URL Rewriting in AxCMS.net

URL Rewriting is used for providing more friendly looking and smarter URLs for pages and documents. It improves user-friendliness and supports search engine optimization.

URL Rewriting with AxCMS.net 9

In AxCMS.net 9 **URL rewriting** functionality is built-in with new **AxCMSHttpModule**.

AxCMS.net provides interface I**URLResolver**.

```
public interface IUrlResolver
{
    bool TrySetCurrentElement(HttpRequest request);
    List<string> GetUrls(IProtected element);
}
```

Method TrySetCurrentElement should analyze the provided HttpRequest and, if request matches **UrlResolver**'s URL format, set AxContext.CurrentElement to page or document. UrlResolver may optionally also set AxContext.CurrentActiveID for current navigation node and AxContext.CurrentChannelAxID for current channel. If these properties are not set, **AxCMSHttpModule** will try to detect them in the usual way. Method should return true, if URL matches the UrlResolver's URL format (also if there are no elements behind the URL - to display proper 404 page) or false, if URL is not handled by resolver.

Method GetUrls should return all URLs that can be used for addressing the provided element. These URLs are used when e.g. setting link in label - they are provided to editor, so editor can choose the actual URL for the link.

Both MS and LS web.config should define the list and the order of the resolver classes, e.g.:

```
<configSections>
 <section name="UrlResolvers" type="Axinom.AECMS.HttpHandler.UrlResolverConfiguration, AxCMS.BL" />
</configSections>
<UrlResolvers>
 <Resolver class="Axinom.AECMS.HttpHandler.AliasUrlResolver, AxCMS.BL" mode="on" />
 <Resolver class="Axinom.AECMS.HttpHandler.NavigationUrlResolver, AxCMS.BL" mode="on" />
</UrlResolvers>
```

AxCMS.net creates static instances of the resolvers on application initialization and for each request **AxCMSHttpModule** asks every resolver in the defined order to try to set current element. If resolver does not assign element, **AxCMSHttpModule** will try the next one. If one of the resolvers returns true (meaning that URL provided should be handled by AxCMS.net), but does not set CurrentElement, AxCMS.net will show error page.

AxCMS.net provides several **resolvers**:

- **DefaultUrlResolver** – matching aspx pages to their .AxCMS urls and assigning ActiveID and Channel. DefaultResolver is not defined in web.config and is always the last resolver **AxCMSHttpModule** tries. If resolving with default resolver is unsuccessful, module will return processing to IIS.
- **NoExtensionUrlResolver** – matching AxCMS page with URL without extension. E.g. /ForumModule returning page ForumModule
- **NavigationUrlResolver** – matching pages and documents with URLs made from their navigation path. E.g. /Solutions/Modules/ returning default page for navigation node /Solutions/Modules. /Solution/Modules/ForumModule returning page ForumModule backward classified in the node /Solutions/Modules. NavigationUrlResolver requires properly defined navigation with AxSite hosts.
- **AliasUrlResolver** – matching pages and documents with URL formed from Details.AdditionalString parameter. E.g. /offer2009 returning page having Details.AdditionalString=“offer2009”.

For any resolver which does not use extension, wildcard mapping have to be set in IIS5/6/7 (classic mode), so that requests are handled by ASP.NET. In IIS7 integrated mode you have to set IIS to invoke **AxCMSHttpModule** for non-ASP.NET requests.

Sample for NavigationUrlResolver

The following sample demonstrates how to modify a template file to support NavigationUrlResolver.

1. Activate the resolver in your MS and LS configuration files

   <Resolver class="Axinom.AECMS.HttpHandler.NavigationUrlResolver, AxCMS.BL" mode="on" />

2. Modify PremiumSampleContext.cs file by commenting out the old URL builder and adding the modified one

   using System.Linq;

   //url = CMSConfigurationSettings.VirtualPath + GetURLForNavigationID(navID) + "?ActiveID=" + navID.ToString();

   var categories = Registry.GetCategoryTreeManager().Tree.MakePathToRoot(node.AxID).Cast<AxCategory>().Reverse();
   var names = categories.Skip(2).Select(o => o.Name);
   var path = String.Join("/", names.ToArray());
   url = CMSConfigurationSettings.VirtualPath + "/" + path;

3. Build the solution

Make sure that you have entered site host under site navigation properties

Sample for NoExtensionUrlResolver

Using that URL rewrite method will allow you to build up URLs using a page name in URL.

1. Activate the resolver in your MS and LS configuration files

   <Resolver class="Axinom.AECMS.HttpHandler. NoExtensionUrlResolver, AxCMS.BL" mode="on" />

2. Modify PremiumSampleContext.cs file by commenting out the old URL builder and adding the modified one

   //url = CMSConfigurationSettings.VirtualPath + GetURLForNavigationID(navID) + "?ActiveID=" + navID.ToString();url = CMSConfigurationSettings.VirtualPath + GetURLForNavigationID(navID).Replace(".AxCMS", ""); //Modify the replace value if you are using custom extension

3. Build the solution

Page name shouldn't include white spaces.

URL Rewriting before AxCMS.net 9

Before AxCMS.net 9 there were several ways to **implement URL rewriting**.

Changing page encoding format

This is not actual URL rewriting, but can still be used for SEO etc.

You can extend AxPage in your AxCMS.net project and rewrite the names of pages on the fly to allow for better search engine indexation (For example, replace underscore separated page names - `**page_name**` with hyphen separated style names - `**page-name**`).

For more information about AxCMS.net name rewrite capability, please check Extending AxPage document.

Problem of this approach is that most of your urls will still be in format http://sitename/pagename.extension?ActiveID=1.

Using IIS7 URL rewriting or custom ISAPI or HttpModule

There are quite a lot third-party modules available and also URL rewriting module exists for IIS7. While configuring such modules, you have to define rewriting rules. For example, to use URL format http://sitename/language/pagename for pages in format language_pagename, you have to define following rules:

1. Redirect rule, using the regular expression:

   From **([_0-9a-z-]+)_(en|de).AxCMS([?=_0-9a-z-]+|())** to **{R:2}/{R:1}{R:3}**

   This redirects requests from site navigation to formatted links, e.g.:

   Solutions_en.AxCMS?ActiveID=1003 to **en/Solutions?ActiveID=1003**

   Products_de.AxCMS to **de/Products**
2. Redirect helper rule for templates folders. It assigns the right location for these folders. Without that rule templates folder's location is changed to wrong ('en/templates' instead of 'templates').:

   From **(en|de)/(templates)([./_0-9a-z-]+)** to **{R:2}{R:3}**

   For example:

   en/templates/images/print.gif to **templates/images/print.gif**
3. Rewrite rule. Main rewrite rule to finish the response:

   From **(en|de)/([0-9a-z-]+)([?=0-9a-z-]+|())** to **{R:2}_{R:1}.AxCMS{R:3}**

   This matches rewritten URLs to real pages, e.g.:

   en/Solutions?ActiveID=1003 to **Solutions_en.AxCMS?ActiveID=1003**

   de/Products to **Products_de.AxCMS**

Major problem with this approach is that you cannot use AxCMS.net data (e.g. navigation) to rewrite URLs, so rewriting to http://sitename/navi1/navi2/ is quite complex.

Writing own HttpModule for URL rewriting

You can create your own url rewriting http module which will allow rewriting URL's in for example `**http://myAxCMSsite.com/catalog/item/11**` style.

We will cover rewriting URL's using custom HttpModule approach. To start with, you will need to create a class in your AxCMS.net solution's **YourProject.BL** project which inherits from **IHttpModule**. For this class to be used in URL processing process, you need to register it's name in **<httpModules>** section of your **LS.Web.config**

Assuming that the class is named MyRedirectModule:

```
<httpModules>
   <add name="MyRedirectModule" type="YourProject.BL.MyRedirectModule />
</httpModules>
```

You will need to implement

public void Init(HttpApplication application)

{}

and

Dispose()

{}

methods for the **IHttpModule** interface. In the **Init()**,

you should hook up your rewrite event to the **application** 's BeginRequest event:

application.BeginRequest += new EventHandler(MyRewrite_BeginRequest);

In the **context_BeginRequest()** method you should handle the logic for rewriting URL's, which is basically such:

Create a helper class to map virtual URL's you want the user to see to their actual counterparts located on your server, which returns a Dictionary<string, string>, where two strings are - virtual and real paths.

Then, you can use this data to check if there is page corresponding to the path requested by user and redirect the request accordingly:

```
private void MyRewrite_BeginRequest(object sender, EventArgs e)
{
  HttpApplication app = sender as HttpApplication;
  string requestedPath = context.Request.Path;
  string realPath;
  Dictionary<string, string> mappings = YourHelperClass.GetMappingsDictionary();

  //
  // You can implement some additional logic here, like regex checking
  // for ?id=11 like strings and mapping proper redirects for them

  if(mappings.Contains(requestedPath))
  {
    realPath = mappings[requestedPath];
    app.Context.Rewrite(realPath, false);
  }

}
```

If you like to know more about **URL Rewriting**:

- Refer to URL Rewriting related forum entries>>
- Or look up the MSDN article on the subject>>.

Problem with this approach is that while you can make your controls to display rewritten links in navigation etc, editor will still add links in old format.