

UrlRewritingNet.UrlRewrite

Dokumentation

Albert Weinert & Thomas Bandt

2.0

Inhaltsverzeichnis

Was ist UrlRewritingNet.UrlRewrite	3
Warum URLs umschreiben?.....	3
Funktionen von UrlRewritingNet.UrlRewrite.....	3
Einschränkungen von UrlRewritingNet.UrlRewrite	4
Beachtenswertes bei der Entwicklung einer Webanwendung.....	5
Regeln so spezifisch wie möglich	5
Den Root-Operator „~“ verwenden.....	5
Arbeiten mit URLs ohne Dateiendung	5
Unterschiede zwischen IIS und Visual Studio WebDev Server	5
Installation von UrlRewritingNet.UrlRewrite.....	7
System-Voraussetzungen.....	7
Installation	7
Installation der Assembly.....	7
Installation des Konfigurationsschemas	7
Einrichten des Konfigurationsbereiches	7
Einbinden von UrlRewritingNet als HttpModul	8
Upgrade von 1.1 auf 2.0 (Notwendige Anpassungen).....	8
Regeln einen Namen geben	8
Attribut „compileRegEx“ entfernen.....	9
Konfigurationsschema aktualisieren.....	9
Konfiguration	10
Attribute für <urlrewritingnet />	10
Die <providers /> Auflistung	11
Die <rewrites /> Auflistung	11
RegEx Rewrite Attribute.....	12
Einstellungen am Server	13
Andere Dateiendungen ASP.NET 2.0 zuordnen.....	13
IIS 5.0/5.1	13
IIS 6.0.....	13
Alle Requests durch ASP.NET 2.0 leiten.....	13
IIS 5.0/5.1	13
IIS 6.0.....	13
Ändern von Rewrite-Regeln zu Laufzeit.....	15
Erstellung eines eigenen Rewrite-Rule Provider.....	16
Entwicklung der Rewrite-Logik	16
Erstellung des Providers.....	18
Einbinden des Providers in die Web.config	18
Copyright.....	20

Was ist UrlRewritingNet.UrlRewrite

UrlRewritingNet.UrlRewrite ist ein Modul welches in eine ASP.NET 2.0 Anwendung eingebunden werden kann um URLs (Internet Adressen) umzuschreiben. So das man in der Adressleiste etwas anderes sieht, als das womit der Server arbeitet. Mit UrlRewritingNet.UrlRewrite müssen einfach nur ein paar entsprechende Regeln (Rules) für das umschreiben definiert werden.

Warum URLs umschreiben?

Wenn man nun z.B. einen Blog entwickelt, dann werden die Einträge üblicherweise in einer Datenbank abgelegt, um nun einen Blog Eintrag zuzugreifen braucht man dazu ja die Identität (Id) des Datensatz. Um die Id nun über eine URL an den Server zu übergeben so wird dies üblicherweise im QueryString gemacht (<http://meinblog.tld/show.aspx?id=234>) damit kann ein Computer was anfangen.

Heutzutage wird von Suchmaschinen auch die URL in die Suche mit einbezogen, dazu gibt o.g. URL nicht viel her. Auch wird der QueryString meist ignoriert. Schöner wäre es doch wenn man eine sprechende URL (<http://meinblog.tld/show/tolle-neuigkeit-alles-wird-besser.aspx>) hätte. Da hat die Suchmaschine etwas auszuwerten und selbst als Mensch kann man über so eine URL eine Information über den Inhalt der Seite bekommen.

Jedoch ist es in einem dynamischen System wie in einem Blog (Online-Shop, Forum, Online-Datenbank usw.) nicht praktikabel für jeden Eintrag eine eigene Seite anzulegen. Nun kommt das umschreiben von URLs ins Spiel, damit können virtuellen Adressen auf reale Seiten auf dem Server zeigen und ohne das dies großartig mehr Aufwand in der Entwicklung der Anwendung bedeutet.

Man muss jedoch darauf achten dass man aus der URL auch wieder etwas Eindeutiges erzeugen kann mit dem die Serveranwendung auch etwas anfangen kann (z.B. <http://meinblog.tld/show.aspx?subject=tolle-neuigkeit-alles-wird-besser>). Die nun real existierende Seite show.aspx holt sich den Subject-Parameter und sucht nach einem entsprechenden Eintrag mit der Überschrift. Der Benutzer sieht weiterhin die virtuelle Adresse im Browser. Natürlich könnte auch die die Id mit in die URL eingearbeitet sein um diese dann wie gewohnt auszuwerten.

Funktionen von UrlRewritingNet.UrlRewrite

Es gibt einige Rewrite-Lösungen für ASP.NET, etliche haben davon jedoch Nachteile, zum Beispiel ist die ASP.NET 2.0 Unterstützung mangelhaft (z.B. Themes und MasterPages). Für andere braucht man einen Administratorzugriff auf den Server um z.B. eine ISAPI Erweiterung einzubinden.

Dies ist mit UrlRewritingNet.UrlRewrite nicht nötig und viele Probleme werden damit vermieden.

- ♥ Umschreiben von URLs auf Basis von Regulären Ausdrücken
- ♥ Themes und MasterPages werden unterstützt
- ♥ OutputCache wird unterstützt
- ♥ Einsatz auch im Shared-Hosting s.g. Medium Trust Umgebungen möglich
- ♥ Bei einem Postback ändert sich die URL nicht.
- ♥ Eigene Rewrite-Rule-Provider möglich
- ♥ Auch Redirects zu anderen Domains oder Seiten sind möglich
- ♥ Cookieless Session werden unterstützt
- ♥ Regeln können zur Laufzeit entfernt und hinzugefügt werden

♥ Einfach in der Installation und Anwendung

Einschränkungen von `UrlRewritingNet.UrlRewrite`

Soviel gutes hat auch ein paar Schattenseiten. Da das Umschreiben mit .NET 2.0 mittels vollzogen wird können nur URLs umgeschrieben werden die vom ASP.NET 2.0 verarbeitet werden. Dies bedeutet das die Dateiendung der mit der Adresse endet der ASP.NET 2.0 ISAPI Library zugeordnet sein muss. Siehe dazu „Einstellungen am Server“ (Seite 13), dazu braucht es jedoch Administratorzugriff auf den Server, oder jemanden der dies übernimmt. Für Standardendungen von ASP.NET (z.B. .aspx) müssen Sie jedoch nichts weiter machen. Und dies reicht in den meisten Fällen auch aus.

Auch aus diesem Grund sind nicht ohne weiteres Rewrites von Adressen ohne Dateiendung möglich (z.B. <http://meinblog.tld/user/albert>). Um dies Einschränkung zu umgehen muss man jeden URL-Request des Servers von ASP.NET 2.0 verarbeiten lassen. Siehe dazu „Arbeiten mit URLs ohne Dateiendung“ (Seite 5)

Leider funktionieren in der aktuellen Version keine *CrossPagePostings* ohne Abschaltung der Sicherheitsüberprüfung auf gültige Daten. Da der Server eine andere Ursprungsseite erwartet und den Vorgang aus Sicherheitsgründen nicht zulässt.

Beachtenswertes bei der Entwicklung einer Webanwendung

Damit die Anwendung selbst und die Browser keine Probleme mit dem umgeschriebenen URLs haben sollte man folgendes beachten.

Regeln so spezifisch wie möglich

Die Regeln sollten nicht zu allgemein definiert werden, dies hätte meist zur Folge das auch Adressen umgeschrieben werden die nicht dazu gedacht sind und dadurch der Zugriff auf Bilder, StyleSheets, WebServices, JavaScript usw. nicht mehr möglich ist.

Bei den regulären Ausdrücken sollte der Ausdruck so gewählt sein das der die URL mit möglichst vielen Details beschreibt (also nicht „`^~/(.*)/(.*)\.aspx`“). So das eine URL auch eindeutig erkannt werden kann. Zum Beispiel an den Stellen wo nur Ziffern erwartet werden auch entsprechend darauf prüfen.

Den Root-Operator „~“ verwenden.

Für den Zugriff auf Ressourcen (z.B. Bilder, CSS, JavaScript) innerhalb der Web-Anwendung ist es wichtig den Root-Operator in einer URL-Angabe eines Control zu verwenden. Der Root-Operator verweist immer auf den Anwendungsordner und dabei wird das Umschreiben der URL berücksichtigt.

Also `<asp:Image ImageUrl=“~/images/pictures.gif runat=“server“/>`
anstatt `<asp:Image ImageUrl=“../../images/pictures.gif“ runat=“server“/>`
verwenden.

Auch für HtmlControls kann dies gemacht werden, sofern diese auf dem Server ausgeführt werden
`<image src=“~/images/pictures.gif“ runat=“server“/>`.

Bei CSS- oder JavaScript-Datei Einbindungen innerhalb des `<header />` Tags kann das `runat=“server“` bei den `<script />` und `<link />` Tags entfallen sofern das `<header/>`-Tag selbst mit `runat=“server“` läuft. An anderen Stellen der Seite ist ein entsprechendes `runat=“server“` jedoch erforderlich.

Arbeiten mit URLs ohne Dateieindung

Wenn man URLs ohne Dateieindung verwendet, dann ist der erste Schritt den IIS so zu konfigurieren das jeder Request an den WebServer mit ASP.NET 2.0 verarbeitet wird (siehe „Einstellungen am Server“, Seite 13).

Nun sollte man nicht für URLs ohne Dateieindung eine spezielle Rewrite-Regel erstellen, sondern in der `UrlRewritingNet`-Sektion der `Web.config` eine „defaultPage“ (z.B. `default.aspx`) einstellen. Damit wird bei einem Request auf <http://meinblog.tld/albert> auf <http://mainblog.tld/albert/default.aspx> umgeleitet. Dies ist notwendig da sonst ASP.NET 2.0 bei einem `ResolveClientUrl()` die Adresse nicht richtig auflösen kann, und somit unter anderem auch Themes nicht mehr richtig funktionieren.

Die Rewrite-Regel muss dann ganz „normal“ auf die `default.aspx` reagieren und nicht auf die Adresse ohne Dateieindung.

Unterschiede zwischen IIS und Visual Studio WebDev Server

`UrlRewritingNet.UrlRewrite` funktioniert sowohl mit dem IIS (Internet Information Server) als auch mit dem im Visual Studio 2005 eingebauten WebServer. Es gibt jedoch einen kleinen Unterschied. Der im Studio eingebaute WebServer verarbeitet jeden Request mit ASP.NET 2.0 und der IIS macht dies nicht. Dies ist zu beachten wenn man andere Dateieindungen wie `(.html, .php, .xml .usw)` umschreiben

möchte, oder wenn man URLs ohne Dateiendung verwendet. Dies ist mit dem eingebauten Server ohne Probleme möglich, beim IIS muss der Server erst entsprechend konfiguriert werden. Am besten testet man die Anwendung auch auf dem IIS um Überraschungen zu vermeiden.

Installation von UrlRewritingNet.UrlRewrite

System-Voraussetzungen

UrlRewritingNet.UrlRewrite läuft mit jedem WebServer wo auch ASP.NET 2.0 drauf installiert ist. Da die einzigen Voraussetzungen ASP.NET 2.0 sind.

Getestet wurde UrlRewritingNet mit

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

Die Lauffähigkeit unter Mono wurde nicht geprüft.

Installation

Für eine Installation der UrlRewritingNet.UrlRewrite sind folgende grundlegenden Schritte erforderlich.

1. Installation der Assembly
2. Installation des Konfigurationsschemas
3. Einrichten des Konfigurationsbereiches in der Web.config
4. Einbinden von UrlRewritingNet als HttpModul

Installation der Assembly

Einfach die *UrlRewritingNet.UrlRewriter.dll* in den Bin-Ordner der Web-Anwendung kopieren

Installation des Konfigurationsschemas

Die *urlwritingnet.xsd* Datei in irgendeinen Ordner der Web-Anwendung kopieren, ohne die Schema Datei ist kein IntelliSense im Konfigurationsbereich von UrlRewritingNet möglich. Ist die Web-Anwendung teil einer Projektmappe (Solution) so kann die Schema-Datei auch irgendwo innerhalb der Solution abgelegt werden.

Einrichten des Konfigurationsbereiches

Damit aus der Web.config auch die Konfiguration gelesen werden kann muss der Bereich für UrlRewritingNet erst dem Web bekannt gemacht werden.

Dazu den Bereiche `<configSections />` am Anfang der Web.config einsetzen.

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

Sollte schon ein `<configSections />` Bereich vorhanden sein, so muss nur der `<section />` Teil in vorhandenen Bereich kopiert werden.

Ist der Bereich entsprechend bekannt gemacht worden, so muss er noch erstellt werden.

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

Diese Bereich muss direkt innerhalb von <configuration /> liegen, aber erst nach <configSections/>. Auf keinen Fall darf er innerhalb von <appSetting/> oder <system.web/> liegen. Sollte dies unklar sein, so kann dies in der Bespielanwendung nachgesehen werden.

Die Angabe des „xmlns“-Attribute ist für eine IntelliSense Unterstützung erforderlich.

Einbinden von UrlRewritingNet als HttpModul

Damit nun auch wirklich die Requests über UrlRewritingNet verarbeitet werden, muss das UrlRewritingNet.UrlRewrite als HttpModul in den ASP.NET Request-Zyklus eingebunden werden. Dies geschieht im <system.web /> Bereich der Web.config.

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

Sollte dort schon ein <httpModules/> Bereich vorhanden sein, so ist die <add /> Anweisung einfach dort einzufügen.

Nun ist UrlRewritingNet.UrlRewrite fertig konfiguriert und wartet auf den Einsatz.

Upgrade von 1.1 auf 2.0 (Notwendige Anpassungen)

Auf Grund der umfangreichen Änderungen für die Version 2.0 sind wenige Änderungen an der Konfiguration notwendig.

1. Alle Regeln brauchen nun einen eindeutigen Namen
2. Das Attribut „compileRegex“ darf nicht mehr verwendet werden
3. Konfigurationsschema aktualisieren

UrlRewritingNet gibt beim starten der Anwendung entsprechende Fehlermeldungen wenn diese Anforderungen nicht erfüllt sind.

Regeln einen Namen geben

Alle vorhandenen Regeln müssen mit einem eindeutigen Namen versehen werden, dies ist notwendig um diese im Bedarfsfall zur Laufzeit wieder zu ändern.

Wenn eine Regel vorher so aussah.

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

So muss nun das „name“ Attribut hinzugefügt werden.

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

Attribut „compileRegEx“ entfernen

Das „compileRegEx“ Attribut wird nicht mehr verwendet da zum einem die Regulären Ausdrücke immer kompiliert aufgerufen werden und zum anderen da dies nicht mehr zum Kernsystem gehört sondern zum RewriteProvider.

Aus dem Konfigurationsabschnitt `<urlrewritingnet />` muss, sofern es drin steht, das „compileRegEx“ Attribut entfernt werden.

Konfigurationsschema aktualisieren

Damit die Intellisense in der `<urlrewritingnet />`-Sektion der Web.config auch weiterhin funktioniert muss der Namespace auf die aktuelle Version angepasst werden.

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

Zusätzlich muss noch die vorhandene urlrewritingnet.xsd durch die aktuelle ersetzt werden, erst dann hat auch die IntelliSense von Visual Studio die notwendigen Informationen.

Konfiguration

Die Einstellungen von `UrlRewritingNet.UrlRewrite` werden in der `Web.config` vorgenommen, und zwar in dem Bereich der während Installation (siehe „Einrichten des Konfigurationsbereiches“ auf Seite 7) angelegt worden ist.

Hier ein kleines Beispiel:

```
<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&id=$2"
      ignoreCase="true" />
    <add name="Rule2"
      provider="MyGreatProvider"
      myattribute="/foo/bar/dhin.aspx"
      rewriteUrlParameter="ExcludeFromClientQueryString"
      rewrite="Domain"
      ignoreCase="true" />
  </rewrites>
</urlrewritingnet>
```

Diese Konfiguration kann sehr umfangreich werden, deshalb ist es möglich diese bei Bedarf in eine externen Datei auszulagern. Dazu ist das `configSource` Attribute zu verwenden.

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

Der Inhalt von `ExternalRewrite.config` entspricht dem kompletten Konfigurationsbereich.

Attribute für `<urlrewritingnet />`

Bei den Einstellungen die man mit den Attributen für `<urlrewritingnet />` festlegt sind die globalen Einstellungen für `UrlRewritingNet.UrlRewrite`.

rewriteOnlyVirtualUrls

true, false (Standard: true)

Gibt an das real auf dem Server vorhandene Seiten nicht umgeschrieben werden und direkt geladen werden.

contextItemsPrefix

Zeichenkette

Wenn die Parameter der Url auch in `HttpContext.Current.Items[]` abgelegt werden soll, so kann man hier einen Prefix eintragen der mit einem Punkt getrennt vor dem Parameter-Namen in `HttpContext.Current.Items[]`.

defaultProvider

ProviderName (Standard: RegEx)

Name des RewriteProvider der verwendet wird wenn bei einer Rule nicht explizit einer angegeben wird.

defaultPage

Dateiname

Dateiname der Standardseite zu der weitergeleitet wird wenn man auf eine URL ohne Dateiendung zugreift (siehe „Arbeiten mit URLs ohne Dateiendung“ Seite 5).

Die <providers /> Auflistung

Hier werden die Provider die man verwendet eingetragen, genaueres findet sich unter „Einbinden des Providers in die Web.config“ auf Seite 18 und in der Dokumentation des entsprechenden Providers.

Die <rewrites /> Auflistung

In der Rewrites Auflistung werden alle Rewrite-Regeln eingetragen, diese Regeln werden in der Reihenfolge bearbeitet wie sie hier aufgelistet werden. Die erste Regel die sich für eine URL zuständig fühlt erhält den Zuschlag, danach werden für die Request keine weiteren Regeln überprüft.

Hinzugefügt werden die Regeln mit dem <add /> Element, dieses enthält dann die entsprechenden Attribute. Die Standard Attribute für alle Regeln sind folgende.

name

Regelname

Name der Regel die man erstellen will, dieser ist frei wählbar muss jedoch eindeutig sein. Mit diesem Namen kann die Regel auch programmatisch angesprochen werden (siehe „Ändern von Rewrite-Regeln zu Laufzeit“ Seite 15)

provider

ProviderName

Name des RewriteProvider der für diese Regel verwendet wird, gibt man keinen an so wird der defaultProvider verwendet.

redirect

None, Application, Domain (Standard: None)

Wenn ein Redirect anstelle eines Rewrites gemacht werden soll.

None Normaler Rewrite

Application Redirect innerhalb der Webanwendung

Domain Redirect zu einer anderen Domain, die Domain muss dann Bestandteil der Ziel-URL sein.

redirectMode

Permanent, Temporary (Standard: Temporary)

Wenn ein Redirect gemacht werden soll kann man hiermit entscheiden ob dieser als Permanent (HTTP-Status Code 301) oder Temporär (HTTP-Status Code 302) gekennzeichnet wird. Suchmaschinen können auf diesen Statuscode reagieren.

rewrite

Application, Domain (Standard: Application)

Gibt an ob beim Rewrite die Domain mit berücksichtigt werden soll.

rewriteUrlParameter

ExcludeFromClientQueryString, StoreInContextItems, IncludeQueryStringForRewrite (Standard: ExcludeFromClientQueryString)

ExcludeFromClientQueryString

Eventuelle in **destinationUrl** (im Falle des RegExRuleProvider) angegebene Url-Parameter werden nicht in *Page.ClientQueryString* eingetragen. So das auch ein einwandfreier Postback möglich ist. Url-Parameter die in der Browser-Adressleiste stehen sind weiterhin in *Page.ClientQueryString* vorhanden. In *Request.QueryString[]* sind jedoch alle Url-Parameter vorhanden.

StoreInContextItems

Alle Url-Parameter, die aus dem Rewrite und diejenigen die im Browser stehen, werden auch in *HttpContext.Current.Items[]* abgelegt, vor dem Namen wird ein eventuell angegebener **contextItemsPrefix** gesetzt.

IncludeQueryStringForRewrite

Bei einem Rewrite werden auch der QueryString an die Regel übergeben und kann somit berücksichtigt werden.

Als Entwickler eines Rewrite-Provider ist man gehalten seinen Provider so zu entwickeln dass er dem Standardverhalten entspricht, jedoch kann mit anderen Provider das Verhalten abweichen.

RegEx Rewrite Attribute

Der Standard Rewrite Provider für reguläre Ausdrücke hat noch folgende Attribute.

virtualUrl

regulärer Ausdruck

Ein regulärer Ausdruck auf den geprüft wird und eine Ersetzung durch **destinationUrl** gemacht

destinationUrl

regulärer Ersetzungsdruck

Ein regulärer Ersetzungsdruck der die eigentliche Zielseite beschreibt.

regexOptions

Multiline, ExplicitCapture, Singleline, IgnorePatternWhitespace, RightToLeft, ECMAScript, CultureInvariant

Für eine optimale Kontrolle über den Regulären Ausdruck kann zusätzlich zu **ignoreCase** hier alle weiteren *RegexOptions* gesetzt werden. Dies ist jedoch nur für Spezialfälle nötig. Ausführliche Erklärungen dazu finden sich in der MSDN Dokumentation.

Einstellungen am Server

UrlRewritingNet.UrlRewrite funktioniert im Normalfall ohne jegliche Administrative Einstellungen am Server. Es gibt jedoch Anforderungen wo dies notwendig werden kann, wenn z.B. URLs mit bestimmten Dateiendungen (z.B. .html, .xml, .htm usw.) umgeschrieben werden sollen, oder wenn URLs ohne Dateiendung verwendet werden sollen.

Wir empfehlen dies jedoch zu vermeiden und nur anzuwenden wenn genau bekannt ist auf welchen Servern die Anwendung eingesetzt wird und man selbst die Konfiguration bestimmen kann.

Andere Dateiendungen ASP.NET 2.0 zuordnen

IIS 5.0/5.1

IIS 6.0

Öffnen des IIS Managers, auf das entsprechende zu konfigurierende Web gehen und die Eigenschaften aufrufen. Dort in dem Register „Basisverzeichnis (Home Directory)“ bei den Anwendungseinstellungen (Application settings) auf „Konfiguration (Configuration)“ klicken.

Nun auf dem Register „Zuordnungen (Mappings)“ zusätzliche Dateiendungen hinzufügen. Dazu einfach folgende Einstellungen eintragen.

Ausführbare Datei (Executable)

c:\windows\microsoft.net\framework\v2.0.50727\aspnet_isapi.dll
(kann je nach Installation abweichen)

Erweiterung (Extension)

.html
(Beispiel für.html)

Verben, begrenzen auf (Verbs, Limit to)

GET,HEAD,POST,DEBUG

Skriptmodul (Script engine)

anhaken

Verifizieren das Datei existiert (Verify that file exists)

haken wegmachen, da sonst kein Rewrite Möglich ist.

Alle Requests durch ASP.NET 2.0 leiten

IIS 5.0/5.1

Beim IIS 5.0 und IIS 5.1 muss der Server so eingestellt werden dass alle Dateien mit der Endung *.* von der aspnet_isapi.dll verarbeitet wird.

IIS 6.0

Beim IIS 6 ist es nicht mehr möglich *.* der aspnet_isapi.dll zuzuweisen, dort besteht jedoch Möglichkeit ISAPI Filter als Anwendungsplatzhalter (Wildcard application maps) hinzuzufügen.

Dazu einfach wie unter „Andere Dateiendungen ASP.NET 2.0 zuordnen, IIS 6.0“ auf Seite 13 beschrieben vorgehen, jedoch keine Dateiendung hinzufügen sondern unter **Anwendungsplatzhalter** (Wildcard

application maps) „c:\windows\microsoft.net\framework\v2.0.50727\aspnet_isapi.dll“ hinzufügen.
Auch dort darf bei **Verifizieren das Datei existiert** (Verify that file exists) kein haken gemacht werden.

Ändern von Rewrite-Regeln zu Laufzeit

Für das ändern von Rewrite-Regeln zur Laufzeit stellt `UrlRewritingNet.UrlRewrite` vier statische Methoden zu Verfügung.

```
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);
    }
}
```

Die Parameter sind sich meist sehr ähnlich, `ruleName` gibt den Namen der Regel an die man sich bezieht und `rewriteRule` ist die neue `RewriteRule`.

Mit `AddRewriteRule()` fügt man eine neue Regel hinzu und stellt diese ans Ende der Liste, mit `RemoveRewriteRule()` wird eine Regel entfernt.

`ReplaceRewriteRule()` ersetzt eine vorhandene Regel gleichen Namens.

Mit `InsertRewriteRule()` kann man eine Regel an einer bestimmten Position einfügen, die Position gibt man mit `positionRuleName` an.

Es muss immer eine Instanz einer speziellen `RewriteRule`-Klasse erstellt und diese dann mit den gewünschten Parametern bestücken. Ist eine Rule geändert steht Sie während der Laufzeit der Anwendung zu Verfügung. Wird die Anwendung neu gestartet sind alle Änderungen weg und müssen neu vorgenommen werden.

Hier ein kleines Beispiel:

```
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");
```

Erstellung eines eigenen Rewrite-Rule Provider

Falls die Möglichkeiten der Regulären Ausdrücke nicht ausreichen sollten, besteht die Möglichkeit sich einen eigenen Rewrite-Rule Provider zu entwickeln und dann einzubinden.

Die Entwicklung eines Providers lässt sich in 3 Schritte einteilen.

1. Entwicklung der Rewrite-Logik
2. Erstellung des Providers
3. Einbinden des Providers in die Web.config

Dies ist auch schon alles, um den ganzen Rest wie das Umschreiben der URLs innerhalb von ASP.NET, das Themes, das Caching usw. funktionieren, darum kümmert sich weiterhin `UrlRewritingNet.UrlRewrite`.

Hier wird als Beispielimplementierung der eingebaute RegEx Provider verwendet.

Entwicklung der Rewrite-Logik

Nach welchen Prinzipien Ihr Rewrite-Regel arbeitet können wir natürlich nicht vorgeben, da sind sie völlig frei. Die Klasse die die Logik implementiert muss von der abstrakten Klasse `RewriteRule` abgeleitet sein. Dort sind dann die drei Methoden zu überschreiben `Initialize()`, `IsRewrite()` und `RewriteUrl()`.

Ein Objekt der Klasse wird dann jeweils vom Provider Instanziert wenn eine entsprechende `RewriteRule` in der Konfiguration angegeben ist. Das Objekt wird nur einmal je Regel zur Laufzeit instanziiert. Es ist darauf zu achten das Web Anwendungen grundsätzlich Multithreaded sind; eine eventuell nötige Synchronisation auf Daten innerhalb des Objektes muss gewährleistet sein. Da die Methoden prinzipiell mehrfach gleichzeitig aufgerufen werden sollte. Man sollte die Methoden so entwickeln als ob es statische Methoden wären.

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

Wird einmalig beim Instanzieren der Klasse aufgerufen, darin sollten die benötigten `RewriteSettings` ausgelesen werden. Dies können die Standard-Settings sein, oder auch Regel spezifische die in der Konfiguration angegeben werden. Die Klasse `RewriteSettings` stellt Methoden zu Verfügung um diese auslesen zu können.

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

In dieser Methode sollte nur ein kurzer Test passieren ob die übergebene URL von dieser Regel erfasst wird (return true;) oder nicht (return false;). Mehr sollte hier nicht passieren. Auch sollten wenn möglich keine Daten schon zwischengespeichert werden die dann von `RewriteUrl()` gebraucht werden.

```
public override string RewriteUrl(string url)
```

Mit dieser Methode wird dann die Magie implementiert die aus der virtuellen URL die physikalische URL bildet. Als Rückgabe wird die „echte“ Adresse erwartet. Auch ist hier zu berücksichtigen ob es ein Domain oder Application Redirect/Rewrite war.

Hier die Beispiel Implementierung von 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);
    }
}
```

Erstellung des Providers

Der Provider hat nur eine Aufgabe, ein Objekt der eigenen RewriteRule Klasse zu instanziiieren. Weitere Logik ist nicht erforderlich dieser Weg wurde gewählt damit nicht zur Laufzeit noch „erst“ der notwendige Provider gesucht werden muss und das ganze ein wenig optimierter abläuft.

Der eigenen Provider muss von der Klasse `UrlRewritingRule Provider` abgeleitet sein und auch nur eine Methode implementieren.

Hier eine komplette Klasse.

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

Das war es schon, mehr ist nicht notwendig. Sollte in `Initialize()` notwendig sein und der eigene Provider zusätzliche Parameter braucht, dann kann dies natürlich gemacht werden.

Einbinden des Providers in die Web.config

Damit `UrlRewritingNet` auch mit dem Provider arbeiten kann, muss dieser in der `Web.config` deklariert werden. Dafür muss einfach eine `<providers/>` Auflistung erstellt werden. Nach dem ASP.NET 2.0 Provider Standard.

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

Es kann eine beliebige Anzahl von eigenen Provider verwendet werden, durch die Angabe des Providernamen in einer Regel kann jeder Regel mit einem anderen Provider arbeiten.

Der Standard-RegEx Provider wird immer eingebunden und mit dem Namen „RegEx“ angesprochen, es sei denn es wird ein eigener Provider mit dem Namen „RegEx“ in der Konfiguration eingetragen.

Eine Beispielkonfiguration:

```
<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&show=$2"
      ignoreCase="true" />
    <add name="Rule2"
```

```
    provider="MyProviderName"  
    rewriteUrlParameter="ExcludeFromClientQueryString"  
    ignoreCase="true" />  
</rewrites>  
</urlrewritingnet>
```

Copyright

```
/* UrlRewritingNet.UrlRewrite
 * Version 2.0
 *
 * Diese Library ist Copyright 2006 by Albert Weinert and Thomas Bandt.
 *
 * http://der-albert.com, http://blog.thomasbandt.de
 *
 * Diese Library wird so zu Verfügung gestellt wie sie ist.
 * Es werden Garantien gegeben oder sind impliziert
 *
 * Die Library kann kostenlos in freien sowie kommerziellen Projekten verwendet
 * werden.
 *
 * Wenn die Library anderen zu Verfügung gestellt wird muss dies
 * unentgeltlich geschehen.
 *
 * Es ist erlaubt den Quelltext an die eigenen Anforderungen anzupassen.
 * Wenn Verbesserungen gemacht werden dann müssen diese öffentlich verfügbar
 * gemacht werden. Entweder werden diese uns zugesandt oder auf einer
 * eigenen Website veröffentlicht. Sollten die Änderungen auf einer eigenen
 * Website veröffentlicht werden so sind wir entsprechend zu benachrichtigen.
 * Dies ist kein Versprechen das auch wir die Änderungen übernehmen.
 *
 * Diese Copyright-Notiz (in englischer Sprache) muss in dem geänderten
 * Quelltext verbleiben.
 *
 * Es ist nicht erlaubt eine kommerzielle Rewrite Engine die auf dieser
 * Library basiert zu erstellen.
 *
 * Basiert auf http://weblogs.asp.net/fmarguerie/archive/2004/11/18/265719.aspx
 *
 * Für weitere Informationen siehe: http://www.urlrewriting.net/
 */
```