UrlRewritingNet.UrlRewrite

Documentation

Albert Weinert & Thomas Bandt

2.0

Table of contents

What is UrlRewritingNet.UrlRewrite? ... 4
- Why rewriting my Urls? ... 4
- Functions of UrlRewritingNet.UrlRewrite ... 4
- Limitations of UrlRewritingNet.UrlRewrite ... 4

Notable things by developing a web application ... 6
- Make your rewrite rules so specific as possible ... 6
- Use the root operator „~“ ... 6
- Working with Urls without file extension ... 6
- Differences between IIS and Visual Studio WebDev Server ... 6

Installation of UrlRewritingNet.UrlRewrite ... 7
- System requirements ... 7
- Installation ... 7
  - Installation of the Assembly ... 7
  - Installation of the configuration schema ... 7
  - Setting up the configuration settings area in the Web.config ... 7
  - Embedding UrlRewritingNet as HttpModul ... 8
- Necessary changes by upgrading from 1.1 to 2.0 ... 8
  - Give your rules unique names ... 8
  - Remove „compileRegEx“ attribute ... 8
  - Refresh configuration schema ... 8

Configuration ... 9
- Attributes for <urlrewritingnet /> ... 9
- The <providers /> listing ... 10
- The <rewrites /> listing ... 10
- RegEx Rewrite Attributes ... 11

Settings on the webserver (IIS) ... 12
- Assign other file extensions with ASP.NET 2.0 ... 12
  - IIS 5.0/5.1 ... 12
  - IIS 6.0 ... 12
- Handle all requests by ASP.NET 2.0 ... 12
  - IIS 5.0/5.1 ... 12
  - IIS 6.0 ... 12

Changing of Rewrite Rules on runtime ... 13

Create your own Rewrite-Rule-Provider ... 14
- Developing the rewrite logic ... 14
- Creation of a new provider ... 16
- Embedding the provider in Web.config ... 16

Copyright notice ... 17

What is UrlRewritingNet.UrlRewrite?

UrlRewritingNet.UrlRewrite is a module which could be embedded in an ASP.NET 2.0 application to rewrite Urls (Internet addresses) for displaying the user another URL than used from the server. With UrlRewritingNet.UrlRewrite you only have to define a few simple rules for rewriting your addresses.

Why rewriting my Urls?

A little example. You're developing blog software which stores its entries in a database. To get the entry for displaying it in a details page you need the identity (ID) of the record to display. To get the ID you usually transport this value by Query String: http://myblog.com/details.aspx?id=234.

If your blog is ready and online you want to be found by potential readers on search engines like Google or Yahoo. These search engines send bots out to the World Wide Web to find interesting content. So what do you mean what the bot does with a Url like shown above? Not much, right.

So, wouldn't it be cooler if the bot could find the topic of the blog entry in the Url for example? A rewritten Url could look like this: http://myblog.com/detail/good-news-for-a-better-world-234.aspx.

The machine (search engine bot) has something to analyze and the user can imagine what the topic is about on this page, too.

Functions of UrlRewritingNet.UrlRewrite

There are some solutions for rewriting Urls with ASP.NET, but mostly there are some disadvantages, for example missing support for Themes and Master Pages. For some you need Administrator rights to install an ISAPI extension on the server.

This isn't necessary by UrlRewritingNet.UrlRewrite and you could avoid many problems.

- ♥ Rewriting Urls based on regular expressions
- ♥ Support for Themes and Master Pages
- ♥ Support for OutputCacheing
- ♥ Use in medium trust level environments (shared hosting) possible
- ♥ Consistent Url after post back
- ♥ Adding own rewrite rule providers possible
- ♥ Redirects (also permanent) to other domains or websites possible
- ♥ Support for Cookie less Sessions
- ♥ Adding rewrite rules on runtime
- ♥ Very easy installation and use

Limitations of UrlRewritingNet.UrlRewrite

So many good things have of course a shady side, too. Because of rewriting with the .NET 2.0 engine only requests coming over ASP.NET 2.0 can be handled. This means that the file extension have to be handled by ASP.NET 2.0 ISAPI library (see: Server Settings, page 14). By default this is for example .aspx – but if you want to rewrite with other extensions, you have to set this up by yourself or ask your Administrator to do this.

For this reason there is also no rewrite without extension possible (for example http://myblog.com/user/bill). For a solution for this special problem see page 6, Rewrite Urls without file extension.

We are also sorry to tell you, that „Cross Page Postings“ are not available yet without disabling security checks.

Notable things by developing a web application

For fast success you should take a look at the following basics.

Make your rewrite rules so specific as possible

If you make your rules not specific enough, you will generate not expected effects. For example you avoid access on images, style sheets, web services, java scripts or other elements on the server.

So please specify your rules as exactly as possible. For example not „^~/(.*)/(.*).aspx“. Use the specific regular expression elements for numbers if you await numbers and no generic placeholder and so on.

Use the root operator „~“

For access on resources in your web (for example images, style sheets or java scripts) always use the root operator „~“. With this operator every path will build correctly, starting from the applications root.

```
Right  <asp:Image ImageUrl=“~/images/pictures.gif runat=“server“/>
Wrong <asp:Image ImageUrl=“../../images/pictures.gif“ runat=“server“/>
```

You can also use this for HtmlControls. Just add runat=“server“ and it works.

```
<image src=“~/images/pictures.gif“ runat=“server“/>.
```

If the head tag is running on server (runat=“server“) you can leave out the runat attribute here for JavaScript and style sheet tags (and only here, not in the body of the document!).

Working with Urls without file extension

If you want to work with Urls without file extension you first have to configure the IIS (see page 14, settings on server)

Now you have defined a default Page in the UrlrewritingNet configuration section in Web.config. With that setting requests on /folder/ are redirected to /folder/default.aspx, if the attribute is set to default.aspx for example.

This is necessary because otherwise ASP.NET 2.0 couldn’t resolve the right address by calling ResolveClientUrl() method, which brings some funny effects with Themes for example with.

Important: the rewrite rule has to be configured to handle the „default.aspx“ and not the Url without extension!

Differences between IIS and Visual Studio WebDev Server

UrlRewritingNet is working with IIS (Internet Information Services) as well with the Visual Studio 2005 built in web server. But there is a little difference.

The WebDev Server passes every (!) request through ASP.NET – IIS don’t.

This is important if you want to use extensions like .html, .php, xml or other for example, or if you’re working without file extensions. The built in server does handle this „correctly“, the IIS has to be set up before getting this running on it. In our opinion the best way to test is to use the IIS also on the development machine.

Installation of UrlRewritingNet.UrlRewrite

System requirements

UrlRewritingNet.UrlRewrite is running on each Web server which is running ASP.NET 2.0.

Tested Web server with UrlRewritingNet.UrlRewrite:

- ♥ IIS 5.0
- ♥ IIS 5.1
- ♥ IIS 6.0
- ♥ Visual Studio 2005 WebDev Server

We didn't test it with Mono.

Installation

To install UrlRewritingNet.UrlRewrite you have to follow these easy steps:

1. Installation of the Assembly („.dll")
2. Installation of the configuration schema
3. Setting up the configuration settings area in the Web.config
4. Embedding UrlRewritingNet as Http Module

Installation of the Assembly

Just copy *UrlRewritingNet.UrlRewriter.dll* in your web applications /bin folder.

Installation of the configuration schema

To get IntelliSense support just copy the file urlwritingnet.xsd in you web application (wherever you want). If you web application is part of a solution, you can also put the file anywhere in the solution.

Setting up the configuration settings area in the Web.config

To get the configuration settings from the web.config this area has to be advertised. Just replace the <configSections> if exists.

```
<configuration>
  <configSections>
    <section name="urlrewritingnet"
             restartOnExternalChanges="true"
             requirePermission ="false"
             type="UrlRewritingNet.Configuration.UrlRewriteSection,
                   UrlRewritingNet.UrlRewriter"  />
  </configSections>
</configuration>
```

If the area is advertised, just create it.

```
  <urlrewritingnet
      xmlns="http://www.urlrewriting.net/schemas/config/2006/07" >
  </urlrewritingnet>
```

This section has to be placed in <configuration /> but after <configSections />. Do not place this in <appSettings /> or <system.web>! Fo example se the sample web application. To get IntelliSense support you have to add the „xmlns" attribute.

Embedding UrlRewritingNet as HttpModul

To handle all incoming requests with UrlRewritingNet you have to register the component as Http Module in the <system.web /> section in Web.config.

```
<system.web>
  <httpModules>
   <add name="UrlRewriteModule"
        type="UrlRewritingNet.Web.UrlRewriteModule, UrlRewritingNet.UrlRewriter" />
  </httpModules>
</system.web>
```

Now UrlRewritingNet configuration is complete.

Necessary changes by upgrading from 1.1 to 2.0

By upgrading from 1.1 to 2.0 you have to change a little bit on your existing configuration.

1. All rules need a unique name.
2. Don't use the attribute „compileRegEx" anymore.
3. Refresh configuration schema

If one of these requirements is not given, the application throws exceptions.

Give your rules unique names

All existing (and of course new) rules have to get unique names. This is necessary to change them on runtime.

Old rule entry:

```
<add virtualUrl="^~/girls/(.*)/(.*).aspx"
     rewriteUrlParameter="ExcludeFromClientQueryString"
     destinationUrl="~/Default.aspx?name=$1&amp;show=$2"
     ignoreCase="true" />
```

New rule entry:

```
<add name="Gallery"
     virtualUrl="^~/girls/(.*)/(.*).aspx"
     rewriteUrlParameter="ExcludeFromClientQueryString"
     destinationUrl="~/Default.aspx?name=$1&amp;show=$2"
     ignoreCase="true" />
```

Remove „compileRegEx" attribute

The attribute „compileRegEx" has become obsolete. So please remove it from all entries and the `<urlrewritingnet />` configuration area.

Refresh configuration schema

For getting IntelliSense support you have to update the namespace to the current version.

```
<urlrewritingnet
  xmlns="http://www.urlrewriting.net/schemas/config/2006/07" >
```

And of course you have to replace the existing urlrewritingnet.xsd with the new one.

Configuration

von UrlRewritingNet.UrlRewrite is configured in the area in Web.config which you set up during the installation (see page 8).

Here a little sample:

```
  <urlrewritingnet
    rewriteOnlyVirtualUrls="true"
    contextItemsPrefix="QueryString"
    defaultPage = "default.aspx"
    xmlns="http://www.urlrewriting.net/schemas/config/2006/07" >
    <providers>
      <add name="MyGreatProvider" type="MyApp.Web.MyGreatProvider,
MyGreatProvider.dll"/>
    </providers>
    <rewrites>
      <add name="Rule1"  virtualUrl="^~/(.*)/Detail(.*).aspx"
           rewriteUrlParameter="ExcludeFromClientQueryString"
           destinationUrl="~/Default.aspx?language=$1&amp;id=$2"
           ignoreCase="true" />
      <add name="Rule2"
           provider="MyGreatProvider"
           myattribute="/foo/bar/dhin.aspx"
           rewriteUrlParameter="ExcludeFromClientQueryString"
           rewrite="Domain"
           ignoreCase="true" />
    </rewrites>
  </urlrewritingnet>
```

This configuration can become very comprehensive, so you can swap it out into an external config file. Please use the configSource attribute for that:

```
  <urlrewritingnet configSource="ExternalRewrite.config" />
```

In ExternalRewrite.config you have to put the complete configuration area.

Attributes for <urlrewritingnet />

rewriteOnlyVirtualUrls

true, false (Standard: true)

Prevents rewriting for real existing files on the server if *true*.

contextItemsPrefix

String

If the Query String parameters should putted down in *HttpContext.Current.Items[],* you can define a prefix, which will be inserted before the parameters name with a point after.

defaultProvider

ProviderName (Standard: RegEx)

Name of the default Rewrite Provider used if no other is assigned.

defaultPage

Dateiname

Name of the default page which is used on access without file extension (see page 6, working without file extension).

The <providers /> listing

Here can the custom providers be added. Fore more information see page 19 „Embedding the provider in the Web.config“ and the documentation of the specific provider.

The <rewrites /> listing

Here can the rewrite rules be added. The rules are processed from up to down, the first matching rule is used for the rewriting and the process is being ready (no more rule will be processed).

To add a new rule just type a new <add /> element, IntelliSense will present you the available attributes which are listed below.

name

Name of the rule

A free selectable name of the rule which has to be unique. With this name (ID) you can do some magic with this rule programmatically, too (see page 16)

provider

ProviderName

Name of the used provider. If no one is selected, the default provider will be used.

redirect

None, Application, Domain (Standard: None)

To make a redirect instead of a rewrite.

None	Normal rewrite
Application	Redirect within your web application
Domain	Redirect to another domain which has to be part of the destinationUrl

redirectMode

Permanent, Temporary (Standard: Temporary)

Here you can select of which type your redirect is – permanent (HTTP status code 301) or temporary (HTTP status code 302). If you want to transfer a website from one domain to another permanently use „Permanent“.

rewrite

Application, Domain (Standard: Application)

Defines whether the domain has to be included in the process or not.

rewriteUrlParameter

ExcludeFromClientQueryString, StoreInContextItems , IncludeQueryStringForRewrite (Standard: ExcludeFromClientQueryString)

ExluceFromClientQueryString

In *Request.QueryString[]* all parameters are available. In *Page.ClientQueryString* are only

the parameters added, which are visible in the browsers address field, so a postback is possible. Parameters added in **destinationUrl** are not added in *Page.ClientQueryString*.

StoreInContextItems
All parameters listed in the Url field of the browser and the parameters from the rewrite are added in *HttpContext.Current.Items[]* too, prefixed by a string defined in **contextItemsPrefix**.

IncludeQueryStringForRewrite
Include the full Query String (the parameters in the browsers address field) in the rewrite process, so that they can be handled by the engine.

As a developer of a rewrite provider you should look to develop your provider meeting the standard behavior, but of course – it can vary.

RegEx Rewrite Attributes

Our standard provider has the following attributes.

virtualUrl

regular expression
A regular expression which is used for replacing the current Url with **destinationUrl**.

destinationUrl

regular expression replacement term
A regular expression term describing the target page (physical file).

regexOptions

Multiline, ExplicitCapture, Singleline, IgnorePatternWhitespace, RightToLeft, ECMAScript, CultureInvariant
For an optimal control over the regular expressions you can add additional to **ignoreCase** here some *RegexOptions*. But this is only necessary in special cases. For more information see the Microsoft MSDN documentation.

Settings on the web server (IIS)

Normally UrlRewritingNet.UrlRewrite will work without any administrative settings on IIS. But if you want some additional feature like mapping specific file extensions to handle them with ASP.NET (for example .xml or .html), you have to do something on the web server.

We recommend doing this only if you know what you are doing. Remember: never change a running system!

Assign other file extensions with ASP.NET 2.0

IIS 5.0/5.1

IIS 6.0

Open the IIS Manager tool, select the web which you want to configure and open the settings menu. Select register „Home Directory“, go to „Application settings“ and click on „Configuration“.

Here you can add additional file extensions or map existing extensions to ASP.NET under register „Mappings“ with the following settings.

Executable

c:\windows\microsoft.net\framework\v2.0.50727\aspnet_isapi.dll
(can vary by installation)

Extension

.html
(Sample for.html)

Verbs, Limit to)

GET,HEAD,POST,DEBUG

Script engine

check

Verify that file exist

check off

Handle all requests by ASP.NET 2.0

IIS 5.0/5.1

Handle all requests with placeholder *.* by aspnet_isapi.dll.

IIS 6.0

With IIS 6 you can’t handle *.* requests, but you can assign an ISAPI filter as an application placeholder (Wildcard application maps).For it you have to follow the steps described on page 14, but not adding a file extension under „Wildcard application maps“, but „c:\windows\microsoft.net\framework\v2.0.50727\aspnet_isapi.dll“.

Do not check „verify that file exists“!

Changing of Rewrite Rules on runtime

For changing the rewrite rules on runtime UrlRewritingNet.UrlRewrite offers four static methods:

```
namespace UrlRewritingNet.Web
{
    static public class UrlRewriting
    {
        public static void AddRewriteRule(string ruleName,
                                           RewriteRule rewriteRule);

        public static void RemoveRewriteRule(string ruleName);

        public static void ReplaceRewriteRule(string ruleName,
                                              RewriteRule rewriteRule);


        public static void InsertRewriteRule(string positionRuleName,
                                              string ruleName,
                                              RewriteRule rewriteRule);
    }
}
```

The parameters are very similar, ruleName is the name of the current rule and rewriteRule the current rule object.

With AddRewriteRule() new rules can be added (at the end of the list) and with RemoveRewriteRule() rules can be removed, with ReplaceRewriteRule – surprise – rules can be replaced and with InsertRewriteRule() a rule can be added on a defined position.

It has always a new instance of the RewriteRule class to be created with the wanted parameters. Is a rule changing it is available during the runtime of the application – if the application is being restarted, also all changes have to be done again.

Here a little sample:

```
RegExRewriteRule rule = new RegExRewriteRule();
rule.VirtualUrl = "^~/abteilung/(.*)/default.aspx";
rule.DestinationUrl = "~/showabteilung.aspx?abteilgun=$1";
rule.IgnoreCase = true;
rule.Rewrite = RewriteOption.Application;
rule.Redirect = RedirectOption.None;
rule.RewriteUrlParameter = RewriteUrlParameterOption.ExcludeFromClientQueryString;
UrlRewriting.AddRewriteRule("AbteilungsRegel", rule);

UrlRewriting.RemoveRewriteRule("AbteilungsRegel");
```

Create your own Rewrite-Rule-Provider

If you're limited by the way the Library usually with Regular Expressions works, it is possible to develop and integrate an own Rewrite-Rule-Provider.

The development can be divided in 3 steps:

1. Developing of the rewrite logic
2. Creation of the provider
3. Embedding the provider in Web.config

That's it. You don't have to concern about Themes, Caching and so on – this does UrlRewritingNet.UrlRewrite anymore.

As sample implementation we provider the built in RegEx Provider.

Developing the rewrite logic

You are absolutely free in implementing your own logic with an own provider. You only have to consider some little rules.

The class implementing your logic has to be based of the abstract class RewriteRule. And it has to override 3 methods: Initialize(), IsRewrite() and RewriteUrl().

If a suitable RewriteRule is added in the configuration an object of the class will be instantiated by the provider (once per runtime). You have to consider that web applications are multithreaded, so you have to synchronize your data by yourself inside of the object because possible concurrent calls of the methods. You should develop these methods as they were static methods.

```
public override void Initialize(RewriteSettings rewriteSettings);
```

One-time called by instantiating the class. Here you should read out the RewriteSettings. The class RewriteSettings offers helper methods for doing this.

```
public override bool IsRewrite(string requestUrl);
```

Here you should shortly test if the URL passes into your rewrite schema (return true;) or not (return false;), nothing more if possible.

```
public override string RewriteUrl(string Url)
```

Here you have to implement the magic which makes the physical Url from the virtual one. As return value it awaits the "real" address. Consider whether it was a domain rewrite or an application redirect or rewrite.

A sample implementation of RegExRewriteRule.

```
public class RegExRewriteRule : RewriteRule
    {
        private Regex regex;
        public override void Initialize(RewriteSettings rewriteSettings)
        {
            base.Initialize(rewriteSettings);
            this.RegexOptions = rewriteSettings.GetEnumAttribute<RegexOptions>("regexOptions",
                                RegexOptions.None);
            this.VirtualUrl = rewriteSettings.GetAttribute("virtualUrl", "");
            this.destinationUrl = rewriteSettings.GetAttribute("destinationUrl", "");
        }
        private void CreateRegEx()
        {
            UrlHelper urlHelper = new UrlHelper();
            if (IgnoreCase)
            {
                this.regex = new Regex(urlHelper.HandleRootOperator(virtualUrl),
                             RegexOptions.IgnoreCase | RegexOptions.Compiled | regexOptions);
            }
            else
            {
                this.regex = new Regex(urlHelper.HandleRootOperator(virtualUrl),
                             RegexOptions.Compiled | regexOptions);
            }
        }
        private string virtualUrl = string.Empty;
        public string VirtualUrl
        {
            get { return virtualUrl; }
            set
            {
                virtualUrl = value;
                CreateRegEx();
            }
        }
        private string destinationUrl = string.Empty;
        public string DestinationUrl
        {
            get { return destinationUrl; }
            set { destinationUrl = value; }
        }
        private RegexOptions regexOptions = RegexOptions.None;
        public RegexOptions RegexOptions
        {
            get { return regexOptions; }
            set
            {
                regexOptions = value;
                CreateRegEx();
            }
        }
        public override bool IsRewrite(string requestUrl)
        {
            return this.regex.IsMatch(requestUrl);
        }
        public override string RewriteUrl(string url)
        {
            return this.regex.Replace(url, this.destinationUrl, 1);
        }
    }
```

Creation of a new provider

The only job of the provider is to instantiate an object of your own RewriteRule class. Additional logic isn't necessary.

Your own provider has to be based on class UrlRewritingRuleProvider and has to implement only one method.

Here a complete class:

```
public class RegExUrlRewritingProvider : UrlRewritingProvider
{
    public override UrlRewritingNet.Web.RewriteRule CreateRewriteRule()
    {
        return new RegExRewriteRule();
    }
}
```

That's it!

Of course you can trigger more parameters to the provider if you need more, like in every other ASP.NET 2.0 provider by override the Initialize() method.

Embedding the provider in Web.config

For working with the new provider you have to register it in Web.config. Simply create a providers list after ASP.NET 2.0 provider standard schema.

```
<providers>
  <add name="MyProviderName" type="Classname, Assembly"/>
</providers>
```

You can add multiple providers. By giving your provider a name you can assign each rule to its provider. By default the standard provider is "RegEx" unless you add one with this name. A sample configuration:

```
<urlrewritingnet
  rewriteOnlyVirtualUrls="true"
  contextItemsPrefix="QueryString"
  defaultProvider="RegEx"
  defaultPage = "default.aspx"
  xmlns="http://www.urlrewriting.net/schemas/config/2006/07" >
  <providers>
    <add name="MyProviderName" type="MyProviderClassname, MyAssembly"/>
  </providers>
  <rewrites >
    <add name="Rule1"
         provider="RegEx"
         virtualUrl="^~/girls/(.*)/(.*).aspx"
         rewriteUrlParameter="ExcludeFromClientQueryString"
         destinationUrl="~/Default.aspx?name=$1&amp;show=$2"
         ignoreCase="true" />
    <add name="Rule2"
         provider="MyProviderName"
         rewriteUrlParameter="ExcludeFromClientQueryString"
         ignoreCase="true" />
  </rewrites>
</urlrewritingnet>
```

Copyright notice

```
/* UrlRewritingNet.UrlRewrite
 * Version 2.0
 *
 * Copyright 2006 by Albert Weinert and Thomas Bandt.
 *
 * http://der-albert.com, http://blog.thomasbandt.de
 *
 * This library is offered „as it is“, without any warranty.
 *
 * The library can be used for free in free and commercial projects.
 *
 * If this library is offered others his has to be free of charge.
 *
 * It is allowd to change the source code for own needs. By improving the
 * changed/improved code has to be published, either by sending the code to
 * us or by making it available on a own website (if so, you have to notify us).
 * This is no promise for implementing the changed code in the next release.
 *
 * This copyright notice has to be placed in the source code.
 *
 * It is not allowed to create a commercial rewrite engine based on this
 * library!
 *
 * Based on http://weblogs.asp.net/fmarguerie/archive/2004/11/18/265719.aspx
 *
 * For further informations see: http://www.urlrewriting.net/
 */
```