Endeca URL Optimization API for the RAD Toolkit for ASP.NET

Developer's Guide

Version 2.1.3 • March 2012

ORACLE

ENDECA

Contents

Copyright and disclaimer

Preface

Oracle Endeca's Web commerce solution enables your company to deliver a personalized, consistent customer buying experience across all channels — online, in-store, mobile, or social. Whenever and wherever customers engage with your business, the Oracle Endeca Web commerce solution delivers, analyzes, and targets just the right content to just the right customer to encourage clicks and drive business results.

Oracle Endeca Guided Search is the most effective way for your customers to dynamically explore your storefront and find relevant and desired items quickly. An industry-leading faceted search and Guided Navigation solution, Oracle Endeca Guided Search enables businesses to help guide and influence customers in each step of their search experience. At the core of Oracle Endeca Guided Search is the MDEX Engine,™ a hybrid search-analytical database specifically designed for high-performance exploration and discovery. The Endeca Content Acquisition System provides a set of extensible mechanisms to bring both structured data and unstructured content into the MDEX Engine from a variety of source systems. Endeca Assembler dynamically assembles content from any resource and seamlessly combines it with results from the MDEX Engine.

Oracle Endeca Experience Manager is a single, flexible solution that enables you to create, deliver, and manage content-rich, cross-channel customer experiences. It also enables non-technical business users to deliver targeted, user-centric online experiences in a scalable way — creating always-relevant customer interactions that increase conversion rates and accelerate cross-channel sales. Non-technical users can control how, where, when, and what type of content is presented in response to any search, category selection, or facet refinement.

These components — along with additional modules for SEO, Social, and Mobile channel support — make up the core of Oracle Endeca Experience Manager, a customer experience management platform focused on delivering the most relevant, targeted, and optimized experience for every customer, at every step, across all customer touch points.

About this guide

This guide describes the major tasks involved in developing an application that utilizes the Endeca URL Optimization API for the RAD Toolkit for ASP.NET.

This guide assumes that you are familiar with Endeca's terminology and basic concepts.

This guide covers only the features of the Endeca URL Optimization API for the RAD Toolkit for ASP.NET, and is not a replacement for the available material documenting other Endeca products and features.

Who should use this guide

This guide is intended for developers who are building applications that leverage both the Endeca URL Optimization API and the Endeca RAD Toolkit for ASP.NET.

This document assumes that the reader has a working knowledge of the following software and concepts:

- Basic Endeca concepts such as dimensions, dimension values, refinements, ancestors, records, aggregate records, etc.
- Configuring Endeca dimensions using Developer Studio
- Endeca RAD Toolkit for ASP.NET, specifically the following objects:
  - `UrlManager`
  - `UrlBuilder`
  - `Dimension`
  - `DimensionValue`
  - `DimensionState`

If you are using the Endeca URL Optimization API in conjunction with the Sitemap Generator, please read the *Sitemap Generator Usage Guide* in addition to this URL Optimization API Guide. This guide is not a replacement for the *Sitemap Generator Usage Guide*.

Conventions used in this guide

This guide uses the following typographical conventions:

Code examples, inline references to code elements, file names, and user input are set in `monospace` font. In the case of long lines of code, or when inline monospace text occurs at the end of a line, the following symbol is used to show that the content continues on to the next line: ¬

When copying and pasting such examples, ensure that any occurrences of the symbol and the corresponding line break are deleted and any remaining space is closed up.

Contacting Oracle Endeca Customer Support

Oracle Endeca Customer Support provides registered users with important information regarding Oracle Endeca software, implementation questions, product and solution help, as well as overall news and updates.

You can contact Oracle Endeca Customer Support through Oracle's Support portal, My Oracle Support at *https://support.oracle.com*.

Chapter 1

Installation

This section describes installation procedures for the Endeca URL Optimization API.

System requirements

This section provides a list of minimum system requirements for the Endeca URL Optimization API for the RAD Toolkit for ASP.NET.

Endeca software requirements

The Endeca URL Optimization API for the RAD Toolkit for ASP.NET requires the following Endeca packages:

- Platform Services
- MDEX Engine
- RAD Toolkit for ASP.NET

Note: The Endeca Presentation and Logging APIs for .NET are part of the required RAD Toolkit installation and are installed into `<`*`installation path`*`>\Endeca\RADToolkits\<`*`version`*`>\ASP.NET\bin`.

To determine the compatibility of the URL Optimization API with other Endeca installation packages, see the *Oracle Endeca Guided Search Compatibility Matrix* available on the Oracle Technology Network (OTN).

Other software requirements

The following third party software must be installed before you install the URL Optimization API for the RAD Toolkit for ASP.NET:

- Microsoft .NET Framework. The URL Optimization API for the RAD Toolkit for ASP.NET is supported for all versions of the Microsoft .NET Framework that are supported by the RAD Toolkit for ASP.NET.
- Microsoft Visual Studio 2005 with Service Pack 1 (any edition) or Visual Studio 2008.
- On machines running the RAD Toolkit for ASP.NET reference application, you need Internet Information Services (IIS) 5.1 or later.

Supported operating systems

The URL Optimization API is supported on Windows Server 2003 and Windows Server 2008 R2 Enterprise.

Installing the URL Optimization API

The URL Optimization API for the RAD Toolkit for ASP.NET is distributed as a zip file, `UrlOptimizationAPIRADNET-<version>.zip`. It can be unpacked using WinZip or an alternate decompression utility, and may be unzipped into any location.

To install the URL Optimization API:

1. Extract `UrlOptimizationAPIRADNET-<version>.zip` using WinZip or an alternate decompression utility.

   Oracle recommends that you install the URL Optimization API to the same directory as your Endeca installation. For example, extracting to `C:\` (on Windows) creates a directory structure of `C:\Endeca\SEM\URLOptimizationAPIs\RAD Toolkit for ASP.NET\<`*`version`*`>`.
2. Navigate to the`\bin` subdirectory of the URL Optimization API.
   For example: `C:\Endeca\SEM\URLOptimizationAPIs\RAD Toolkit for ASP.NET\<`*`version`*`>\bin`
3. Copy the `Endeca.Web.Url.Seo.dll` and the `Endeca.Web.Url.Seo.xml` files into the `bin` directory of your own application.
   For example: `C:\Apps\MyApp\bin`

After installing the URL Optimization API, you must integrate it with your Web application to see results. See "Setting up a Reference Application" in this guide for an example of how to integrate URL Optimization with the Content Assembler Reference Application.

Package contents

The URL Optimization API package includes a number of components. This section is a summary of the information included in each of these components and a description of their location in the distribution directory.

The `C:\Endeca\SEM\URLOptimizationAPIs\RAD Toolkit for ASP.NET\<`*`version`*`>` directory is the root for URL Optimization API. The directory contains the following components:

Component	Description
Root directory	Contains the release notes `README_UORAD.txt`.
`bin`	Contains the `Endeca.Web.Url.Seo.dll` and the `Endeca.Web.Url.Seo.xml` files. These files contain the full implementation of the URL Optimization API.
`doc`	Contains the *Oracle Endeca Guided Search Third-Party Software Usage and Licenses* document and the generated API documentation.

Component	Description
`samples`	Contains the sample form `rewriter`, the sample `SeoUrl¬Provider`, the sample Sitemap Generator integration files and the sample Web site files.
`Sample\SitemapGeneratorIntegration`	Contains a example `SeoUrlProvider` and example Sitemap Generator files to exhibit URL Optimization API and Sitemap Generator integration.
`samples\UrlProvider`	Contains the sample `SeoUrlProvider`.

Chapter 2

Introduction

This section provides an introduction to the URL Optimization API and its capabilities.

Introduction to URL optimization

Dynamically generated URLs that are comprised of meaningless strings and no keywords may negatively impact search engine ranking as well as user experience. As an answer to this problem, the Endeca URL Optimization API enables users to create site links using directory-style URLs that include keywords and store the dynamic information in the base URL rather than in the query string.

The resulting URLs do not contain any URL query parameters. Instead, all of the necessary Endeca values are stored in the URL path, resulting in search engine-friendly URLs.

Overview of the URL Optimization API capabilities

The URL Optimization API is designed to help increase your natural search engine rankings by enabling the creation of search engine-friendly URLs.

Integration of keywords into the URL string

Many search engines take URL strings in as part of their relevancy ranking strategy. Generating URLs that include keywords can increase your natural search engine ranking as well as create visitor-friendly URLs that are easier for front-end users to understand.

Using the URL Optimization API, you can configure the following strings to display in the URL:

- Dimension names
- Dimension value names
- Ancestor names
- Record property strings
- Text search strings

For example, the base URL for the Merlot page can be configured to include ancestors in the string: `http://localhost/ContentAssemblerRefApp/Content.aspx/Wine-Red-Merlot/`

The optimized URL is more comprehensible to front-end users and more search-engine friendly than the traditional URL which contains no keywords: `http://localhost:8888/endeca_jspref/con¬troller.jsp?sid=122C7EA4C912&Ne=6200&enePort=15000&eneHost=localhost&N=8025`

Canonicalizing the URL string

Dynamic sites often produce syntactically different URLs for the same page. Multiple variant URLs result in duplicate content and therefore lower natural search engine ranking.

For example, users might be able to reach the Napa white wine page by first clicking on “Napa” and then clicking on “White”, or by first clicking on “White” and then “Napa.” This creates two syntactically unique links pointing to the same Napa White page:

- `http://localhost:8888/urlformatter_jspref/controller/Wine-White/Region-Germany/_/N-1z141vcZ66t`
- `http://localhost:8888/urlformatter_jspref/controller/Region-Germany/Wine-White/_/N-1z141vcZ66t`

To ensure that only one version of the URL per page is used in links throughout the site, the URL Optimization API provides provides options for canonicalizing URLs.

Configuring the word separator string

It is possible to customize the word separator for each keyword string in the URLs. By default, the word separator is the dash character "-":

```
http://localhost:8888/urlformatter_jspref/controller/Wine-White/Region-
Germany/_/N-1z141vcZ66t
```

Moving Endeca URL parameters out of the query string

In order to create directory-style URLs, you can limit the number of Endeca parameters in the query string by moving them from the query string and into the path-params section of the URL.

For example, the following URL has the Endeca parameters N, Ntk, Ntt, and Ntx in the query string:

```
http://localhost/ContentAssemblerRefApp/Content.aspx/Bor¬
deaux?N=4294966952&fromsearch=false&Ntk=All&Ntt=red&Ntx=mode%2bmatchallpar¬
tial
```

Using the URL Optimization API, you can move Endeca parameters into the path-params section of the URL. For example, the following URL includes the N and Ntt parameters in the base URL:

```
http://localhost/ContentAssemblerRefApp/Content.aspx/Bordeaux/_/N-
4294966952/Ntt-red?fromsearch=false&Ntk=All&Ntx=mode%2bmatchallpartial
```

Encoding Endeca Parameters

In order to shorten URLs, the URL Optimization API allows base-36 encoding of Endeca parameters.

For example, the following URL for Vintage > 1996 contains the dimension value ID for 1996 (4294962059):

```
http://localhost/ContentAssemblerRefApp/Content.aspx/_/N-4294962059
```

By base-36 encoding the N parameter, you can shorten the URL:

```
http://localhost/ContentAssemblerRefApp/Content.aspx/_/N-1z13xxn
```

Related Links

About optimizing the misc-path on page 38

With the URL Optimization API you can configure dimensions, dimension values, record properties, and aggregate record properties to display in the misc-path of URLs. You can also specify the order in which dimensions and dimension values display on navigation pages.

About optimizing the path-params and query string on page 53
The URL Optimization API provides functionality for encoding path parameters and moving Endeca path parameters from the query string into the path-params section of the URL.

Configuring the path-param-separator on page 52
Using the `PathSeparatorToken` property in the `BaseSeoUrlProvider` constructor, you can configure the path-params-separator string.

Anatomy of an optimized Endeca URL on page 35
An optimized Endeca URL is made up of four configurable sections.

Duplicate content and URL canonicalization

Dynamic sites often produce syntactically different URLs for the same page. Multiple variant URLs result in duplicate content and therefore lower natural search engine ranking. Canonicalizing your URLs reduces that duplicate content and improves search engine ranking.

Many search engines base their relevancy ranking algorithms on the number and quality of links that point to a particular page. The more links there are that point to a particular page, the higher the page rank. Dynamic URLs can dilute the link value of a page by creating multiple versions of a URL.

For example, users might be able to reach the Napa Red wine page by first clicking on “Napa” and then clicking on “Red”, or by first clicking on “Red” and then “Napa.” This creates two syntactically unique links pointing to the same Napa Red page:

- `http://localhost/ContentAssemblerRefApp/Content.aspx/Wine-Red/Region-Na¬pa/_/N-1z141vcZ66t`
- `http://localhost/ContentAssemblerRefApp/Content.aspx/Region-Napa/Wine-Red/_/N-1z141vcZ66t`

To the search engine, each version of the URL appears to be its own unique page, and each page takes a portion of the link references.

To improve quality, search engines try to minimize the appearance of largely similar pages within results sets. Among other strategies, all indexed pages are evaluated for duplicates and near-duplicates before a page is selected to be displayed in the search results. In the case of the Napa Red page, only one of the two URLs would be selected -- and therefore only half of the link references are evaluated. This link dilution of the Napa Red page may result in a lower position within search results. Multiple parameters in URLs have the same effect.

In order to avoid multiple versions of URLs per page, links throughout the site should be standardized (canonicalized), and requests for a non-standard version of the URL should be redirected to the canonical version via a "301" (permanent) redirect.

The URL Optimization API provides functionality to canonicalize URLs. With canonicalization enabled, equivalent pages appear with the same syntax even if they are navigated to through different paths. For example, you can configure your `UrlProvider` to arrange dimensions alphabetically in an ascending order:

- `http://localhost/ContentAssemblerRefApp/Content.aspx/Region-Napa/Wine-Red/_/N-1z141vcZ66t`

Now even if a user navigates to "Red" before "Napa", the link still appears as "/Region-Napa/Wine-Red."

Note: There are a number of configuration options for URL canonicalization.

Related Links

About optimizing the misc-path on page 38
: With the URL Optimization API you can configure dimensions, dimension values, record properties, and aggregate record properties to display in the misc-path of URLs. You can also specify the order in which dimensions and dimension values display on navigation pages.

Chapter 3

Setting up a Reference Application

This section describes the sample `SeoUrlProvider` included with the URL Optimization API, and provides instructions for integrating it into an existing Endeca Content Assembler reference application.

Reference application prerequisites

Before integrating URL Optimization API with an existing Endeca Content Assembler reference application, you must have the reference application running on a properly configured MDEX Engine with the Endeca wine data set.

For information about setting up the Endeca Content Assembler sample wine data project, please refer to the *Oracle Endeca Experience Manager Getting Started Guide*.

Once you have the sample wine data project set up, ensure that the following dimensions are configured to **Show with record** and **Show with record list** in Developer Studio.

- Wine Type
- Region
- Winery
- Vintage
- Designation

Related Links

Preparing your dimensions on page 25
If you intend to display dimensions or dimension values in your URLs, you must configure each of the dimensions to **Show with record** and **Show with record list**.

Preparing your properties on page 25
If you intend to display record properties in your URLs, you must configure each property to **Show with record** and **Show with record list**.

About the sample UrlProvider

The sample `UrlProvider` (`Endeca.Web.Url.Seo.Sample.SeoUrlProvider`) included with the URL Optimization API is an `SeoUrlProvider` intended to demonstrate the capabilities to the URL Optimization API for the RAD Toolkit for ASP.NET.

The sample `SeoUrlProvider` is configured for the standard Endeca wine data set so that you integrate it into an Endeca Content Assembler reference application. This allows you to explore the functionality provided by the URL Optimization API.

While it is not the required approach, the sample `SeoUrlProvider` uses Factory methods to instantiate member variables of the `BaseSeoUrlProvider`.

Note: The sample `SeoUrlProvider` is provided for reference only and is not part of the supported software.

Integrating the URL Optimization API with the Content Assembler reference application

This section provides instructions for integrating the URL Optimization API with the Endeca Content Assembler reference application.

Important: You must install the RAD Toolkit for ASP.NET before implementing URL optimization.

This procedure assumes that you have already set up the Content Assembler reference application. For more information about the Content Assembler reference application, please refer to the *Oracle Endeca Experience Manager Getting Started Guide*.

To enable URL Optimization in your Content Assembler reference application:

1. Navigate to the `bin` subdirectory of your URL Optimization API installation directory.
   For example: `C:\Endeca\SEM\URLOptimizationAPIs\RAD Toolkit for ASP.NET\`*`version`*`\bin`
2. Copy the `Endeca.Web.Url.Seo.dll` and `Endeca.Web.Url.Seo.xml` to the `bin` subdirectory of your Content Assembler API reference application directory.
   For example: `C:\Endeca\ContentAssemblerAPIs\RAD Toolkit for ASP.NET\<version>\reference\ContentAssemblerRefApp\bin`
3. Navigate to the `bin` subdirectory of your RAD Toolkit for ASP.NET installation directory.
   For example: `C:\Endeca\RADToolkits\<`*`version`*`>\ASP.NET\bin`
4. Copy the following files to the `bin` subdirectory of your Content Assembler API reference application directory.
   - `Endeca.Data.dll`
   - `Endeca.Data.XML`
   - `Endeca.Web.dll`
   - `Endeca.Web.XML`
   - `Endeca.Web.UI.WebControls.dll`
   - `Endeca.Web.UI.WebControls.xml`
5. Navigate to the `samples\UrlProvider` subdirectory of your URL Optimization API installation directory.
   For example: `C:\Endeca\SEM\URLOptimizationAPIs\RAD Toolkit for ASP.NET\`*`version`*`\samples\UrlProvider`
6. Copy the sample `SeoUrlProvider.cs` to the `reference\ContentAssemblerRefApp\App_Code` subdirectory of your Content Assembler API installation directory.

For example: `C:\Endeca\ContentAssemblerAPIs\RAD Toolkit for ASP.NET\`*`version`*`\reference\ContentAssemblerRefApp\App_Code`

7. Navigate to the `reference\ContentAssemblerRefApp` subdirectory of your Content Assembler API installation and open the `Web.config`.
8. Locate the following section:

```
<endeca.web>
 <urlProvider defaultType="Endeca.Web.Url.BasicUrlProvider,Endeca.Web"
/>
</endeca.web>
```

9. Replace the `urlProvider defaultType`.

   For example:

```
<endeca.web>
  <urlProvider defaultType="Endeca.Web.Url.Seo.Sample.SeoUrlProvider,En¬
deca.Web.Url.Seo" />
</endeca.web>
```

10. Clear your Web browser cache. If you do not clear the cache, unoptimized URLs that have been cached may display as you navigate within the updated application.
11. Restart IIS.
12. (Optional) Open a Web browser and navigate to your Content Assembler reference application to verify the deployment: *http://localhost/ContentAssemblerRefApp/Content.aspx*.

    Replace *ContentAssemblerRefApp* with the name of the virtual directory in IIS.
13. Navigate to Wine Type > Red.

    The URL that displays should be similar to the following:

    `http://localhost/ContentAssemblerRefApp/Content.aspx/Wine-Red/_/N-66t`

    Note that keywords display in the misc-path of the URL.

Remember: If you choose to enable URL optimization in your Content Assembler reference application, you cannot use the same instance of the reference application as your preview application in Endeca Workbench.

Chapter 4

Implementing URL optimization

You can implement URL optimization with a new Endeca application or you can modify an existing application built with the Endeca RAD Toolkit for ASP.NET to create optimized URLs.

Implementing URL optimization with a new application

This section provides a high-level overview of the tasks required to implement URL optimization with a new Endeca application built using the RAD Toolkit for ASP.NET.

To implement URL optimization for a new Endeca application:

1. Ensure that you have installed the required Endeca software, including:
   - Platform Services
   - MDEX Engine
   - RAD Toolkit for ASP.NET
2. Ensure that any properties and dimensions that you want to display in optimized URLs are configured to **Show with record** and **Show with record list**.
3. Create multiple `UrlManager` instances to handle different commands and register your data sources with the appropriate `UrlManager`.

   The URL Optimization API does not support using certain combinations of commands with a single `UrlManager`. Therefore, you must use multiple `UrlManager` instances in your application.
4. When building search engine-optimized URLs, use the `UrlBuilder` convenience methods, such as `SelectDimensionValue` or `SelectRecord`. If you use the `UrlBuilder.BuildUrl` method directly, use `UrlBuilder.BuildUrl((Action(CommandActionInfo)), IIndexProvider, string)` rather than `BuildUrl(Action(CommandActionInfo))`.
5. Write a custom `UrlProvider` or, alternatively, modify the sample `SeoUrlProvider` included in this software distribution.

   The URL Optimization API implements serialization logic that enables you to easily create a custom `UrlProvider` to build search engine-optimized Endeca URLs. The sample `SeoUrlProvider` provides a good starting point for your implementation.

   Note: The URL Optimization API generates URLs that are compatible with the Endeca Sitemap Generator. It is also possible to write a `UrlProvider` using your own serialization logic, although in this case the Endeca Sitemap Generator can no longer create valid sitemaps for your application.

6. Integrate the custom `UrlProvider` into your application.

For more details about the `UrlManager` and `UrlProvider` classes, as well as the general tasks involved with building URLs with the RAD Toolkit for ASP.NET, please refer to the *RAD Toolkit for ASP.NET Developer's Guide*.

Related Links

System requirements on page 9
: This section provides a list of minimum system requirements for the Endeca URL Optimization API for the RAD Toolkit for ASP.NET.

Preparing your dimensions on page 25
: If you intend to display dimensions or dimension values in your URLs, you must configure each of the dimensions to **Show with record** and **Show with record list**.

Preparing your properties on page 25
: If you intend to display record properties in your URLs, you must configure each property to **Show with record** and **Show with record list**.

About using multiple UrlManagers on page 27
: In order to generate URLs with the URL Optimization API for the RAD Toolkit for ASP.NET, you may need to use multiple `UrlManager` instances.

About ensuring that URLs are optimized on page 30
: In cases where a `UrlManager` does not have enough information to generate an optimized URL, the URL Optimization API generates a URL that is correct, but not search-engine optimized.

Creating an SEO UrlProvider on page 36
: While there are a number of ways to configure URL optimization with the URL Optimization API, this guide documents the approach taken by the sample `SeoUrlProvider`. The following sections describe the procedure for creating and customizing a `UrlProvider` similar to the sample `SeoUrlProvider` included with the URL Optimization API.

Using the SeoUrlProvider with your application on page 59
: Once you have created and configured your custom `SeoUrlProvider`, you need to integrate the `UrlProvider` into you application.

Modifying an existing application built with the RAD Toolkit for ASP.NET

This section provides a high-level overview of the modifications you must make to an existing application built using the RAD Toolkit for ASP.NET in order to implement URL optimization.

This procedure assumes that your application uses URLs instead of postbacks to manage state transitions.

To modify an existing application to use the URL Optimization API:

1. Upgrade the RAD Toolkit for ASP.NET to the latest version.
2. Ensure that any properties and dimensions that you want to display in optimized URLs are configured to **Show with record** and **Show with record list**.
3. Convert your application to use multiple `UrlManager` instances to handle different commands and register your data sources with the appropriate `UrlManager`.

The URL Optimization API does not support using certain combinations of commands with a single `UrlManager`. Therefore, you must use multiple `UrlManager` instances in your application.

Note that the `MultipleRecordDetailsCommand`, which is used in the RAD Toolkit for ASP.NET reference application, is not supported by the URL Optimization API. If your application uses the `MultipleRecordDetailsCommand`, you must register it with a separate `UrlManager` that uses the `BasicUrlProvider` from the RAD Toolkit for ASP.NET. Alternatively, you can modify your application to use a `RecordDetailsCommand` to retrieve several `Record` objects sequentially.

4. When building search engine-optimized URLs, use the `UrlBuilder` convenience methods, such as `SelectDimensionValue` or `SelectRecord`. If you use the `UrlBuilder.BuildUrl` method directly, use `UrlBuilder.BuildUrl((Action(CommandActionInfo)), IIndexProvider, string)` rather than `BuildUrl(Action(CommandActionInfo))`.
5. Write a custom `UrlProvider` or, alternatively, modify the sample `SeoUrlProvider` included in this software distribution.

   The URL Optimization API implements serialization logic that enables you to easily create a custom `UrlProvider` to build search engine-optimized Endeca URLs. The sample `SeoUrlProvider` provides a good starting point for your implementation.

   Note: The URL Optimization API generates URLs that are compatible with the Endeca Sitemap Generator. It is also possible to write a `UrlProvider` using your own serialization logic, although in this case the Endeca Sitemap Generator can no longer create valid sitemaps for your application.

6. Integrate the custom `UrlProvider` into your application.

For more details about the `UrlManager` and `UrlProvider` classes, as well as the general tasks involved with building URLs with the RAD Toolkit for ASP.NET, please refer to the *RAD Toolkit for ASP.NET Developer's Guide*.

Related Links

System requirements on page 9
: This section provides a list of minimum system requirements for the Endeca URL Optimization API for the RAD Toolkit for ASP.NET.

Preparing your dimensions on page 25
: If you intend to display dimensions or dimension values in your URLs, you must configure each of the dimensions to **Show with record** and **Show with record list**.

Preparing your properties on page 25
: If you intend to display record properties in your URLs, you must configure each property to **Show with record** and **Show with record list**.

About using multiple UrlManagers on page 27
: In order to generate URLs with the URL Optimization API for the RAD Toolkit for ASP.NET, you may need to use multiple `UrlManager` instances.

About ensuring that URLs are optimized on page 30
: In cases where a `UrlManager` does not have enough information to generate an optimized URL, the URL Optimization API generates a URL that is correct, but not search-engine optimized.

Creating an SEO UrlProvider on page 36
: While there are a number of ways to configure URL optimization with the URL Optimization API, this guide documents the approach taken by the sample `SeoUrlProvider`. The following

sections describe the procedure for creating and customizing a `UrlProvider` similar to the sample `SeoUrlProvider` included with the URL Optimization API.

Using the SeoUrlProvider with your application on page 59
Once you have created and configured your custom `SeoUrlProvider`, you need to integrate the `UrlProvider` into you application.

Chapter 5

Preparing your application

This section describes the basic requirements and recommendations for writing your application.

Preparing your dimensions

If you intend to display dimensions or dimension values in your URLs, you must configure each of the dimensions to **Show with record** and **Show with record list**.

You only need to configure the dimensions you intend to include in URLs. Configuring all dimensions to **Show with record** and **Show with record list** may have performance implications.

To configure a dimension to **Show with record** and **Show with record list**:

1. Open your project in Endeca Developer Studio.
2. From the **Project Explorer** on the left, click Dimensions.
   The **Dimensions** dialog displays.
3. Select the dimension you need to edit.
4. Select the **Show with record list** checkbox.
5. Select the **Show with record** checkbox.
6. Click **OK**.
7. Save your changes.

For more information, please refer to the *Oracle Endeca Developer Studio Help*.

Preparing your properties

If you intend to display record properties in your URLs, you must configure each property to **Show with record** and **Show with record list**.

You only need to configure the properties you intend to include in URLs. Configuring all properties to **Show with record** and **Show with record list** may have performance implications.

To configure a property to **Show with record** and **Show with record list**:

1. Open your project in Endeca Developer Studio.
2. From the **Project Explorer** on the left, click Dimensions.
   The **Dimensions** dialog displays.

3. Select the dimension you need to edit.
4. Select the **Show with record list** checkbox.
5. Select the **Show with record** checkbox.
6. Click **OK**.
7. Save your changes.

For more information, please refer to the *Oracle Endeca Developer Studio Help*.

Handling images and external JavaScript files in optimized URLs

When you modify your application to produce optimized URLs, it is important to ensure that the server can still locate resources requested by the application, such as image files, JavaScript files, and CSS files.

Relative URLs are partial URLs that omit host and port information. There are two types of relative URLs:

- "Site-relative" URLs are relative to the root directory on the site that hosts the Web page, for example: `/sitemap.htm`
- "Non-site-relative" URLs are relative to their parent pages, for example: `../sitemap.htm`

Because relative paths are relative to the URL that is requested, not the URL that is ultimately resolved, optimized URLs may create unresolved links when external resources are referenced. When using the URL Optimization API, Endeca recommends replacing non-site-relative URLs with site-relative URLs to ensure that links resolve properly.

URL transitioning

Managing redirects is an important aspect of search engine optimization. In order to maintain page rank for resources within your website, you need an effective strategy to manage URL changes.

As you transition from traditional Endeca URLs to optimized Endeca URLs, or when you change the configuration of optimized URLs, it is important to ensure that:

- Links throughout your Web site are updated
- Links to external resources (such as image files, CSS, or Javascript files) are updated
- External links to your Web site are permanently redirected to the new URLs

Links throughout your own Web site and to your own external resources can simply be updated to the new URLs. However, external references to your site must be redirected in order to prevent unresolved links.

The URL Optimization API is responsible for transforming URLs into Endeca search and navigation queries, and vice-versa. It does not implement redirect logic. In order to redirect incoming requests, you must include the appropriate logic in your application controller. By comparing an inbound URL to the canonical (optimized) form, you can redirect to the canonical URL in cases where the inbound URL is different.

Oracle recommends including HTTP 301 redirects. Unlike HTTP 302 redirects, which collect ranking information and index content on a site against the source URL, 301 redirects apply this information to the destination URL.

Chapter 6

Building URLs with the URL Optimization API

The URL Optimization API leverages the core functionality of the RAD Toolkit for ASP.NET in order to build search engine-optimized URLs. However, there are some important differences in how you use the API in order to ensure that the URLs in your application are properly optimized.

About using multiple UrlManagers

In order to generate URLs with the URL Optimization API for the RAD Toolkit for ASP.NET, you may need to use multiple `UrlManager` instances.

The URL Optimization API does not support using certain combinations of commands with a single `UrlManager`. If you register any of the unsupported combinations with the same instance of a `Url¬Manager`, the URL Optimization API throws an exception. Therefore, depending on the kinds of queries in your application, you may need the following distinct `UrlManager` instances:

- One to handle `NavigationCommand`, `DimensionSearchCommand`, `CompoundDimension¬SearchCommand`, and `MetadataCommand`
- One to handle `RecordDetailsCommand`
- One to handle `AggregateRecordDetailsCommand`

Although you must have separate `UrlManager` instances, they can all use the same `UrlProvider` to serialize the command state into search engine-optimized URLs.

Note that the URL Optimization API does not support the `MultipleRecordDetailsCommand`. If you need to create URLs for a `MultipleRecordDetailsCommand`, you need to register it with its own `UrlManager` that uses the `BasicUrlProvider` to serialize URLs. In this case, the URLs for the `MultipleRecordDetailsCommand` are not search-engine optimized.

Related Links

Invalid command combinations on page 29
If you register any of the following combinations with the same instance of a `UrlManager`, the URL Optimization API throws an exception.

Working with multiple UrlManagers

The URL Optimization API requires the use of multiple `UrlManager` instances in order to properly generate links for all the pages in your application.

This section assumes that you are familiar with building URLs using the RAD Toolkit for ASP.NET. For more details about working with the `UrlManager` class, please refer to the *RAD Toolkit for ASP.NET Developer's Guide*.

In this example, one `UrlManager` is used to register a `NavigationDataSource` and `Dimension¬SearchDataSource`, while a separate `UrlManager` is used to register a `RecordDetailsData¬Source`. It assumes you have already created data sources or commands in your application and assigned them an ID, as in the following:

```
    <end:NavigationDataSource
      ID="dsNav"
      runat="server"
      MdexHostName="<%$ Snippet:_mdex.MdexHostName %>"
      MdexPort="<%$ Snippet:_mdex.MdexPort %>">
    </end:NavigationDataSource>

    <end:DimensionSearchDataSource
      ID="dsDimensionSearch"
      runat="server"
      MdexHostName="<%$ Snippet:_mdex.MdexHostName %>"
      MdexPort="<%$ Snippet:_mdex.MdexPort %>">
    </end:DimensionSearchDataSource>

    <end:RecordDetailsDataSource
      ID="dsRecordDetails"
      runat="server"
      MdexHostName="<%$ Snippet:_mdex.MdexHostName %>"
      MdexPort="<%$ Snippet:_mdex.MdexPort %>">
    </end:RecordDetailsDataSource>
```

1. Register the `NavigationDataSource` and `DimensionSearchDataSource` with the `UrlMan¬ager` named "Default."

   ```
   UrlManager.Default.Register(dsNav);
   UrlManager.Default.BaseUrl = "~/Content.aspx";
   UrlManager.Default.Register(dsDimensionSearch);
   UrlManager.Default.InitializeFrom(Request.Url);
   ```

2. Create another `UrlManager` named "RecordDetail" and register the `RecordDetailsDataSource` with it.

   ```
   UrlManager recordDetailUrlManager = UrlManager.Get("RecordDetail");
   recordDetailUrlManager.Register(dsRecordDetails);
   recordDetailUrlManager.BaseUrl = "~/Content.aspx";
   recordDetailUrlManager.InitializeFrom(Request.Url);
   ```

3. When generating links, be sure to use the correct `UrlManager` instance. For example, when creating the URL for a record detail page, use the `recordDetailUrlManager`.

   ```
   UrlManager recordDetailUrlManager = UrlManager.Get("RecordDetail");
   String url = recordDetailUrlManager.Urls.SelectRecord(RecordDetailsData¬
   SourceID, (Record)record)
   ```

Working with multiple UrlProviders

You can specify more than one `UrlProvider` in your application, for example if you need to generate URLs for commands that are not supported by the URL Optimization API.

Each `UrlManager` uses only one `UrlProvider`. However, if you use multiple `UrlManager` instances, you can specify a different `UrlProvider` for each one. In general, you want to use a single `Url¬Provider` to serialize all the URLs in your application.

You specify the assembly-qualified type name for the `UrlProvider` classes in the application's `Web.config` file. In this example, all instances of `UrlManager` use the `SeoUrlProvider` except for the `UrlManager` designed to handle `MultipleRecordDetailsCommand`, which is not supported by the URL Optimization API.

```
<endeca.web>
  <urlProviders>
    <add urlManagerId="Default" type="Endeca.Web.Url.Seo.Sample.SeoUrl¬
Provider,Endeca.Web.Url.Seo" />
   <add urlManagerId="RecordDetail" type="Endeca.Web.Url.Seo.Sample.SeoUrl¬
Provider,Endeca.Web.Url.Seo" />
    <add urlManagerId="AggrRecordDetail" type="Endeca.Web.Url.Seo.Sam¬
ple.SeoUrlProvider,Endeca.Web.Url.Seo" />
    <add urlManagerId="MultiRecordDetail" type="Endeca.Web.Url.BasicUrl¬
Provider,Endeca.Web" />
  </urlProviders>
</endeca.web>
```

You can also set the `UrlProvider` on the `UrlManager` programmatically, for example:

```
UrlManager multiRecordDetailUrlManager = UrlManager.Get("MultiRecordDetail");
multiRecordDetailUrlManager.UrlProvider = new BasicUrlProvider();
```

Invalid command combinations

If you register any of the following combinations with the same instance of a `UrlManager`, the URL Optimization API throws an exception.

The URL Optimization API requires the use of multiple `UrlManager` instances in order to properly generate links for all the pages in your application.

Command	Invalid combination
`NavigationCommand`	`RecordDetailsCommand`
`NavigationCommand`	`AggregateRecordDetailsCommand`
`RecordDetailsCommand`	`AggregateRecordDetailsCommand`
`RecordDetailsCommand`	`DimensionSearchCommand`
`RecordDetailsCommand`	`CompoundDimensionSearchCommand`
`RecordDetailsCommand`	`MetadataCommand`
`AggregateRecordDetailsCommand`	`DimensionSearchCommand`
`AggregateRecordDetailsCommand`	`CompoundDimensionSearchCommand`
`AggregateRecordDetailsCommand`	`MetadataCommand`

Note: The `MultipleRecordDetailsCommand` is not supported by the URL Optimization API.

Related Links

Working with multiple UrlManagers on page 27

The URL Optimization API requires the use of multiple `UrlManager` instances in order to properly generate links for all the pages in your application.

About avoiding invalid URLs

Using the URL Optimization API makes it possible to inadvertently create URLs that exceed the MAX_PATH limit of the .NET Framework.

The .NET Framework imposes a limit of 260 characters on the physical path in URLs. Any attempt to access a link that violates this limit returns an HTTP 400 (Bad Request) code. This error is logged in the IIS logs along with the URL that was requested.

The URL Optimization API may produce long URLs in cases such as the following:

- many dimensions set to display as part of the misc-path
- deep dimensions set to display the full hierarchy as part of the misc-path
- very long dimension names set to display as part of the misc-path

In order to ensure that end users do not encounter errors while browsing your site, you should wrap any calls to `BuildUrl` or the `UrlBuilder` helper methods with logic that checks the length of the path and shortens the misc-path if necessary. Note that the protocol, server information, application name, and query string are not included in the 260-character limit.

About ensuring that URLs are optimized

In cases where a `UrlManager` does not have enough information to generate an optimized URL, the URL Optimization API generates a URL that is correct, but not search-engine optimized.

The `UrlBuilder` class provides a number of convenience methods that can be used to perform common actions on a command and generate the resulting URL string. In the vast majority of cases, these convenience methods guarantee that you are passing in enough information to the `UrlManager` to build an optimized URL. Therefore, for common use cases, it is recommended that you use these convenience methods to generate URLs.

For example, the following call to `UrlBuilder` generates a link to select a dimension value:

```
string url = UrlManager.Default.Urls.SelectDimensionValue(
  NavigationDataSourceID, dval);
```

where `NavigationDataSourceID` is a string containing the ID of the `NavigationDataSource`, and `dval` is the `DimensionValue` object to select.

The `UrlBuilder` class is part of the Endeca RAD API. For further information about using the `Url¬Builder` helper methods, please refer to the *RAD Toolkit for ASP.NET Developer's Guide* and the *API Reference for the RAD Toolkit for ASP.NET*.

The following sections describe certain situations in which the convenience methods do not always ensure that the `UrlManager` can generate an optimized URL.

Building an optimized URL to select a dimension value from a BusinessRule object

`BusinessRule` objects store their navigation state as a collection of `DisconnectedDimensionValue` objects, which do not contain enough information for the `UrlManager` to build an optimized URL.

Calling `UrlBuilder.SelectDimensionValue` with a `DisconnectedDimensionValue` results in a functional but non-optimized URL. When selecting dimension values returned with a `BusinessRule` object, you must use the `UrlBuilder.SelectDisconnectedDimensionValue` method to build optimized URLs.

For example:

```
string url = UrlManager.Default.Urls.SelectDisconnectedDimensionValue(
  NavigationDataSourceID, disconnectedDimensionValue, record);
```

where `NavigationDataSourceID` is a string containing the ID of the `NavigationDataSource`, `disconnectedDimensionValue` is the `DisconnectedDimensionValue` object to select, and `record` is a `Record` object that is returned with the business rule that fired. This `Record` object contains a `DimensionValue` object that corresponds to the `DisconnectedDimensionValue`, and that `DimensionValue` object is used to generate an optimized URL.

Building an optimized URL to an aggregate record detail page

Using the URL Optimization API affects the way that you build URLs to select an aggregate record.

Two pieces of information are needed to create a URL for an aggregate record detail page: The ID of the aggregate record to be selected and the aggregation key. The aggregation key is usually present on the `NavigationDataSource` representing a navigation state that includes the aggregate records.

The `UrlBuilder` class exposes several convenience methods that can be used to select an aggregate record. The `SelectAggregateRecord(AggregateRecordDetailsDataSourceID, NavigationDataSourceID, aggRec)` method assumes that your `NavigationDataSource` and `AggregateRecordDetailDataSource` are registered with the same URL manager. However, with the URL Optimization API, you are required to use separate instances of `UrlManager` to handle the `NavigationDataSource` and `AggregateRecordDetailDataSource`, and you should use the following convenience method to build URLs for aggregate records:

```
string url = UrlManager.Default.Urls.SelectAggregateRecord(
  AggregateRecordDetailsDataSourceID, navigationDataSource, aggRec);
```

where `AggregateRecordDetailsDataSourceID` is the string containing the ID of the `AggregateRecordDetailsDataSource`, `navigationDataSource` is the `NavigationDataSource` object containing the appropriate aggregation key, and `aggRec` is the `AggregateRecord` object to select.

An alternate approach is to select an aggregate record by setting the aggregation key directly on the `AggregateRecordDetailsDataSource`, as in the following example:

```
// Find both your aggregate record details data source
// and the navigation data source containing the appropriate
// aggregation key.
AggregateRecordDetailsDataSource aggrRecDataSource =
  BindingUtility.FindDataSource<AggregateRecordDetailsDataSource>(
  "dsAggregateRecordDetails", this);
NavigationDataSource navDS =
  BindingUtility.FindDataSource<NavigationDataSource>("dsNav", this);
// set the aggregation key on the aggregate record details data
// source to be that on the navigation data source
aggrRecDataSource.AggregationKey = navDS.AggregationKey;

// call the UrlBuilder.SelectAggregateRecord method with the
// aggregate record detail data source id and the AggregateRecord
// object to select.
string url = UrlManager.Default.Urls.SelectAggregateRecord(
  AggregateRecordDetailsDataSourceID, aggRec);
```

This is equivalent to calling the `UrlBuilder.BuildUrl` method as follows:

```
  string url = aggrRecordDetailUrlManager.Urls.BuildUrl(
    delegate (CommandActionInfo commands)
    {
      AggregateRecordDetailsCommand aggCommand =
        commands.Get<AggregateRecordDetailsCommand>(
        AggregateRecordDetailsDataSourceID);
      aggCommand.Identifier = aggRec.Id;
      aggCommand.AggregationKey = navDS.AggregationKey;
    }, new IndexedAggregateRecord(aggRec), AggregateRecordDetailsDataSour¬
ceID);
```

Building optimized URLs without using convenience methods

In situations for which the `UrlBuilder` class does not provide convenience methods, you must take care to pass sufficient information to the `BuildUrl` method to generate optimized URLs.

For example, there is no convenience method to copy the state of one `NavigationCommand` to another, as when building a "see-all" link for a dynamic spotlight cartridge in the Content Assembler API.

In a typical Content Assembler application (without URL optimization), you would create the link using the `BuildUrl` method as follows:

```
  UrlManager.Default.Urls.BuildUrl(
    delegate(CommandActionInfo commands)
    {
      NavigationCommand navCmd =
        commands.Get<NavigationCommand>(NavigationDataSourceID);
      navCmd.CopyFrom(spotlightNavCommand);
    });
```

where `spotlightNavCommand` is the `NavigationCommand` object returned by the Content Assembler as part of the `IRecordListProperty` object that represents the spotlighting results. However, because the `UrlManager` relies on its data sources to provide the information to create optimized URLs, and the spotlighting results typically are not associated with any data source in the application, this method generates a non-optimized URL.

To provide the `UrlManager` with enough information to create an optimized URL, first retrieve the `NavigationResult` object from the `IRecordListProperty` object and use it to create an `IInd¬exProvider` object. Then pass in this `IIndexProvider` with the call to `UrlBuilder.BuildUrl` as in the following example:

```
  UrlManager.Default.Urls.BuildUrl(
    delegate(CommandActionInfo commands)
    {
      NavigationCommand navCmd =
        commands.Get<NavigationCommand>(NavigationDataSourceID);
      navCmd.CopyFrom(spotlightNavCommand);
    }, new IndexedNavigationResult(spotlightNavResult), NavigationDataSour¬
ceID);
```

This call to `BuildUrl` uses the dimension values in this indexed result object to create an optimized URL representing the navigation state associated with the spotlight cartridge.

The same general process applies to any use case for which a convenience method does not already exist in `UrlBuilder`. When constructing the `IIndexProvider`, ensure that the object you index has all the information necessary to create an optimized URL. This may vary depending on the information you want to display in the URL and the kind of link you are constructing. For example, when building a URL to select a dimension value, make sure that the object that you pass to the `IIndexProvider` constructor contains that dimension value.

Chapter 7

Configuring URLs

This section provides information about creating and using a `UrlProvider` similiar to the `SeoUrl¬Provider` included with the URL Optimization API to optimize URLs in applications built with the Endeca RAD Toolkit for ASP.NET. The information and examples provided in this section relate to basic URL configuration tasks, and do not cover the entire breadth of the URL Optimization API capabilities. Endeca recommends consulting the API documentation as you develop your application.

Anatomy of an optimized Endeca URL

An optimized Endeca URL is made up of four configurable sections.

General URL References

When referring to URLs in general, the URL Optimization API documentation may use the terms "base URL" and "URL query parameters." The "base URL" is the part of the URL that precedes the question mark.

For example, in the URL:

```
http://www.example.com/pathparam1/pathparam2/pathparam3/results?queryparam=123
```

the base URL is the string that displays before the question mark:

```
http://www.example.com/pathparam1/pathparam2/pathparam3/results
```

Optimized Endeca URLs

For reference purposes, the documentation identifies four distinct sections of optimized Endeca URLs:

- misc-path
- path-param-separator
- path-params
- query string

For example, the following URL is broken down into subsections:

```
http://localhost:8888/controller[/Wine-Red-Merlot/Napa/Pine-Ridge/_/N-12ZafZfd?Ne=123]
```

The sections of the URL encased in square brackets can be broken down into the following components:

```
[/<misc-path>][/<path-param-separator>][/<path-params>][?<query-string>]
```

The components correspond to the following strings:

Section	String
misc-path	Wine-Red-Merlot/Napa/Pine-Ridge
path-param-separator	_
path-params	N-12ZafZfd
query string	Ne=123

misc-path

This section of the URL incorporates keywords into the URL in order to create user-friendly and search engine-optimized URLs. The misc-path section of the optimized URL can be generated based on dimension names, dimension values, ancestor names, and record properties. The misc-path component is largely ignored by the application.

path-param-separator

The path-param-separator component is used to identify the end of the misc-path and the starting point for path parameters. This string is configurable.

path-params

Together with the query string, the path-params segment of the URL represents the current state of the application. This may include the numerical representation of the navigation state or a specific record, as well as any other parameter key-value pairs that have an effect on the displayed content. This component can be configured to contain several parameters that would typically be included as part of the query string in traditional Endeca URLs, such as the N, Ne, Ntt, and R parameters.

query string

The query string component of the URL follows the question mark character. The combination of the path-params and query string represents the current state of the application. Endeca parameters such as N, Ne, Ntt, and R that are not configured to display in the path-params section of the URL display in the query string.

Related Links

About optimizing the misc-path on page 38
With the URL Optimization API you can configure dimensions, dimension values, record properties, and aggregate record properties to display in the misc-path of URLs. You can also specify the order in which dimensions and dimension values display on navigation pages.

About optimizing the path-params and query string on page 53
The URL Optimization API provides functionality for encoding path parameters and moving Endeca path parameters from the query string into the path-params section of the URL.

Configuring the path-param-separator on page 52
Using the `PathSeparatorToken` property in the `BaseSeoUrlProvider` constructor, you can configure the path-params-separator string.

Creating an SEO UrlProvider

While there are a number of ways to configure URL optimization with the URL Optimization API, this guide documents the approach taken by the sample `SeoUrlProvider`. The following sections describe the procedure for creating and customizing a `UrlProvider` similar to the sample `SeoUrlProvider` included with the URL Optimization API.

Note: The procedures documented in this section are based on the sample `SeoUrlProvider` which uses Factory methods to instantiate member variables derived from the `BaseSeoUrlProvider` parent class. These Factory classes are not part of the URL Optimization API, but provide a simple and flexible mechanism for instantiating objects and organizing code for reuse. You are not required to use Factory classes to instantiate objects.

To create an `SeoUrlProvider`:

1. Include the following libraries in your `SeoUrlProvider`:
```
using Endeca.Web.Url.Seo.Formatting;
using System.Collections.ObjectModel;
using System.Collections.Specialized;
using System;
using System.Collections.Generic;
using Endeca.Web.Url.Seo.Utility;
using Endeca.Web.Url.Seo.Canonicalizers;
using Endeca.Data;
```
2. Create a namespace for your `SeoUrlProvider` class.
For example, the sample `SeoUrlProvider` included with the URL Optimization API uses the following namespace:
```
namespace Endeca.Web.Url.Seo.Sample
```
3. Create an `SeoUrlProvider` class that inherits from the `BaseSeoUrlProvider` class:
For example:
```
   public class SeoUrlProvider : BaseSeoUrlProvider
   {
   }
```

In order to begin optimizing URLs, you need to set certain required properties in the constructor of the `SeoUrlProvider` object. For example:

```
   public class SeoUrlProvider : BaseSeoUrlProvider
   {

      public SeoUrlProvider()
         : base()
      {
        this.NavigationCommandFormatter = NavigationCommandFormatterFacto¬
ry.Create();
        this.RecordDetailsCommandFormatter = RecordDetailsCommandFormatter¬
Factory.Create();
         this.AggregateRecordDetailsCommandFormatter = AggregateRecordDe¬
tailsCommandFormatterFactory.Create();
        this.NavigationCommandCanonicalizer = DimensionValueCollectionCanon¬
icalizerFactory.Create();
         this.ParameterEncoders = ParameterEncodersFactory.Create();
         this.PathParameters = PathParameterFactory.Create();
      }


   }
```

The instructions in this chapter explain which of the derived properties are required for each task. You can set these properties on your `SeoUrlProvider` object as you work through the chapter.

About optimizing the misc-path

With the URL Optimization API you can configure dimensions, dimension values, record properties, and aggregate record properties to display in the misc-path of URLs. You can also specify the order in which dimensions and dimension values display on navigation pages.

NavigationCommandFormatter

The `NavigationCommandFormatter` is responsible for configuring dimension names, roots, ancestors, and dimension value names in the misc-path of URLs for navigation pages. `NavigationCommandFormatters` use `DimensionValuePathFormatters` to format the individual dimension and dimension values to be displayed in the URL.

For example, the following URL is for the navigation state Region > Napa:

```
http://localhost/ContentAssemblerRefApp/Content.aspx/?&Ne=8&N=4294967160
```

Using URL Optimization API, that same URL can be formatted as follows:

```
http://localhost/ContentAssemblerRefApp/Content.aspx/Napa/_/N-1z141vc
```

NavigationCommandCanonicalizer

The `NavigationCommandCanonicalizer` is responsible for ordering the dimension and dimension value names included in the misc-path for navigation pages. For example, an end-user can reach the Wine Type > Red, Region > Napa page by navigating first to Wine Type > Red and then to Region > Napa, or by navigating to Region > Napa and then Wine Type > Red. To avoid two syntactically different URLs for the same Wine Type > Red, Region > Napa page, you can use the `NavigationCommandCanonicalizer` to standardize the order of dimension and dimension values in the misc-path.

Note: There are a number of configuration options available for the arrangement of the dimension and dimension value names.

RecordDetailsCommandFormatter

URL optimization for record detail pages is configured separately from navigation pages and aggregate record detail pages. In addition to dimension names, roots, ancestors, and dimension value names, you can also include record properties in the URL string for record detail pages.

AggregateRecordDetailsCommandFormatter

URL optimization for aggregate record detail pages is configured separately from navigation pages and record detail pages. In addition to dimension names, roots, ancestors, and dimension value names, you can also include record properties in the URL string for aggregate record detail pages.

Related Links

Duplicate content and URL canonicalization on page 15

Dynamic sites often produce syntactically different URLs for the same page. Multiple variant URLs result in duplicate content and therefore lower natural search engine ranking. Canonicalizing your URLs reduces that duplicate content and improves search engine ranking.

Anatomy of an optimized Endeca URL on page 35
An optimized Endeca URL is made up of four configurable sections.

Optimizing URLs for navigation pages

Using the URL Optimization API, you can include dimension and dimension value names in the misc-path of URLs. Dimension and dimension value names, also referred to as "keywords," are canonicalized to prevent duplicate content.

Note: For dimensions to display properly in the URL, they must be configured to **show with record** and **show with record list**.

Before you begin optimizing URLs, create an `SeoUrlProvider`.

To optimize URLs for navigation pages:

1. Set the `NavigationCommandFormatter` and `NavigationCommandCanonicalizer` properties in the constructor of the `SeoUrlProvider` object:

```
      public SeoUrlProvider()
         : base()
      {
        this.NavigationCommandFormatter = NavigationCommandFormatterFac¬
tory.Create();
         this.NavigationCommandCanonicalizer = DimensionValueCollection¬
CanonicalizerFactory.Create();
      }
```

2. Use the `NavigationCommandFormatterFactory` to instantiate a `NavigationCommandFor¬matter` object.
For example:

```
   public static class NavigationCommandFormatterFactory
   {
      public static NavigationCommandFormatter Create()
      {
        NavigationCommandFormatter formatter = new NavigationCommandFor¬
matter();
      }

   }
```

3. Set a `UseDimensionNameAsKey` property on the `NavigationCommandFormatter` object.
For example:

```
   public static class NavigationCommandFormatterFactory
   {
      public static NavigationCommandFormatter Create()
      {
        NavigationCommandFormatter formatter = new NavigationCommandFor¬
matter();
         formatter.UseDimensionNameAsKey = true;
      }

   }
```

Setting the `useDimensionNameAsKey` to `false` creates a key on the dimension ID numbers.

4. Set a `DimensionValuePathFormatters` property `NavigationCommandFormatter` object.

For example:

```
   public static class NavigationCommandFormatterFactory
   {
      public static NavigationCommandFormatter Create()
      {
        NavigationCommandFormatter formatter = new NavigationCommandFor¬
matter();
         formatter.UseDimensionNameAsKey = true;
         formatter.DimensionValuePathFormatters = DimensionValuePathFor¬
mattersFactory.CreateForNavigationCommand();
         return formatter;
      }

   }
```

5. Use a `DimensionValuePathFormattersFactory` to instantiate a `DimensionValuePathFor¬matters` object for the `NavigationCommandFormatter`.
   For example:

```
   public static class DimensionValuePathFormattersFactory
   {
      public static Dictionary<string, IDimensionValuePathFormatter>
CreateForNavigationCommand()
      {
         Dictionary<string, IDimensionValuePathFormatter> dimValPaths =
             new Dictionary<string, IDimensionValuePathFormatter>();

      }
```

6. Add a `DimensionValuePathFormatter` for each dimension that you intend to include in the `DimensionValuePathFormattersFactory`.

   Note: Use the `dimValPaths.Add` method call to key each `DimensionValuePathFor¬matter`. If you set the `useDimensionNameAsKeyDimensionValuePathFormatter` as `false` in step three, be sure to use the dimension ID instead of the dimension name.

   For example:

```
      public static Dictionary<string, IDimensionValuePathFormatter>
CreateForNavigationCommand()
      {
         Dictionary<string, IDimensionValuePathFormatter> dimValPaths =
             new Dictionary<string, IDimensionValuePathFormatter>();

         DimensionValuePathFormatter wineTypeFormatter = DimensionVal¬
uePathFormatterFactory.Create(true, true, true);
        DimensionValuePathFormatter regionFormatter = DimensionValuePath¬
FormatterFactory.Create(false, false, true);
        DimensionValuePathFormatter wineryFormatter = DimensionValuePath¬
FormatterFactory.Create(false, false, true);

         dimValPaths.Add("Wine Type", wineTypeFormatter);
         dimValPaths.Add("Region", regionFormatter);
         dimValPaths.Add("Winery", wineryFormatter);

      }
```

   Note: The `DimensionValuePathFormatterFactory.Create(bool, bool, bool);` method is defined in step seven.

7. Use a `DimensionValuePathFormatterFactory` to instantiate a `DimensionValuePathFor¬matter` and add the following formatters:

Formatter	Description
AppendRoot	Specifies whether or not to append root dimension values to the URL. Set to `true` to append root dimension values.
AppendAncestors	Specifies whether or not to append ancestor dimension values to the URL. Set to `true` to append ancestor dimension values.
AppendDimensionVal¬ue	Specifies whether or not to append the selected or descriptor dimension values to the URL. Set to `true` to append ancestor dimension values.
RootFormatter	Sets the `IStringFormatter` to customize URL formatting for dimension roots in the URL. If the value is null, roots do not display in the URL.
PathFormatter	Sets the `IStringFormatter` to customize URL formatting for dimension, ancestor, and dimension value names. If the value is null, dimension, ancestor, and dimension value names do not display in the URL.

   For example:

```
   public static class DimensionValuePathFormatterFactory
   {
      public static DimensionValuePathFormatter Create(bool appendRoot,
 bool appendAncestors, bool appendDimVals)
      {
         DimensionValuePathFormatter formatter = new DimensionValuePath¬
Formatter();
         formatter.AppendRoot = appendRoot;
         formatter.AppendAncestors = appendAncestors;
         formatter.AppendDimensionValue = appendDimVals;

         formatter.RootFormatter = StringFormatterFactory.CreateDefault¬
StringFormatter();
         formatter.PathFormatter = StringFormatterFactory.CreateDefault¬
StringFormatter();

         return formatter;

      }

   }
```

8. Use the `StringFormatterFactory` to instantiate an `IStringFormatter` object.
   For example:

```
   public static class StringFormatterFactory
   {
      public static IStringFormatter CreateDefaultStringFormatter()
      {
```

9. Instantiate a `RegexReplacementStringFormatter` to replace non-word characters.

```
   public static class StringFormatterFactory
   {
      public static IStringFormatter CreateDefaultStringFormatter()
      {
         // Convert non word characters to '-'
         RegexReplacementStringFormatter nonWordCharsToDashFormatter
           //character class subtraction, whitespace character excluding
 the unicode characters
              = new RegexReplacementStringFormatter(@"(?:[\W-[\u00C0-
\u00FF]])+", "-");

      }
```

10. Instantiate a second `RegexReplacementStringFormatter` to remove leading and trailing characters created by the `RegexReplacementStringFormatter nonWordCharsToDashFormatter`.

```
   public static class StringFormatterFactory
   {
      public static IStringFormatter CreateDefaultStringFormatter()
      {
         // Convert non word characters to '-'
         RegexReplacementStringFormatter nonWordCharsToDashFormatter
           //character class subtraction, whitespace character excluding
 the unicode characters
              = new RegexReplacementStringFormatter(@"(?:[\W-[\u00C0-
\u00FF]])+", "-");

         // Trim leading and trailing '-' characters
         RegexReplacementStringFormatter trimDashCharsFormatter
            = new RegexReplacementStringFormatter(@"^-?([\w\u00C0-
\u00FF][\w-\u00C0-\u00FF]*[\w\u00C0-\u00FF])-?$", "$1");

        return new CompositeStringFormatter(nonWordCharsToDashFormatter,
 trimDashCharsFormatter);

      }
```

11. Use a `DimensionValueCollectionCanonicalizerFactory` to instantiate an `IDimensionValueCollectionCanonicalizer`.
    For example:

```
public static IDimensionValueCollectionCanonicalizer Create()
      {
        DimensionValueCollectionCanonicalizer canonicalizer = new Dimen¬
sionValueCollectionCanonicalizer();
        canonicalizer.Identifier = DimensionValueCollectionCanonicaliz¬
er.CanonicalIdentity.DimensionName;
         canonicalizer.Direction = SortDirection.Descending;

         return canonicalizer;

      }
```

Note: By design, the URL Optimization API prevents the creation of syntactically different URLs by canonicalizing keywords. You can choose from a number of configuration options to control the arrangement of canonicalized keywords.

Related Links

Preparing your dimensions on page 25
: If you intend to display dimensions or dimension values in your URLs, you must configure each of the dimensions to **Show with record** and **Show with record list**.

Preparing your properties on page 25

If you intend to display record properties in your URLs, you must configure each property to **Show with record** and **Show with record list**.

Canonicalization configuration options

The `canonicalizer.Identifier` and `canonicalizer.Direction` properties on the `IDimen¬sionValueCollectionCanonicalizer` object determine the method and direction of dimension ordering for navigation page URLs.

The `canonicalizer.Identifier` property determines how the dimensions should be sorted. The `canonicalizer.Direction` property determines in which direction to sort them.

For example, the following code sample creates a canonicalized URL that sorts by dimension ID in an ascending order:

```
   public static class DimensionValueCollectionCanonicalizerFactory
   {
      public static IDimensionValueCollectionCanonicalizer Create()
      {
         DimensionValueCollectionCanonicalizer canonicalizer = new Dimen¬
sionValueCollectionCanonicalizer();
         canonicalizer.Identifier = DimensionValueCollectionCanonicaliz¬
er.CanonicalIdentity.DimensionValueName;
         canonicalizer.Direction = SortDirection.Descending;

         return canonicalizer;

      }

   }
```

The following example configurations use the dimensions:

- Wine Type (dimension ID: 6200)
- region (dimension ID: 8)

and the dimension values:

- Red (dimension value ID: 8021)
- napa (dimension value ID: 4294967160)

Canonical Identity

`Canonical Identity value`	Description	Example base URL (sort direction ascending)
`DimensionName`	Sorts alphabetically by dimension name. Capital letters are given precedence.	`http://localhost/Con¬tentAssemblerRefApp/Con¬tent.aspx/Wine-red/region-Napa/`
`DimensionNameCaseInsensi¬tive`	Sorts alphabetically by dimension name, giving no regard to capitalization.	`http://localhost/Con¬tentAssemblerRefApp/Con¬tent.aspx/region-Na¬pa/Wine-red/`

Canonical Identity value	Description	Example base URL (sort direction ascending)
`DimensionId`	Sorts by dimension ID.	`http://localhost/ContentAssemblerRefApp/Content.aspx/region-Napa/Wine-red/`
`DimensionValueName`	Sorts alphabetically by dimension value name. Capital letters are given precedence.	`http://localhost/ContentAssemblerRefApp/Content.aspx/region-Napa/Wine-red/`
`DimensionValueNameCaseInsensitive`	Sorts alphabetically by dimension value name, giving no regard to capitalization.	`http://localhost/ContentAssemblerRefApp/Content.aspx/region-Napa/Wine-red/`
`DimensionValueId`	Sorts by dimension value ID.	`http://localhost/ContentAssemblerRefApp/Content.aspx/Wine-red/region-Napa/`

SortDirection

`canonicalizer.Direction value`	Description	Example base URL (sorted by dimension ID)
`Ascending`	Sorts in an ascending order based on the `canonicalizer.Identifier`.	`http://localhost/ContentAssemblerRefApp/Content.aspx/region-Napa/Wine-red/`
`Descending`	Sorts in a descending order based on the `canonicalizer.Identifier`.	`http://localhost/ContentAssemblerRefApp/Content.aspx/Wine-red/region-Napa/`

Note: Canonicalizing the dimension and dimension value names in the misc-path changes the order in which they appear in the path-params section of the URL. For example, if Napa is configured to display before Red in the misc-path, the Napa dimension value ID displays before the Red dimension value ID in the path-params section.

Optimizing URLs for record detail pages

Using the URL Optimization API, you can include dimension names, dimension value names, and record properties in the misc-path of URLs for record detail pages.

Note: For dimensions to display properly in the URL, they must be configured to **show with record** and **show with record list**.

Before you begin optimizing URLs, create an `SeoUrlProvider`.

To optimize URLs for record detail pages:

1. Set the `RecordDetailsCommandFormatter` property in the constructor of the `SeoUrlProvider` object:

   ```
         public SeoUrlProvider()
            : base()
         {
           this.RecordDetailsCommandFormatter = RecordDetailsCommandFormat¬
   terFactory.Create();
         }
   ```

2. Use the `RecordDetailsCommandFormatterFactory` to instantiate a `RecordDetailsCom¬ mandFormatter`.
   For example:

   ```
      public static class RecordDetailsCommandFormatterFactory
      {
         public static RecordDetailsCommandFormatter Create()
         {
            RecordFormatter recordFormatter = new RecordFormatter();

         }

      }
   ```

3. Set a `UseDimensionNameAsKey` property on the `RecordDetailsCommandFormatter`.
   For example:

   ```
      public static class RecordDetailsCommandFormatterFactory
      {
         public static RecordDetailsCommandFormatter Create()
         {
            RecordFormatter recordFormatter = new RecordFormatter();

            recordFormatter.UseDimensionNameAsKey = true;
         }

      }
   ```

   Setting the `useDimensionNameAsKey` to `false` creates a key on the dimension ID numbers.

4. Set a `DimensionValuePathFormatters` property on the `RecordDetailsCommandFormatter`.
   For example:

   ```
      public static class RecordDetailsCommandFormatterFactory
      {
         public static RecordDetailsCommandFormatter Create()
         {
            RecordFormatter recordFormatter = new RecordFormatter();

            recordFormatter.UseDimensionNameAsKey = true;
           recordFormatter.DimensionValuePathFormatters = DimensionValuePath¬
   FormattersFactory.CreateForRecordDetailsCommand();

         }

      }
   ```

5. Use a `DimensionValuePathFormattersFactory` to instantiate a `DimensionValuePathFor¬ matters` object for the `RecordDetailsCommandFormatter`.

For example:

```
      public static Dictionary<string, IDimensionValuePathFormatter>
CreateForRecordDetailsCommand()
      {
         Dictionary<string, IDimensionValuePathFormatter> dimValPaths =
             new Dictionary<string, IDimensionValuePathFormatter>();

      }
```

6. Add a `DimensionValuePathFormatter` for each dimension that you intend to include in the URLs.

   Note: Use the `dimValPaths.Add` method call to key each `DimensionValuePathFor¬matter`. If you set the `useDimensionNameAsKeyDimensionValuePathFormatter` as `false` in step three, be sure to use the dimension ID instead of the dimension name.

   For example:

```
      public static Dictionary<string, IDimensionValuePathFormatter>
CreateForRecordDetailsCommand()
      {
         Dictionary<string, IDimensionValuePathFormatter> dimValPaths =
             new Dictionary<string, IDimensionValuePathFormatter>();

         DimensionValuePathFormatter wineTypeFormatter = DimensionVal¬
uePathFormatterFactory.Create(true, true, true);
        DimensionValuePathFormatter regionFormatter = DimensionValuePath¬
FormatterFactory.Create(false, false, true);
        DimensionValuePathFormatter wineryFormatter = DimensionValuePath¬
FormatterFactory.Create(false, false, true);

         dimValPaths.Add("Wine Type", wineTypeFormatter);
         dimValPaths.Add("Region", regionFormatter);
         dimValPaths.Add("Winery", wineryFormatter);

      }
```

   Note: If you have already followed the steps for optimizing navigation pages and want to reuse the dimension value path formatters and string formatters for your record detail pages, you can skip to step eleven.

7. Use a `DimensionValuePathFormatterFactory` to instantiate a `DimensionValuePathFor¬matter` object and add the following formatters:

Formatter	Description
`AppendRoot`	Specifies whether or not to append root dimension values to the URL. Set to `true` to append root dimension values.
`AppendAncestors`	Specifies whether or not to append ancestor dimension values to the URL. Set to `true` to append ancestor dimension values.
`AppendDimensionVal¬ue`	Specifies whether or not to append the selected or descriptor dimension values to the URL. Set to `true` to append ancestor dimension values.
`RootFormatter`	Sets the `IStringFormatter` to customize URL formatting for dimension roots in the URL. If the value is null, roots do not display in the URL.

Formatter	Description
`PathFormatter`	Sets the `IStringFormatter` to customize URL formatting for dimension, ancestor, and dimension value names. If the value is null, dimension, ancestor, and dimension value names do not display in the URL.

For example:

```
   public static class DimensionValuePathFormatterFactory
   {
      public static DimensionValuePathFormatter Create(bool appendRoot,
 bool appendAncestors, bool appendDimVals)
      {
         DimensionValuePathFormatter formatter = new DimensionValuePath¬
Formatter();
         formatter.AppendRoot = appendRoot;
         formatter.AppendAncestors = appendAncestors;
         formatter.AppendDimensionValue = appendDimVals;

         formatter.RootFormatter = StringFormatterFactory.CreateDefault¬
StringFormatter();
         formatter.PathFormatter = StringFormatterFactory.CreateDefault¬
StringFormatter();

         return formatter;

      }

   }
```

8. Use the `StringFormatterFactory` to instantiate an `IStringFormatter` object.
   For example:

```
   public static class StringFormatterFactory
   {
      public static IStringFormatter CreateDefaultStringFormatter()
      {
```

9. Instantiate a `RegexReplacementStringFormatter` to replace non-word characters.

```
   public static class StringFormatterFactory
   {
      public static IStringFormatter CreateDefaultStringFormatter()
      {
         // Convert non word characters to '-'
         RegexReplacementStringFormatter nonWordCharsToDashFormatter
           //character class subtraction, whitespace character excluding
 the unicode characters
              = new RegexReplacementStringFormatter(@"(?:[\W-[\u00C0-
\u00FF]])+", "-");

      }
```

10. Instantiate a second `RegexReplacementStringFormatter` to remove leading and trailing characters created by the `RegexReplacementStringFormatter nonWordCharsToDashFor¬ matter`.

```
   public static class StringFormatterFactory
   {
      public static IStringFormatter CreateDefaultStringFormatter()
      {
         // Convert non word characters to '-'
         RegexReplacementStringFormatter nonWordCharsToDashFormatter
           //character class subtraction, whitespace character excluding
 the unicode characters
              = new RegexReplacementStringFormatter(@"(?:[\W-[\u00C0-
\u00FF]])+", "-");

         // Trim leading and trailing '-' characters
         RegexReplacementStringFormatter trimDashCharsFormatter
            = new RegexReplacementStringFormatter(@"^-?([\w\u00C0-
\u00FF][\w-\u00C0-\u00FF]*[\w\u00C0-\u00FF])-?$", "$1");

        return new CompositeStringFormatter(nonWordCharsToDashFormatter,
 trimDashCharsFormatter);

      }
```

11. Optionally, configure the `recordFormatter` to include record properties in the record detail URLs.
    a) Add a `propertyKeys` property on the `recordFormatter`.
    b) Instantiate a list of `propertyKeys`.
    c) Add any record properties that you want to include in the URL string to the list of `propertyKeys`.

    For example:

```
   public static class RecordDetailsCommandFormatterFactory
   {
      public static RecordDetailsCommandFormatter Create()
      {
         RecordFormatter recordFormatter = new RecordFormatter();

         recordFormatter.UseDimensionNameAsKey = true;
        recordFormatter.DimensionValuePathFormatters = DimensionValuePath¬
FormattersFactory.CreateForRecordDetailsCommand();

         List<string> propertyKeys = new List<string>();
         propertyKeys.Add("P_Name");
         recordFormatter.PropertyKeys = propertyKeys;

      }
```

    This sample code adds the record name to the URL. For example, if an end-user navigates to the Alenquer record detail page, "Alenquer" displays in the misc-path.

Related Links

Preparing your dimensions on page 25
    If you intend to display dimensions or dimension values in your URLs, you must configure each of the dimensions to **Show with record** and **Show with record list**.

Preparing your properties on page 25
    If you intend to display record properties in your URLs, you must configure each property to **Show with record** and **Show with record list**.

Optimizing URLs for aggregate record detail pages

Using the URL Optimization API, you can include dimension and dimension value names in the URLs for aggregate record detail pages. These are configured separately from the optimizations for navigation pages.

Note: For dimensions to display properly in the URL, they must be configured to **show with record** and **show with record list**.

Before you begin optimizing URLs, create an `SeoUrlProvider`.

To optimize URLs for aggregate record detail pages:

1. Set the `AggregateRecordDetailsCommandFormatter` property in the constructor of the `SeoUrlProvider` object:

```
      public SeoUrlProvider()
         : base()
      {
        this.AggregateRecordDetailsCommandFormatter = AggregateRecordDe¬
tailsCommandFormatterFactory.Create();
      }
```

2. Use the `AggregateRecordDetailsCommandFormatterFactory` to instantiate an `Aggre¬gateRecordDetailsCommandFormatter`.
   For example:

```
   public static class AggregateRecordDetailsCommandFormatterFactory
   {
      public static AggregateRecordDetailsCommandFormatter Create()
      {
         RecordFormatter recordFormatter = new RecordFormatter();

      }

   }
```

3. Set a `UseDimensionNameAsKey` property on the `AggregateRecordDetailsCommandFormat¬ter`.
   For example:

```
   public static class AggregateRecordDetailsCommandFormatterFactory
   {
      public static AggregateRecordDetailsCommandFormatter Create()
      {
         RecordFormatter recordFormatter = new RecordFormatter();

         recordFormatter.UseDimensionNameAsKey = true;
      }

   }
```

   Setting the `useDimensionNameAsKey` to `false` creates a key on the dimension ID numbers.

4. Use a `DimensionValuePathFormattersFactory` to instantiate a `DimensionValuePathFor¬matters` object for the `AggregateRecordDetailsCommandFormatter`.
   For example:

```
      public static Dictionary<string, IDimensionValuePathFormatter>
CreateForAggregateRecordDetailsCommand()
      {
         Dictionary<string, IDimensionValuePathFormatter> dimValPaths =
             new Dictionary<string, IDimensionValuePathFormatter>();

      }
```

5. Add a `DimensionValuePathFormatter` for each dimension that you intend to include in the URLs.

   Note: Use the `dimValPaths.`Add method call to key each `DimensionValuePathFor¬matter`. If you set the `useDimensionNameAsKeyDimensionValuePathFormatter` as `false` in step three, be sure to use the dimension ID instead of the dimension name.

   For example:

```
      public static Dictionary<string, IDimensionValuePathFormatter>
CreateForAggregateRecordDetailsCommand()
      {
         Dictionary<string, IDimensionValuePathFormatter> dimValPaths =
             new Dictionary<string, IDimensionValuePathFormatter>();

         DimensionValuePathFormatter wineTypeFormatter = DimensionVal¬
uePathFormatterFactory.Create(true, true, true);
        DimensionValuePathFormatter regionFormatter = DimensionValuePath¬
FormatterFactory.Create(false, false, true);
        DimensionValuePathFormatter wineryFormatter = DimensionValuePath¬
FormatterFactory.Create(false, false, true);

         dimValPaths.Add("Wine Type", wineTypeFormatter);
         dimValPaths.Add("Region", regionFormatter);
         dimValPaths.Add("Winery", wineryFormatter);

      }
```

   Note: If you have already followed the steps for optimizing navigation pages and want to reuse the dimension value path formatters and string formatters for your aggregate record detail pages, you can skip to step eleven.

6. Use a `DimensionValuePathFormatterFactory` to instantiate a `DimensionValuePathFor¬matter` and add the following formatters:

Formatter	Description
`AppendRoot`	Specifies whether or not to append root dimension values to the URL. Set to `true` to append root dimension values.
`AppendAncestors`	Specifies whether or not to append ancestor dimension values to the URL. Set to `true` to append ancestor dimension values.
`AppendDimensionVal¬ue`	Specifies whether or not to append the selected or descriptor dimension values to the URL. Set to `true` to append ancestor dimension values.
`RootFormatter`	Sets the `IStringFormatter` to customize URL formatting for dimension roots in the URL. If the value is null, roots do not display in the URL.
`PathFormatter`	Sets the `IStringFormatter` to customize URL formatting for dimension, ancestor, and dimension value names. If the value is null, dimension, ancestor, and dimension value names do not display in the URL.

   For example:

```
   public static class DimensionValuePathFormatterFactory
   {
      public static DimensionValuePathFormatter Create(bool appendRoot,
 bool appendAncestors, bool appendDimVals)
      {
         DimensionValuePathFormatter formatter = new DimensionValuePath¬
Formatter();
         formatter.AppendRoot = appendRoot;
         formatter.AppendAncestors = appendAncestors;
         formatter.AppendDimensionValue = appendDimVals;

         formatter.RootFormatter = StringFormatterFactory.CreateDefault¬
StringFormatter();
         formatter.PathFormatter = StringFormatterFactory.CreateDefault¬
StringFormatter();

         return formatter;

      }

   }
```

7. Use the `StringFormatterFactory` to instantiate an `IStringFormatter` object.
   For example:

```
   public static class StringFormatterFactory
   {
      public static IStringFormatter CreateDefaultStringFormatter()
      {
```

8. Instantiate a `RegexReplacementStringFormatter` to replace non-word characters.

```
   public static class StringFormatterFactory
   {
      public static IStringFormatter CreateDefaultStringFormatter()
      {
         // Convert non word characters to '-'
         RegexReplacementStringFormatter nonWordCharsToDashFormatter
           //character class subtraction, whitespace character excluding
 the unicode characters
              = new RegexReplacementStringFormatter(@"(?:[\W-[\u00C0-
\u00FF]])+", "-");

      }
```

9. Instantiate a second `RegexReplacementStringFormatter` to remove leading and trailing characters created by the `RegexReplacementStringFormatter nonWordCharsToDashFor¬matter`.

```
   public static class StringFormatterFactory
   {
      public static IStringFormatter CreateDefaultStringFormatter()
      {
         // Convert non word characters to '-'
         RegexReplacementStringFormatter nonWordCharsToDashFormatter
           //character class subtraction, whitespace character excluding
 the unicode characters
              = new RegexReplacementStringFormatter(@"(?:[\W-[\u00C0-
\u00FF]])+", "-");

         // Trim leading and trailing '-' characters
         RegexReplacementStringFormatter trimDashCharsFormatter
            = new RegexReplacementStringFormatter(@"^-?([\w\u00C0-
\u00FF][\w-\u00C0-\u00FF]*[\w\u00C0-\u00FF])-?$", "$1");

```

```
        return new CompositeStringFormatter(nonWordCharsToDashFormatter,
 trimDashCharsFormatter);

      }
```

10. Optionally, configure the `recordFormatter` to include record properties in the aggregate record detail URLs.
    a) Add a `propertyKeys` property on the `recordFormatter`.
    b) Instantiate a list of `propertyKeys`.
    c) Add any record properties that you want to include in the URL string to the list of `propertyKeys`.

    For example:

```
   public static class AggregateRecordDetailsCommandFormatterFactory
   {
      public static AggregateRecordDetailsCommandFormatter Create()
      {
         RecordFormatter recordFormatter = new RecordFormatter();

         recordFormatter.UseDimensionNameAsKey = true;
        recordFormatter.DimensionValuePathFormatters = DimensionValuePath¬
FormattersFactory.CreateForAggregateRecordDetailsCommand();

         List<string> propertyKeys = new List<string>();
         propertyKeys.Add("P_Name");
         recordFormatter.PropertyKeys = propertyKeys;

      }
```

Related Links

Preparing your dimensions on page 25
If you intend to display dimensions or dimension values in your URLs, you must configure each of the dimensions to **Show with record** and **Show with record list**.

Preparing your properties on page 25
If you intend to display record properties in your URLs, you must configure each property to **Show with record** and **Show with record list**.

Configuring the path-param-separator

Using the `PathSeparatorToken` property in the `BaseSeoUrlProvider` constructor, you can configure the path-params-separator string.

The sample `SeoUrlProvider` provided with the URL Optimization API uses an underscore to separate the misc-path from the path-params in URLs. For example: `http://localhost/ContentAssem¬blerRefApp/Content.aspx/Wine-Red-Pinot-Noir/_/N-66w`

You can change the string using the `PathSeparatorToken` property.

To change the path-param-separator character:

1. Set the `PathSeparatorToken` property in the constructor of the `SeoUrlProvider` object:
   For example:

```
      public SeoUrlProvider()
         : base()
      {
         this.PathSeparatorToken = "_";
      }
```

2. Change the value of the `PathSeparatorToken`.
   For example:

   ```
   this.PathSeparatorToken = "^";
   ```

The new URL displays as: `http://localhost/ContentAssemblerRefApp/Content.as¬px/Wine-Red-Pinot-Noir/^/N-66w`

Note: Be aware that certain non-URL safe characters can cause problems when included in the path portion of the URL. Do not use any of these characters as the path-param-separator.

Related Links

Anatomy of an optimized Endeca URL on page 35
An optimized Endeca URL is made up of four configurable sections.

Characters that should be excluded from the URL path on page 58
Certain non-URL safe characters can cause problems if they are included in the path portion of the URL. These problems range from failure of IIS to load the page to loss of information encoded in the URL string.

About optimizing the path-params and query string

The URL Optimization API provides functionality for encoding path parameters and moving Endeca path parameters from the query string into the path-params section of the URL.

Moving Endeca parameters out of the query string

In order to create directory-style URLs, you can limit the number of parameters in the query string by configuring a list of Endeca parameters to move from the query string and into the path-params section of the URL. For example, the following URL has the Endeca parameters N, Ntk, Ntt, and Ntx in the query string:

```
http://localhost/ContentAssemblerRefApp/Content.aspx/Bor¬
deaux?N=4294966952&fromsearch=false&Ntk=All&Ntt=red&Ntx=mode%2bmatchallpar¬
tial
```

Using the URL Optimization API, you can move Endeca parameters into the path-params section of the URL. For example, the following URL includes the N and Ntt parameters in the base URL:

```
http://localhost/ContentAssemblerRefApp/Content.aspx/Bordeaux/_/N-
4294966952/Ntt-red?fromsearch=false&Ntk=All&Ntx=mode%2bmatchallpartial
```

Modifying Endeca parameter names and values

You can customize the display of Endeca parameter names and values in URLs.

For example, you can modify the N parameter to display as "Endeca":

```
http://localhost/ContentAssemblerRefApp/Content.aspx/Napa/_/Endeca-4294966952
```

Important: If you choose to modify the display of Endeca parameter names and values, you cannot integrate with the Sitemap Generator.

Encoding Endeca parameter values

In order to shorten URLs, the URL Optimization API allows base-36 encoding of Endeca parameter values.

For example, the following URL for Region > Napa contains the dimension value ID for Napa (4294966952):

`http://localhost/ContentAssemblerRefApp/Content.aspx/Napa/_/N-4294966952`

By encoding the N parameter, you can shorten the URL:

`http://localhost/ContentAssemblerRefApp/Content.aspx/Napa/_/N-1z141pk`

Note: Only the numeric Endeca parameters can be encoded:

- N
- Ne
- An
- Dn

Removing session-scope parameters

In order to simplify the URLs, session-scope parameters should be removed from the URL string and stored as session objects. This might include any parameters that do not change value during the session, such as the session ID or MDEX Host and Port values.

Passing non-Endeca parameters to the API

The URL Optimization API can also format arbitrary (non-Endeca) parameters in URLs.

Related Links

Anatomy of an optimized Endeca URL on page 35
An optimized Endeca URL is made up of four configurable sections.

Moving Endeca parameters out of the query string

You can use the URL Optimization API to create directory-style URLs by configuring a list of Endeca parameters to move from the query string and into the path-params section of the URL.

Before you begin optimizing URLs, create an `SeoUrlProvider`.

To move Endeca parameters from the query string and into the path-params section of the URL:

1. Set the `PathParameters` property in the constructor of the `SeoUrlProvider` object:
   For example:
   ```
         public SeoUrlProvider()
            : base()
         {
            this.PathParameters = PathParameterFactory.Create();
         }
   ```
2. Use the `PathParameterFactory` to instantiate a list of Endeca parameters to move out of the query string and into the path-params section of the URL.
   For example:
   ```
      public static class PathParameterFactory
      {
   ```

```
      public static List<string> Create()
      {
         List<string> pathParams = new List<string>();
         pathParams.Add("R");
         pathParams.Add("A");
         pathParams.Add("An");
         pathParams.Add("Au");
         pathParams.Add("N");
         pathParams.Add("No");
         pathParams.Add("Np");
         pathParams.Add("Nu");

         return pathParams;

      }
   }
```

Modifying Endeca parameters

You can customize the display of Endeca parameter names and values in URLs.

Important: If you choose to modify the display of Endeca parameter and value names, you cannot integrate with the Sitemap Generator.

Before you begin optimizing URLs, create an `SeoUrlProvider`.

If you customize the display of Endeca parameter names or values in the URL, you must convert them back to their original state to be correctly processed by the MDEX Engine.

To modify the display of Endeca parameter names or values:

1. Set the `ParameterMapModifier` property in the constructor of the `SeoUrlProvider` object:
   For example:

   ```
         public SeoUrlProvider()
            : base()
         {
            this.ParameterMapModifier = new NParameterMapModifier();
         }
   ```

2. Instantiate an instance of the `IParameterMapModifier.`
   For example:

   ```
   private class NParameterMapModifier : IParameterMapModifier
         {

          public void ModifyOnConstruct(IDictionary<string, UrlParam> seri¬
   alizedCommands)
           {

           }
           public void ModifyOnDeconstruct(IDictionary<string, UrlParam>
   serializedCommands)
           {
   ```

3. Use the `ModifyOnConstruct` method to choose an Endeca parameter and to customize its display.

For example:

```
private class NParameterMapModifier : IParameterMapModifier
      {
       public void ModifyOnConstruct(IDictionary<string, UrlParam> seri¬
alizedCommands)
        {
           if (serializedCommands.ContainsKey("N"))
           {
              UrlParam nValue = serializedCommands["N"];
              UrlParam xValue = new UrlParam("x", nValue.Value);

              serializedCommands.Remove("N");
              serializedCommands.Add(xValue.KeyName, xValue);

           }
        }
   }
```

4. Use the `ModifyOnDeconstruct` method to return the parameter to its original state before being passed to the Endeca MDEX Engine.
   For example:

```
private class NParameterMapModifier : IParameterMapModifier
      {

       public void ModifyOnConstruct(IDictionary<string, UrlParam> seri¬
alizedCommands)
        {
           if (serializedCommands.ContainsKey("N"))
           {
              UrlParam nValue = serializedCommands["N"];
              UrlParam xValue = new UrlParam("x", nValue.Value);

              serializedCommands.Remove("N");
              serializedCommands.Add(xValue.KeyName, xValue);

           }
        }

        public void ModifyOnDeconstruct(IDictionary<string, UrlParam>
serializedCommands)
        {
           if (serializedCommands.ContainsKey("x"))
           {
              UrlParam xValue = serializedCommands["x"];
              serializedCommands.Remove("x");
            serializedCommands.Add("N", new UrlParam("N", xValue.Value));

           }

        }

      }
```

Encoding Endeca parameters

You can use the URL Optimization API to apply base-36 encoding to numeric Endeca parameters.

You must create an `SeoUrlProvider` before completing this procedure.

Only the numeric Endeca parameters can be encoded:

- N
- Ne
- An
- Dn

To encode numeric Endeca parameters:

1. Set the `ParameterEncoders` property in the constructor of the `SeoUrlProvider` object:
   For example:

```
      public SeoUrlProvider()
         : base()
      {
         this.ParameterEncoders = ParameterEncodersFactory.Create();
      }
```

2. Use the `ParameterEncodersFactory` to instantiate a list of numeric Endeca parameters to encode.
   For example:

```
   public static class ParameterEncodersFactory
   {
      public static IDictionary<string, IEncoder> Create()
      {

         Dictionary<string, IEncoder> encoders = new Dictionary<string,
 IEncoder>();
         encoders.Add("N", NumericBaseEncoder.Base36Encoder);
         encoders.Add("Ne", NumericBaseEncoder.Base36Encoder);
         encoders.Add("An", NumericBaseEncoder.Base36Encoder);

         return encoders;

      }
   }
```

Removing session-scope parameters

In order to simplify the URLs, session-scope parameters should be removed from the URL string and stored as session objects.

This might include any parameters that do not change value during the session, such as the session ID or MDEX Host and Port values. For example, the following URL contains information about the the MDEX Host and Port:

`http://localhost/ContentAssemblerRefApp/Content.aspx/N=0&eneHost=local¬
host&enePort=15002`

You can remove the MDEX Host and Port values from the URL and store them as session objects. The resulting URL is simplified:

`http://localhost/contentAssemblerRefApp/Content.aspx`

The following procedure provides instructions for removing the MDEX Host and Port values from the URL, but this procedure can be adapted as necessary to remove other session-scope parameters.

To remove the MDEX Host and Port values from the URL and store them as session attribute values:

1. Set the MDEX Host and Port as session attributes using the following code:

```
string eneHost = “localhost”;
Session[“eneHost”] = eneHost;

string enePort = “15002”;
Session[“enePort”] = enePort;
```

2. Retrieve the session attribute values using the following code:

```
 string eneHost = (string) Session["eneHost"];
 string enePort = (string) Session["enePort"];
```

About passing non-Endeca parameters to the API

The URL Optimization API can format arbitrary (non-Endeca) parameters in URLs.

Using functionality provided by the Endeca RAD Toolkit for ASP.NET, you can add arbitrary parameters to the URL, either for all URLs produced by a `UrlManager` or on a per-URL basis. For details, please refer to the *RAD Toolkit for ASP.NET Developer's Guide*.

By default, the additional parameters are added to the query string. You can move parameters to the path-params section of the URL in the same way as Endeca parameters, by adding them to the `PathParameters` list in your custom `UrlProvider`. Be aware that certain non-URL safe characters can cause problems when included in the path portion of the URL. If these characters are present in either the name or possible values of a parameter, do not move that parameter to the URL path.

Related Links

Moving Endeca parameters out of the query string on page 54

You can use the URL Optimization API to create directory-style URLs by configuring a list of Endeca parameters to move from the query string and into the path-params section of the URL.

Characters that should be excluded from the URL path

Certain non-URL safe characters can cause problems if they are included in the path portion of the URL. These problems range from failure of IIS to load the page to loss of information encoded in the URL string.

These problems may occur even if the characters are URL-encoded. Therefore, you should ensure that the following unsafe characters do not appear in the name or value of any parameter that is set to display in the path:

percent sign	%
ampersand	&
forward slash	/
backward slash	\
question mark	?
less than sign	<

greater than sign	>
colon	:
asterisk	*
plus	+

You should also avoid situations where the path-param-separator character appears in the URL path. The sample `SeoUrlProvider` uses the underscore character ("_") as the separator character. If there is a possibility that a parameter name or value may include the separator character, configure your custom `UrlProvider` to use a different separator, or do not include that parameter in the path.

Related Links

Configuring the path-param-separator on page 52

Using the `PathSeparatorToken` property in the `BaseSeoUrlProvider` constructor, you can configure the path-params-separator string.

Using the SeoUrlProvider with your application

Once you have created and configured your custom `SeoUrlProvider`, you need to integrate the `UrlProvider` into you application.

To integrate your `UrlProvider` into your application:

1. Locate your custom `SeoUrlProvider`.
2. Add your `SeoUrlProvider` to the shared code folder of your application.
   For example, the Content Assembler reference application uses an `App_Code` directory:
   `C:\Endeca\ContentAssemblerAPIs\RAD Toolkit for ASP.NET\`*`version`*`\reference\ContentAssemblerRefApp\App_Code`
3. If your application uses a `Web.config` file (or a similar configuration file) to configure `Url¬Providers`, update the `Web.config` file for the new `SeoUrlProvider`.

   For information about using a `Web.config` to register a `UrlProvider`, please see the *Endeca RAD Toolkit for ASP.NET Developer's Guide*.
4. Clear your Web browser cache.

   If you do not clear your Web browser cache, the browser may determine that the content of a page has not changed and display the old, unoptimized URL.
5. Restart IIS.

Chapter 8

Integrating with the Sitemap Generator

The Sitemap Generator creates an index of your Web site based on information stored in your MDEX Engine, not information stored on your application server. Because of this, you need to ensure that the URLs produced by the Sitemap Generator match the URLs in your application. To make certain that the URLs match, you need to configure the Sitemap Generator's `urlconfig.xml` file to make the same customizations to URLs that the URL Optimization API configurations are making.

The Sitemap Generator urlconfig.xml file

The Sitemap Generator uses an XML configuration file that must mirror your URL configurations in order to output a sitemap that matches your Web application.

The Sitemap Generator creates a site map by issuing a single bulk query against the MDEX Engine to retrieve the necessary record, dimension, and dimension value data. It uses this information to build an index of pages. The formatting of the URLs it creates is controlled by the `urlconfig.xml` file located in the `conf` subdirectory of your Sitemap Generator installation directory. For example: `C:\Endeca\SEM\SitemapGenerator\<version>\conf`

To ensure that the URLs in the sitemap are consistent with the URLs produced by the URL Optimization API, any configurations made in with the URL Optimization API must be mapped and configured appropriately in the Sitemap Generator's `urlconfig.xml` file.

Sample SeoUrlProvider and sample urlconfig.xml mapping

In order to integrate your URL optimizations with the Sitemap Generator, you must mirror these customizations in the Sitemap Generator `urlconfig.xml` file. This section provides mappings between the sample SeoUrlProvider and the sample `urlconfig.xml` Sitemap Generator configuration file.

The following table maps the objects and properties used in the sample `SeoUrlProvider` to their equivalent beans and properties in the sample `urlconfig.xml` file. Any changes outside the scope of these sample files, for example custom beans or custom `DimensionValuePathFormatters`, are not captured in this table.

For those cases where an object maps to more than one bean in the `urlconfig.xml` file, a list of all relevant beans is provided.

Navigation Command Formatter

URL Optimization API sample SeoUrlProvider		Sitemap Generator Sample `urlconfig.xml` Equivalents	
Object	**Property**	**Bean ID**	**Property**
`NavigationCom¬mandFormatter`		`navStateFormat¬ter`	
	`UseDimension¬NameAsKey`		`useDimensionNameAsKey`
	`DimensionValuePath¬Formatters`		`dimLocationFormatters`

Navigation Command Canonicalizer

URL Optimization API sample SeoUrlProvider		Sitemap Generator sample `urlconfig.xml` equivalents	
Object	**Property**	**Bean ID**	**Property**
`NavigationCom¬mandCanonicaliz¬er`		`navStateCanoni¬calizer`	
	`DimensionValueCol¬lectionCanonicaliz¬er.Identity`		`sortByName`, `sortByDi¬mension`, `ignoreCase`
	`DimensionValueCol¬lectionCanonicaliz¬er.Direction`		`ascending`

Record Details Command Formatter

URL Optimization API sample SeoUrlProvider		Sitemap Generator sample `urlconfig.xml` equivalents	
Object	**Property**	**Bean ID**	**Property**
`RecordDetailsCom¬mandFormatter`		`erecFormatter`	
	`UseDimension¬NameAsKey`		`useDimensionNameAsKey`
	`DimensionValuePath¬Formatters`		`dimLocationFormatters`
	`PropertyKeys`		`propertyKeys`
	`PropertyFormatter`		`propertyFormatter`

Aggregate Record Details Command Formatter

URL Optimization API sample SeoUrlProvider		Sitemap Generator sample `urlconfig.xml` equivalents	
Object	**Property**	**Bean ID**	**Property**
`AggregateRecord¬ DetailsCommand¬ Formatter`		`aggrERecFormat¬ ter`	
	`UseDimension¬ NameAsKey`		`useDimensionNameAsKey`
	`DimensionValuePath¬ Formatters`		`dimLocationFormatters`
	`PropertyKeys`		`propertyKeys`
	`PropertyFormatter`		`propertyFormatter`

Parameter Encoder

URL Optimization API sample SeoUrlProvider		Sitemap Generator sample `urlconfig.xml` equivalents	
Object	**Property**	**Bean ID**	**Property**
`ParameterEn¬ coders`		`seoUrlFormatter`	
	`NumericBaseEn¬ coder.Base36En¬ coder`		`urlParamEncoders`

Path Parameters

URL Optimization API sample SeoUrlProvider		Sitemap Generator sample `urlconfig.xml` equivalents	
Object	**Property**	**Bean ID**	**Property**
`PathParameters`		`seoUrlFormatter`	`pathParamKeys`

Dimension Value Path Formatter

URL Optimization API sample SeoUrlProvider		Sitemap Generator sample `urlconfig.xml` equivalents	
Object	**Property**	**Bean ID**	**Property**
• `wineTypeFormat¬ ter` • `regionFormat¬ ter` • `wineryFormat¬ ter`		• `wineTypeFormat¬ ter` • `regionFormat¬ ter` • `wineryFormat¬ ter`	

URL Optimization API sample SeoUrlProvider		**Sitemap Generator sample `urlconfig.xml` equivalents**	
Object	**Property**	**Bean ID**	**Property**
		• `flavorsFormat¬ter` • `vintageFormat¬ter`	
	`AppendRoot`		`appendRoot`
	`AppendAncestors`		`appendAncestors`
	`AppendDimensionVal¬ue`		`appendDescriptor`
	`RootFormatter`		`rootStringFormatter`
	`PathFormatter`		`dimValStringFormat¬ter`
	`Separator`		`separator`

Once you have created custom `DimensionValuePathFormatters` for your `SeoUrlProvider`, you need to replace the dimension formatters in the Sitemap Generator `urlconfig.xml` file with formatters that are specific to your application.

String Formatters

Remember: **URL Optimization API sample SeoUrlProvider**		**Sitemap Generator sample `urlconfig.xml` equivalents**	
Object	**Property**	**Bean ID**	**Property**
`CompositeString¬Formatter`		`defaultStringFor¬matterChain`	
	`recordFormat¬ter.PropertyFormat¬ter`		
	`recordFormat¬ter.PropertyFormat¬ter`		
	`formatter.RootFor¬matter`		
	`formatter.PathFor¬matter`		
`RegexReplace¬mentStringFormat¬ter`		`wineTypeFormatter`	

Remember: URL Optimization API sample SeoUrlProvider		Sitemap Generator sample `urlconfig.xml` equivalents	
Object	**Property**	**Bean ID**	**Property**
	`wineTypeToWineFor¬matter`		`rootStringFormatter`

Related Links

Canonicalization configuration options on page 43
: The `canonicalizer.Identifier` and `canonicalizer.Direction` properties on the `IDimensionValueCollectionCanonicalizer` object determine the method and direction of dimension ordering for navigation page URLs.

Adding custom dimensions to the Sitemap Generator configuration on page 65
: In order for dimensions to display in the URLs produced by the Sitemap Generator, they must be specified in the `<QUERY_FIELD_LIST>` in the Sitemap Generator's `conf.xml` file.

Example Sitemap Generator integration on page 68
: This section provides an example of a modified `SeoUrlProvider` and the equivalent changes that must be made to the Sitemap Generator URL configuration file.

About using regular expressions in string formatters with the Sitemap Generator

The Sitemap Generator uses Java syntax for regular expressions, which differs slightly from the format of regular expressions in the .NET Framework.

For example, the default string formatter included in the sample `SeoUrlProvider` uses an expression similar to the following to replace non-word characters with dashes:

```
[\W-[\u00C0-\u00FF]]+
```

The equivalent expression in Java is as follows:

```
[\W&&[^\u00C0-\u00FF]]+
```

If you use a `RegexReplacementStringFormatter` to format the URL path, take care to ensure that you are specifying the equivalent Java expression for the matching string formatter bean in the Sitemap Generator configuration file. For more details about the format of regular expressions in Java, please refer to the documentation from Sun at *http://java.sun.com/docs/books/tutorial/essential/regex/*.

Adding custom dimensions to the Sitemap Generator configuration

In order for dimensions to display in the URLs produced by the Sitemap Generator, they must be specified in the `<QUERY_FIELD_LIST>` in the Sitemap Generator's `conf.xml` file.

The `<QUERY_FIELD_LIST>` in the `conf.xml` of the Sitemap Generator is configured for the Endeca wine data set. Before you can generate a sitemap for your own data set, you need to specify your own dimensions to the `<QUERY_FIELD_LIST>`.

To specify dimensions in the Sitemap Generator `<QUERY_FIELD_LIST>`:

1. Open the `conf.xml` file located in the `\conf` subdirectory of your Sitemap Generator installation directory.
   For example: `C:\Endeca\SEM\SitemapGenerator\`*`version`*`\conf`
2. Locate the `<QUERY_FIELD_LIST>`.
   For example:

```
<!-- additional elements deleted from this sample -->
<QUERY_FIELD_LIST>
   <QUERY_FIELD>P_Name</QUERY_FIELD>
   <QUERY_FIELD>Wine Type</QUERY_FIELD>
   <QUERY_FIELD>Region</QUERY_FIELD>
   <QUERY_FIELD>Winery</QUERY_FIELD>
   <QUERY_FIELD>Vintage</QUERY_FIELD>
   <QUERY_FIELD>P_Winery</QUERY_FIELD>
   <QUERY_FIELD>Flavors</QUERY_FIELD>
   <QUERY_FIELD>Designation</QUERY_FIELD>
</QUERY_FIELD_LIST>
<!-- additional elements deleted from this sample -->
```

3. Replace the existing wine data dimensions with dimensions specific to your application.

For more information about the Sitemap Generator `conf.xml` file, please refer to the *Endeca Sitemap Generator Developer's Guide*.

Related Links

Example Sitemap Generator integration on page 68
This section provides an example of a modified `SeoUrlProvider` and the equivalent changes that must be made to the Sitemap Generator URL configuration file.

Modifying the root query

If you have modified the root query for your application using the Endeca RAD Toolkit for ASP.NET, you need to make the same changes to the Sitemap Generator configuration.

You must make these changes in the Sitemap Generator `conf.xml` file and the Sitemap Generator `urlconfig.xml` file.

Note: For information using the Endeca RAD Toolkit for ASP.NET to modify root queries, please see the *Endeca RAD Toolkit for ASP.NET Developer's Guide*.

To modify the Sitemap Generator root query settings:

1. Open the `conf.xml` file located in the `conf` subdirectory of your Sitemap Generator installation directory.
   For example: `C:\Endeca\SEM\SitemapGenerator\`*`version`*`\conf`
2. Locate the `<MDEX_ENGINES>` settings.
   For example:

```
<!-- additional elements deleted from this sample -->
<MDEX_ENGINES>
  <ENGINE>
```

```
      <HOST>localhost</HOST>
      <PORT>15000</PORT>
      <ROOT_QUERY><![CDATA[N=0]]></ROOT_QUERY>
  </ENGINE>
</MDEX_ENGINES>
<!-- additional elements deleted from this sample -->
```

3. Modify the `<ROOT_QUERY>` value so that it matches your application's root query settings.
   For example:

```
<!-- additional elements deleted from this sample -->
<MDEX_ENGINES>
  <ENGINE>
      <HOST>localhost</HOST>
      <PORT>15000</PORT>
       <ROOT_QUERY><![CDATA[N=0&Nr=OR(Wine Type:Red)]]></ROOT_QUERY>
  </ENGINE>
</MDEX_ENGINES>
<!-- additional elements deleted from this sample -->
```

Note: The value for `ROOT_QUERY` must be specified in the Presentation API URL format. For more information, see the "Endeca Presentation API URL Query Syntax" section of the *Endeca Basic Development Guide* or the *Endeca Developer's Guide for .NET*.

4. Open the `urlconfig.xml` file located in the `conf` subdirectory of your Sitemap Generator installation directory.
   For example: `C:\Endeca\SEM\SitemapGenerator\`*`version`*`\conf`
5. Locate the `queryBuilder` bean.
   For example:

```
<!-- additional elements deleted from this sample -->
  <bean id="queryBuilder" class="com.endeca.soleng.urlformatter.basic.Ba¬
sicQueryBuilder">

    <property name="queryEncoding">
      <value>UTF-8</value>
    </property>

    <property name="baseUrlENEQuery">
      <null/>
    </property>

    <property name="baseNavigationUrlENEQuery">
      <null/>
    </property>

    <property name="baseERecUrlENEQuery">
      <null/>
    </property>

    <property name="baseAggrERecUrlENEQuery">
      <null/>
    </property>

    <property name="defaultUrlENEQuery">
      <null/>
    </property>
  </bean>
<!-- additional elements deleted from this sample -->
```

6. Modify the `queryBuilder` bean so that all of the `UrlENEQuery` settings match your application settings.
   For example:

```
<!-- additional elements deleted from this sample -->
  <bean id="queryBuilder" class="com.endeca.soleng.urlformatter.basic.Ba¬
sicQueryBuilder">

    <property name="queryEncoding">
      <value>UTF-8</value>
    </property>

    <property name="baseUrlENEQuery">
      <null/>
    </property>

    <property name="baseNavigationUrlENEQuery">
      <value><![CDATA[N=0&Ns=P_Price|1&Nr=8020]]></value>
      <null/>
    </property>

    <property name="baseERecUrlENEQuery">
      <null/>
    </property>

    <property name="baseAggrERecUrlENEQuery">
      <value>An=0</value>
      <null/>
    </property>

    <property name="defaultUrlENEQuery">
      <value>N=0</value>
    </property>

  </bean>
<!-- additional elements deleted from this sample -->
```

Related Links

Example Sitemap Generator integration on page 68
: This section provides an example of a modified `SeoUrlProvider` and the equivalent changes that must be made to the Sitemap Generator URL configuration file.

Example Sitemap Generator integration

This section provides an example of a modified `SeoUrlProvider` and the equivalent changes that must be made to the Sitemap Generator URL configuration file.

There are sample Sitemap Generator integration files located in the `Sample\SitemapGeneratorIntegration` subdirectory of your URL Optimization API installation directory:

- The `SeoUrlProvider` located in the `Sample\SitemapGeneratorIntegration\UrlProvider` subdirectory
- The Sitemap Generator files `conf.xml` and `urlconfig.xml` located in the `Sample\SitemapGeneratorIntegration\SitemapGenerator` subdirectory

You can either review the files included in the `SitemapGeneratorIntegration` subdirectory, or manually make the changes described in this section.

SeoUrlProvider

Make the following changes to the sample `SeoUrlProvider`:

- Uncomment the `vintageFormatter` (line 144)
- Instantiate the `designationFormatter` (line 145)
- Uncomment adding the `vintageFormatter` (line 150)
- Add the `designationFormatter` to the `Dictionary<string, IDimensionValueFormat¬ter>` (line 151)
- Instantiate the `designationFormatter` (line 178)
- Add the `designationFormatter` to the `Dictionary<string, IDimensionValueFormat¬ter>` (line 183)
- Add the property key for `P_Winery` (line 229)
- Change Canonicalization Identity to `DimensionValueName` (line 357)
- Change Canonicalization Direction to `Descending` (line 358)

Sitemap Generator `urlconfig.xml` file

In order to mirror the changes to the SeoUrlProvider, make the following changes to the Sitemap Generator `urlconfig.xml` file (located in the `\conf` subdirectory of your Sitemap Generator installation directory):

- Uncomment the `vintageFormatter` (line 172)
- Add a reference to the `designationFormatter` to the `navStateFormatter`'s `dimLocation¬Formatters` (line 173)
- Change `sortByDimension` to false (line 195)
- Change `ascending` to false (line 196)
- Add a reference to the `designationFormatter` to the `erecFormatter`'s `dimLocationFor¬matters` (line 271)
- Add `P_Winery` to the property keys of the `erecFormatter` (line 278)
- Add a `designationFormatter`

Sitemap Generator `conf.xml` file

In order for the new dimension "Designation" to display in the URLs produced by the Sitemap Generator, "Designation" must also be added to the `<QUERY_FIELD_LIST>` in the Sitemap Generator's `conf.xml` file (Lines 89-98). The `conf.xml` file is located in the `conf` subdirectory of your Sitemap Generator installation directory).

Integrating the URL configuration files with the Sitemap Generator

Once you have customized the `urlconfig.xml` and `conf.xml` files for your application, you can use them with the Sitemap Generator to create an accurate sitemap.

To use the URL configuration files with the Sitemap Generator:

1. Locate the `conf.xml` and `urlconfig.xml` files that you customized for your Sitemap Generator integration.
2. Copy your `conf.xml` and `urlconfig.xml` files to the `conf` subdirectory of your Sitemap Generator installation directory.
   For example: `C:\Endeca\SEM\SitemapGenerator\<version>\conf`

For more information about the Sitemap Generator, please refer to the *Endeca Sitemap Generator Developer's Guide*.

Related Links

Example Sitemap Generator integration on page 68
This section provides an example of a modified `SeoUrlProvider` and the equivalent changes that must be made to the Sitemap Generator URL configuration file.

Index