DotNetNuke Friendly Urls

Scott McCulloch

Version 1.0.0

Last Updated: June 20, 2006

Category: HTTP Modules

Information in this document, including URL and other Internet Web site references, is subject to change without notice. The entire risk of the use or the results of the use of this document remains with the user.

The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, places, or events is intended or should be inferred.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Perpetual Motion Interactive Systems, Inc. Perpetual Motion Interactive Systems may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Perpetual Motion, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

DotNetNuke® and the DotNetNuke logo are either registered trademarks or trademarks of Perpetual Motion Interactive Systems, Inc. in the United States and/or other countries.

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

Abstract

In order to clarify the intellectual property license granted with contributions of software from any person or entity (the "Contributor"), Perpetual Motion Interactive Systems Inc. must have a Contributor License Agreement on file that has been signed by the Contributor.

Contents

DotNetNuke 3.0 Friendly Url's Guide

Introduction

DotNetNuke is a content management system that allows webmasters to quickly build web sites (portals). Webmasters can create, modify and delete any number of pages (previously known as tabs) within their portal.

To provide this functionality, DotNetNuke will store the details about the portal (page hierarchy) and the page content (known as modules) inside a data store (default is a SQL Server database).

As with most data structures a unique key is used to retrieve content out of the data store. Traditionally, this key has been known as "TabID", a parameter that appears in most DotNetNuke URLs for version's 1.x and 2.x. As a page is requested, typically a "TabID" is provided to identify the page within the application's data store.

A traditional URL might have looked like:-

http://www.dotnetnuke.com/default.aspx?tabid=510

When the page is processed, a TabID is used to identify a number of items:-

- The content for the current page.
- The portal for the current page (TabID's (pages) belong to a portal)

The above however, has a number of problems. Firstly, the above URL is not easily remembered by a human, e.g. it is **not human friendly**. Secondly, it is **not search engine friendly**, most search engines will ignore or avoid pages with parameters.

This guide will investigate the requirements, the concepts and the implementation around Friendly URLs for DotNetNuke.

Requirements

Each enhancement proposed for DotNetNuke begins with a User Story that describes the requirements for the enhancement. You can view the requirements for this enhancement at the following URL:

http://www.dotnetnuke.com/Default.aspx?tabid=622

Search Engine Friendly

The term **search engine friendly** relates to how well a search engine will index your site. Traditionally, this has been a weakness for DotNetNuke as the URL contains parameters in the querystring.

The querystring can be defined as the section of the URL that appears after the question mark:-

http://www.dotnetnuke.com/Default.aspx?tabid=622

When a search engine indexes a page with a URL containing a querystring, it will typically perform a shallow crawl or simply avoid that page. The search engine will identify the URL as part of a dynamic application, and hence will change the way it index's that section of content. Typically, this is to limit the possibility of overloading the site or adversely affecting that site's performance during indexing.

While different search engines vary in the way they index dynamic sites, it is commonly recognized that URLs containing a querystring are not indexed as well as sites that do not.

The URL should also contain as much information about the actual page you are on as possible, this could include placing the name of the page, or the page hierarchy into the URL. This helps some search engines with keyword placement based on the URL.

Human Friendly

The term **human friendly** relates to how easily remembered or understood a page's URL is to a human. For example, it is much easier to remember a page called:-

http://www.dotnetnuke.com/newsroom.aspx

Then a URL such as:-

http://www.dotnetnuke.com/default.aspx?tabid=703

While not part of the initial requirements of the enhancement, it was well understood that this was a desire of the community; hence, the architecture should accommodate the possibility of implementing this requirement in the future.

Child Portals

DotNetNuke has a concept of **child portals** which are essentially sub-portals beneath a domain e.g. www.domain.com/child.

There are two parts to this requirement; the first is the **removal of the creation of a physical folder** that is required when creating a child portal. This solution is currently in place to overcome a limitation in IIS where requests are not mapped to the ASPNET worker process unless a file with a valid ASPNET extension is requested. Currently, the folder redirects back to the root directory with an alias of the requested child portal.

The second requirement is the retention of the identity of that child portal (e.g. www.domain.com/child) in subsequent requests to that child portal.

Other Applications

The feature should not affect the processing of Requests for other applications installed in the shared hosting environment.

Performance

The feature should not adversely affect performance (within reason).

Architecture Concepts

When approaching the final solution, a number of approaches were taken and dismissed. The final solution became clear once an understanding was reached about what actually we were trying to achieve. This section will explain the concepts so that you help to understand the implementation.

The best analogy for a URL rewriter I believe is a language interpreter. So we will use this as an example to introduce our concepts.

A language interpreter works as a translator between two parties that would otherwise be unable to communicate. The role of the interpreter is to receive messages from one party and translate those messages into another format the other party can understand.

In the scenario below, we have two people, one that speaks English and the other that speaks German. The English person wants to communicate with the German person, so he/she sends a message to the interpreter that relays the message to the German person. During the relaying of the message, the interpreter converts the message into a format understood by the German, e.g. German dialect. If the German wants to reply, he/she sends the message back through the interpreter, who translates it to English and forwards it on.

Error! Objects cannot be created from editing field codes.

Figure 1: English to German Translation

So, the above example is a quite common scenario, an interpreter has the rules (knowledge of the dialect) to interpret one message and translate it into another format. This scenario also applies to friendly URLs. A browser requests a URL as a friendly format not recognized by the native application, and an intermediate process translates that request into a format understood by that native application.

Error! Objects cannot be created from editing field codes.

Figure 2: Browser to App URL Translation

Just as the language interpreter had to process the rules for the translation from English to German (gained via the knowledge of the German language and the word equivalents in each language), the friendly URL translation must also call upon a set of rules for converting the incoming URL to a URL that the native app will understand. This translation via a set of rules is called the **URL Scheme**.

Just as there are problems when converting words from one language to another, (e.g. some words might not exist or many words exist for a singular), the translator must make a decision on how it translates any given message. Therefore, the **URL Scheme** becomes a very important aspect of any solution if it is to uniquely map one resource to another.

So far we have only talked about interpreting incoming messages, but the same applies for outgoing messages. In our case, these are the URLs present in any page. A translation of these URLs must take place and must conform to the **URL Scheme** for the translator. This translation for human friendly URLs might take place inside our head if we were to guess that the news room page is at the “newsroom.aspx” page by typing that URL in our browser. Typically, a method or algorithm would do the processing for us.

Architecture Components

In the last section (architecture concepts) we discussed the interpretation of an incoming and outgoing messages via a series of rules known as a URL Scheme. Our components reflect the nature of this interpretation and are situated in both the incoming and outgoing processing of a HTTP request.

URL Scheme

We defined earlier that the URL scheme relates to the rules on how a URL is translated from one format to another.

- Our original requirements (in priority) were:-
- Remove querystring parameters
- Do not adversely affect performance
- Retain child portal in URL
- Human Friendly (if possible, while not affecting performance)

The default scheme in DotNetNuke 3.0 is to rewrite the URL, by simply rewriting the parameters into the URL itself.

So the following URL:-

http://www.dotnetnuke.com/default.aspx?tabid=703

Is translated into (and vice versa):-

http://www.dotnetnuke.com/tabid/703/default.aspx

The above simply becomes a simple rewrite of the parameters (e.g. **no more querystring parameters**), and **performance is not adversely impacted.** (Remember, we had a limited time on this enhancement. More detail was put into the actual pluggable nature of the enhancement, then actually the default scheme, so this **scheme is not human friendly**.)

To satisfy the other requirements in the Search Engine Friendly requirement, information about the Tab Path (more suitably know as the page hierarchy) was injected into the URL.

So if you were at a page nested down a few levels such as:-

Home > About > News Room

The URL would look like:-

http://www.dotnetnuke.com/About/NewsRoom/tabid/703/Default.aspx

The above is a useful feature, because some search engines examine parts of the URL when indexing a page. The actual rules for the scheme (for interpretation) actually ignore this section, but it is still useful to have this in the URL.

As of DotNetNuke 3.2, only alphanumeric characters will be rewritten into the URL, along with the space, dash (-) and the underscore (_) characters. All other characters will be left in the query string.

For Example:-

http://www.dotnetnuke.com/default.aspx?tabid=703&param=special/char

Becomes:-

http://www.dotnetnuke.com/tabid/703/default.aspx?param=special/char

The last requirement was to leave the identity of the child portal in the URL. This was also built into the default scheme.

http://www.dotnetnuke.com/Child/About/NewsRoom/tabid/703/default.aspx

To summarize, if we examine the above URL, it can map uniquely to a single page because the URL still contains the primary key from the data store (**tabid)**. The only problem we have not solved is the human friendly URLs, but by examining the pluggable nature of the solution, 3rd parties will be able to provide such a scheme.

HTTP Module (Request)

When translating an incoming URL, the request needs to be intercepted as soon as possible and translated to the actual URL used by ASP.NET.

ASP.NET provides a facility to intercept calls into a page. The process a request goes through to reach a page is known as the HTTP pipeline. A method of intercepting a request is through the use of a HTTP module.

HTTP modules can be used to intercept specific events in a request such as BeginRequest, EndRequest, etc. The BeginRequest event is perfect for our needs because it allows us to insert code before the request has hit the actual application. In this case, it gives us a chance to apply our scheme, translate the URL and rewrite it.

The other advantage of a HTTP module is that allows us to optionally enable, disable or replace the module. For this reason, a HTTP module makes it quite easy to implement your own URL scheme.

The HTTP module in the default scheme makes the decisions based on the logic it contains. It will also examine the map of URLs contained in SiteUrls.config contained in the root folder of your DotNetNuke installation. It is possible to manually place definitions in this file that will make those pages **human friendly**.

For example, if you wanted to add an entry for a specific page on your site, let's say,

http://www.dotnetnuke.com/About/NewsRoom.aspx

Which maps to:-

http://www.dotnetnuke.com/About/NewsRoom/tabid/703/Default.aspx

Your file might look like this:-

```
<?xml version="1.0" encoding="utf-8" ?>
<RewriterConfig>
   <Rules>
      <RewriterRule>
         <LookFor>.*/About/NewsRoom.aspx</LookFor>
         <SendTo>~/default.aspx?tabid=703</SendTo>
      </RewriterRule>
      <RewriterRule>
         <LookFor>.*/TabId/(\d+)(.*)/Logoff.aspx</LookFor>
         <SendTo>~/Admin/Security/Logoff.aspx?tabid=$1</SendTo>
      </RewriterRule>
      <RewriterRule>
```

```
         <LookFor>.*/TabId/(\d+)(.*)/rss.aspx</LookFor>
         <SendTo>~/rss.aspx?TabId=$1</SendTo>
      </RewriterRule>
      <RewriterRule>
         <LookFor>.*/TabId/(\d+)(.*)</LookFor>
         <SendTo>~/Default.aspx?TabId=$1</SendTo>
      </RewriterRule>
   </Rules>
</RewriterConfig>
```

The above is quite a powerful feature, but requires manual intervention on this file.

FriendlyURL Method (Response)

So far we have achieved a method of interpreting an incoming URL via the use of a HTTP module. This module will translate an **incoming** URL into another URL that the underlying application (DotNetNuke) can understand.

What's left is to devise a method of formatting the URLs that are embedded in the HTML document we send to the client (**outgoing**). Unlike the request, the response may contain many URLs in need of conversion and is intertwined with other presentation text such as HTML tags.

In previous versions of DotNetNuke, all internal communication inside the application had been through two methods (NavigateUrl & EditUrl). It was quite simple to apply additional logic inside both of these methods to format each URL into the chosen friendly URL scheme.

The scheme would be chosen through the use of a provider that would translate any given URL into its friendly format. The provider would also allow this piece to be "pluggable".

You can implement your own provider by writing your own implementation of the following methods.

```
Public MustOverride Function FriendlyUrl(ByVal tab As TabInfo, ByVal path As String) As
String
Public MustOverride Function FriendlyUrl(ByVal tab As TabInfo, ByVal path As String,
ByVal pageName As String) As String
Public MustOverride Function FriendlyUrl(ByVal tab As TabInfo, ByVal path As String,
ByVal pageName As String, ByVal settings As PortalSettings) As String
Public MustOverride Function FriendlyUrl(ByVal tab As TabInfo, ByVal path As String,
ByVal pageName As String, ByVal portalAlias As String) As String
```

And changing the following configuration section in web.config:-

```
<friendlyUrl defaultProvider="DNNFriendlyUrl">
   <providers>
      <clear />
      <add name="DNNFriendlyUrl"
      type="DotNetNuke.Services.Url.FriendlyUrl.DNNFriendlyUrlProvider,
DotNetNuke.HttpModules.UrlRewrite" />
   </providers>
</friendlyUrl>
```

Review

In this document, we have reviewed the concepts of URL translation and examined the architecture of our solution. In examining the architecture, we investigated the default URL scheme that we have implemented by default. This scheme, like any other scheme, will have its advantages and disadvantages.

Like other solutions in the DotNetNuke framework, the solution is customizable via the provider pattern and through the use of a HTTP module. We hope to see 3rd parties take this opportunity to expand on features we have built in the default scheme.

Best Regards,

The DotNetNuke Team

Additional Information

The DotNetNuke Portal Application Framework is constantly being revised and improved. To ensure that you have the most recent version of the software and this document, please visit the DotNetNuke website at:

http://www.dotnetnuke.com

The following additional websites provide helpful information about technologies and concepts related to DotNetNuke:

DotNetNuke Community Forums
http://www.dotnetnuke.com/tabid/795/Default.aspx

Microsoft® ASP.Net
http://www.asp.net

Open Source
http://www.opensource.org/

W3C Cascading Style Sheets, level 1
http://www.w3.org/TR/CSS1

Errors and Omissions

If you discover any errors or omissions in this document, please email marketing@dotnetnuke.com. Please provide the title of the document, the page number of the error and the corrected content along with any additional information that will help us in correcting the error.

Appendix A: Document History

Version	Last Update	Author(s)	Changes
1.0.0	Aug 16, 2005	Shaun Walker	• Applied new template
1.0.1	Oct 10, 2005	Scott McCulloch	• Updated for 3.1.1/3.2 Changes