information as required by the original Data Source as parameters.

End User can enter his mapping information.



Directory security settings in IIS

Go to IIS, and click on the properties of the **folder** containing the iview ASP file
Click on directory security.
Check on "basic" only.